Model driven design and data integration in semantic web information systems

Model Driven Design and Data Integration in Semantic Web Information Systems

Kees van der Sluijs

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

van der Sluijs, Kees

Model Driven Design and Data Integration in Semantic Web Information Systems / by Kees van der Sluijs.
Eindhoven: Technische Universiteit Eindhoven, 2012. Proefschrift.

A catalogue record is available from the Eindhoven University of Technology Library
ISBN: 978-90-386-3134-9
NUR: 980
Keywords: Interoperability / Heterogeneous Databases / Information networks / Multimedia Information Systems / Hypertext/Hypermedia
C.R. Subject Classification (1998): D.2.12, H2.5, H3.4c, H5.1, H5.4

SIKS Dissertation Series No. 2012-12
The research reported in this dissertation has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Cover design: Ilse Weisfelt
Printed by University Press Facilities, Eindhoven, the Netherlands.

Model Driven Design and Data Integration in Semantic Web Information Systems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op dinsdag 15 mei 2012 om 16.00 uur

door

Kornelis Antonius Martinus van der Sluijs

geboren te Maasdriel

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. G.J.P.M. Houben
en
prof.dr. P.M.E. De Bra

Contents

Acknowledgements

My life as a PhD-student started in 2004. I was motivated by my Master's graduation project which had been a great experience and a wonderful time. Together with my fascination for science this lead me to ask my supervisor professor Geert-Jan Houben if becoming a PhD-student would be a possibility. Geert-Jan offered me this opportunity for which I am still grateful. It was not a disappointing experience, I generally liked most of the tasks that come with the job and I was especially happy with the amount of variation it offered; projects, papers, teaching, conferences. The experience has been very educational and it helped shaping my way of thinking.

During my fourth year as a PhD-student professor Paul De Bra offered me a three year job as the overall project manager of the European GRAPPLE project, based on a recommendation from Geert-Jan. Obviously, I seized this great opportunity and accepted. The job offered a lot of variation, which was again very different from my work as a PhD-student and brought many new experiences and insights. The project ended successfully and left me with a mostly positive feeling.

Unluckily, I didn't finish my PhD-thesis during the project. I clearly preferred working on the project with its many deadlines and the excitement this new research gave me. This did not compare to the relative boredom of working at the seemingly endless task of compiling all my published work into one book. Without any hard deadlines it was easier to postpone and store away the urgency of completing it somewhere in the back of my head.

At the end of the GRAPPLE-project I was confronted with a number of radical changes in my life, which brought me to rethink my future in many aspects including my work. I decided to leave academia and try something else. This didn't mean I wanted to abandon my PhD-thesis, but working on it proved more challenging than I thought. Part of this was caused by the energy cost of turning my life around, but it was also caused by a lack of positive motivation. The PhD-thesis is famous for being the one type of book for which the effort to write it is more than the combined effort of all its readers that will ever read it. And of course the fact that my work was ageing didn't help either. My main motivation was a negative one, I cannot stand quitting something that I started and intended to finish. However, postponing work is also not quitting and it seemed to work well for some time. This vicious circle was finally broken when, with help of some peer pressure, I realized this was going nowhere. I finally understood that the only way to finish this work was by cutting off all distractions and completely focus myself on this thesis. And that is what I did. This book is the end result. I feel proud, and relieved, for not quitting but finally finishing what I should have done long ago. I learned my lesson that postponing work I don't like is useless. If you want something done, do it now!

This preface cannot be concluded without thanking the many people that have helped in making this work possible. Special *thanks* goes out to Geert-Jan and Paul - I cannot thank you enough for making the entire process possible and the fact that you have never let me down. I also have to thank everyone I have worked with, all my co-authors, and especially Pieter Bellekens, my room mate with whom I shared most of my lunches for years and with whom I share a love for scientific and political news and humor. I also

would like to thank all the students I have tutored during their master thesis - it was an enjoyable experience and your work has been invaluable to me. Furthermore, I want to thank our secretary Riet and her always warm personal interest. I want to wish her all the best with her own hardships. Of course, I cannot leave out Ine, our other secretary, who just always enthusiastically does what needs to be done. Finally, I want to thank my entire PhD committee for their time and effort in reading, providing feedback, and being my opponents at my defense.

Personally, I want to thank all of my family and friends who have always supported me. I will not name you all, but you all know who you are. I couldn't have done this without you. I do need to especially thank Christiaan Tempelman, with whom I've had many interesting discussions and laughs for many years in the train and of course the fun I've had with him during years of fitness, running, festivals and when just having a drink - or two. I hope you'll finish your PhD thesis on heterogenous catalysis quicker than I did mine.

Chapter 1

Introduction

Nowadays the World Wide Web has become an integral part of our lives. The Web satisfies almost every information need one can think of and is available on an ever growing amount of devices. As a technology, the Web has been strongly evolving ever since its inception. Where previously the Web was about interconnected documents, this has now changed into an interactive Web in which users don't only consume but also actively produce content. The Web is now a vast network of interactive Web applications that allows interaction and exchange of data.

This thesis is based on technology that is envisioned as the next step in the evolution of the Web: the Semantic Web [Berners-Lee et al., 2001]. Its vision is that the Web is no longer a Web of data, or a solely a Web of applications. It is a Web of interlinked *knowledge*. It is envisioned that the Semantic Web enables exchange and coupling of heterogenous data on a Web scale. This opens up many new possibilities for Web applications, as it allows using local application specific data and combine and enrich it with external data.

The Semantic Web was envisioned as a part of the answer for data interoperability, i.e. most applications cannot exchange data because they model and use their own internal data structures with their own syntax, structure and meaning. However, Semantic Web languages have also deliberately been kept simple and generic. They are minimally prescribing in how to model your data and it allows authors of semantic web models and data semantically a large degree of freedom. This freedom was seen as a major condition for the Semantic Web to be successful by its designers. It was recognized that there is no universal schema or ontology that the entire world can agree on. Many applications do exist in the world that have a set of requirements for its underlying datamodel that is incompatible with that of others. Therefore instead of enforcing one particularly view on the world, the basic viewpoint in the Semantic Web world is to only provide a generic flexible datamodel and a way to express semantic and leave the actual model and vocabulary up to the developers. This datamodel, basically a directed graph, does provide a standardized solution for syntactic interoperability using one particular model.

In the rest of this introduction the motivation and research goals for this thesis will be discussed. Note that it will refer to concepts which will only be explained in details in Chapter 2.

1.1 Motivation

This thesis has two focal points. The first focal point is extending an existing data driven Model-Driven Web Engineering (MDWE) approach called Hera. This approach is based on the common design pattern that given a data source one wants to create a Web applications

around it. However, instead of looking at the traditional approach in Hera, i.e. creating rather static monolithic Web applications around a datasource with a static schema, we rather looked at the problem of creating a framework for dynamic Web applications that integrates evolving datasources that might have changing schemas. The old versions of Hera were not able to do adapt to those requirements. Therefore, we aimed for a complete redesign of Hera that would fit those requirements. The resulting framework is called Hera-S, where the 'S' stands for Semantic. It aims to improve flexibility in terms of reuse of data and functionality. This can be achieved by exploiting Semantic Web languages like RDF(S) or OWL.

The second focal point is data integration. Hera-S should not only provide reuse of singular monolithic datasources, but also the flexibility to integrate several data sources. The power to couple and mix data sources offers many opportunity of new and interesting applications. However, achieving data integration still leave semantic interoperability issues that need to be dealt with, i.e./ many sources are modeled differently and apply (often implicit) semantics to data values. We recognized that dealing semantic interoperability is a task that is applicable in a much broader sense than Hera-S alone. Therefore, next to investigating this issue in the Hera-S sense we also take a broader perspective and investigate how our solutions might be applicable in a wider range of Web applications. Therefore in a large second part of this thesis we will aim at developing a toolset for dealing with semantic interoperability and see if we can apply this toolset in a broad range of Web applications. If we are able to do this, we should also be able to show how interoperability improves on the possibilities of these Web applications.

1.2 Problems and Research Goals

In this dissertation we present a completely reworked MDWE method especially tailored towards Semantic Web techniques and interoperability of data, while maintaining its strengths in personalization, adaptation and simplicity. Moreover, we also present an approach for coupling semantic heterogenous Resource Description Framework (RDF) data sources and shows how this and other Semantic Web techniques can be applied to improve existing Web Information Systems (WIS) by enabling simpler reuse of external information and provide richer data retrieval by combining information sources.

For this purpose several questions have to be answered.

Question 1: How do MDWE-architectures have to be adapted to enable flexible and dynamic Semantic Web data integration?
We will first look at the literature on MDWE and identify their main features. We then look specifically at Hera-S as a MDWE and present a complete redesign of Hera-S, including a formal basis of the central part of Hera-S: the application model (see Chapter 3). The redesign is flexible in multiple ways. Data models can plugged in or changed during runtime while the same holds for the application model. Also, its design is modular, i.e. the data integration component is separate from the application component and the application component is separate from the presentation component. This allows exchanging any of these components for other specialized components and frameworks.

Question 2: How can MDWE benefit from data integration techniques?
Given the redesigned Hera-S we look how we can make data integration work, both on the data input side as on the user model side where data integration allows us to learn about the

user from external applications. Furthermore we look at applications of Hera-S' flexibility. For example, we see how we can apply aspect-orientation techniques to make designing and implementing additional cross-cutting features into existing Hera-S applications simpler. We also look if we can indeed put an specialized presentation generation component in front of the Hera-S application model component.

Question 3: How to Integrate semantic heterogenous RDF data?
We want to able to create Web applications in Hera-S over integrated set of RDF datasources. As data integration is a broader issue than in Hera-S alone we look at the generic issue at hand: how to integrate semantic heterogenous RDF data. Our central goal is in creating a generic toolset which provides several matching and mapping techniques for designers who want to integrate two specific data sources. We look both at pure syntactic techniques and semantic techniques using external datasources. Another potential source for this task is utilizing human computation. The resulting toolset is aimed to be useful for integration of structured semantic datasources and structureless elements like tags with structured semantic datasources.

Question 4: How can we apply data integration in Web applications in various domains?
Given the toolset that is the projected result of question 3, we see if and how we can utilize this toolset to achieve data integration in Semantic Web applications in different domains. We explore the particular needs and requirements for the applications in the various domains and see how our toolset can be used in solving their specific data integration needs.

Question 5: How can domain specific Web applications benefit from data integration?
Given that we achieve data integration in the various Web applications, we investigate how these applications can exploit that integrated data and achieve additional unique features. We look at various needs that exist for these Web applications and look for specific solutions. For these solutions we always try to keep in mind that many problems are shared across domains and where possible we try to find generally applicable solutions.

1.3 Outline of the Dissertation

First, in chapter 2 we discuss the background technologies that are relevant for this thesis and related work. Next to the evolution of the Web and developments like tagging, we discuss in more detail the Semantic Web, some languages associated with it and the evolution of linked data. We also describe the need for WIS and MDWE methods and describe some of the related methods next to Hera. In chapter 3 we present a complete redesign of the Hera framework, called Hera-S. Next to its architectural design we provide a formal definition of the application model and discuss some of the implementation details of the Hera-S engine which is called Hydragen. Then in chapter 4, as part of answering Question 2, we take a look at how a data integration engine can be put in front of Hera-S. In that chapter we also look at the flexibility of the Hera-S method as we explore an extension of Hera-S which allows the implementation of application cross-cutting features using aspect-orientation. Next to that we also look at using the external presentation engine AMACONT as the presentation engine together with Hydragen. The actual data

integration toolset called Relco is proposed in chapter 5 as an answer for question 3. Part of Relco is a mapping language to describe and execute data transformation of data instances between schemas. This mapping language is an extension of the SPARQL query language. Next to this mapping language Relco contains a set of matching techniques that can work together to help a designer find probable mapping candidates. Relco syntactically uses string matching technologies, semantically it allows exploiting external knowledge sources like word, and finally it enhances ordering the best matches by using context disambiguation and user feedback. Finally, in chapter 6 and 7 we look at a number of real world applications that were built using the Relco toolbox. There we answer question 4 by showing how data integration was achieved for those particular scenarios using Relco. In that same chapter we answer question 5 by exploring several advances that can be made using data integration in these applications as well as looking at particular issues we encountered and solved for those particular instances where we tried to learn generalized lessons for these issues where possible.

Chapter 2

Related Work

In this chapter we discuss the technologies that form the background for this thesis. First we discuss the development of the Web from its early stages into the current Web 2.0 and then the next step in its evolution, namely the Semantic Web. Following the Semantic Web Stack we review the main technologies and languages involved in the Semantic Web, like XML, RDF(S), OWL, SPARQL. We also look at developments like Linked Data and Tagging. Besides the Semantic Web we look at evolution of Web Information Systems and the development of Model Driven Web Engineering for structured Web application development. We look at some of the most important methods that have been created to date, namely WebML, OOHDM, UWE and OOWS.

2.1 Web Development

Since its original inception in 1991 [Berners-Lee et al., 1994], the World Wide Web has come a long way in its evolution. It first grew into a web of handcrafted text documents that were connected via links. When the Web became more popular and some organizations recognized its potential the first Web applications started to appear that generated the content of the Web pages from data that was stored in databases [Berners-Lee and Fischetti, 1999].

It took several years of maturing in which search technology radically improved and broadband internet connections were introduced, when the Web made a large step in its evolution. Web applications slowly turned from a consumption-only relation with its users into a producer-consumer relationship. Via the technological introduction of forums, blogs (e.g. Wordpress, Twitter) and wikis (e.g. Wikipedia), content sharing websites like Youtube and Flickr to explicitly social Web applications (e.g. MySpace, Facebook, LinkedIn) and the parallel development and views on openness in software development, the Web evolved into Web 2.0 [Millard and Ross, 2006]. Web 2.0 stands for interactive Web applications. So Web pages are no longer static entities that have to be loaded one by one. Instead browsers are now used as a cross-browser platform in which interactions will lead to instant changes within a page. Also a first notion of mashups and reuse of data has been introduced: RSS feeds allow making personalized news portals, Youtube movies and Flickr photos can be easily embedded in other Web application and the functionality of for example Google and Yahoo maps can be easily reused and integrated in your own application logic.

Web 2.0 is a great success and access to the Web is considered to be a fundamental need for everyone. Of course, Web technologies are also under continuous development. One

important development is the growing number of devices that are capable of using the Web. Mobile Internet-capable devices for example seem to have reached the maturity required for commercial success. This is due to two key ingredients: increased bandwidth, closing up the gap with common broadband Internet speeds, and widespread availability, mainly achieved by rapid deployment and price-drop of new and superior standards by mobile telecom operators (e.g. UMTS). Not coincidentally, advances in the capabilities of handheld and portable devices (e.g. mobile and smart phones, PDA's, portable game-consoles) have been equally large. More economical power-usage, increased graphical capabilities and heightened processing power have turned these devices into full-fledged Web clients. A similar development can be seen in the living room. An increasing number of people have a Web-capable game console. Also the television is nowadays hooked up to the Web, either via a set-top box, an HD media player or directly via its increasing network capabilities. The different capabilities of these devices force Web developers to rethink their Web application architecture.

A parallel software evolution in the World Wide Web has been the emergence of the Semantic Web [Berners-Lee et al., 2001]. The regular Web contains linked documents, which themselves are not machine-interpretable. This has a number of limitations. As most Web documents are generated from data in private databases and the data is not in a machine-processable form it is hard to find specific information. This can be partially solved by search engines that index the text documents and create indexes based on keyword occurrences. However, especially metadata, i.e. data about data, is missing. Consider for example the information need of a student for planning the cheapest acceptable trip for a scientific conference. The current manual way to do this is typically something like:

- Finding the location of the conference on the conference website.
- Finding cheap hotels in the conference's neighborhood.
- Looking for trusted reviews of that hotel on several hotel reviewing Websites for checking hotel's acceptability.
- Finding the airports closest to that conference site.
- Visit several airline companies that have an acceptable service for evaluating the cost and quality of all connection options from local airports to one of the closest target airports.
- Calculate costs for several (public) transportation options from and to the airports and the hotel.

This can be a very long and laborious process that is hardly automatable without semantic technology that enables the exchange of knowledge and logic. Automated answering of such a question includes coupling datasets from heterogenous Web applications that usually have very incompatible data formats. It also includes taking the user into account as the question 'what is acceptable' is a very personal one.

Another limitation is the lack of metadata. Instead of offering only the pure data in the form of Web pages and multimedia documents, also consider the need for data like 'who is the author of this data', 'when was this data created', 'what do others think of this data', 'what is the technical quality of this data?', etc. The regular Web does not contain the mechanisms for expressing this metadata.

2.2 Semantic Web

The Semantic Web does offer the mechanisms to solve these issues and therefore is a next step in the evolution of the Web.

The Web, as it is, is human-readable. It is machine-processable in the sense that machines can render pages, follow links and index pages based on word occurrence. However, it is not machine-interpretable in the sense that the text itself can be truly understood by machines. Semantic relationships of pages with others are not obvious for the machine. Furthermore, the context of documents is not clear either. This issue can be tackled by using the Semantic Web, because the Semantic Web is "an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." [Berners-Lee et al., 2001].

Figure 2.1 is a regularly used example that illustrates this issue. Suppose this is a web page about a disease in the Chinese language. For non-Chinese speakers the symbols themselves are meaningless, just like the text on every page will be meaningless to a computer. We can observe the markup but not the semantic interpretation. We could use semantic marking, for instance to indicate that the top element of the page is the name of the disease, and subsequent markings indicate the symptoms and the treatment of the disease. Further semantics relationships can indicate that administration of a drug is a disease treatment and that treatment alleviates symptoms. This would already give some semantic interpretation capabilities even though we still do not interpret the symbols themselves. For example, it allows for instance categorization of treatments, i.e. if other pages contain different forms of treatment.

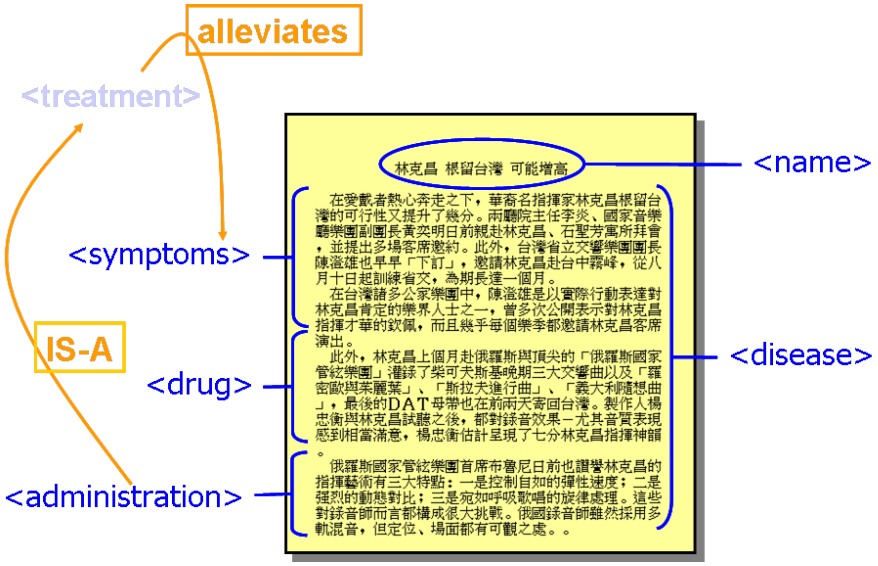

Figure 2.1: Meta-data illustration[a](1)

[a] http://www.slideshare.net/Frank.van.Harmelen/rdf-briefing

Note that current semantic web standards do not allow to express the true meaning of words as illustrated by 2.1. It for instance does not offer a standardized way to discuss diseases, symptoms or drugs, etc. However it does allow to express and structure information we do agree on. The standardized semantic web languages focus on expressing relationships like the 'IS-A' and class and subclass structuring of data. This can be used as a semantic basis that allows basic semantic reasoning using the meaning of these relationships. The semantic web languages can be used for defining vocabularies, e.g.

medicinal vocabularies that do express concepts like disease, symptom and treatment. It allows to express the concepts and the semantic relationship between these concepts but without expressing there explicit meaning.

For another illustration of the metadata issue, consider Figure 2.2. This is an image of a famous painting by Escher. However, this image cannot be automatically processed and interpreted. And even if it could, this would not reveal who the painter is of this work, what its title is, what the technical specifics are, how it relates to other works or what other people think about this picture. Next to the Escher painting you can see an example of metadata that one could want to record about this image.

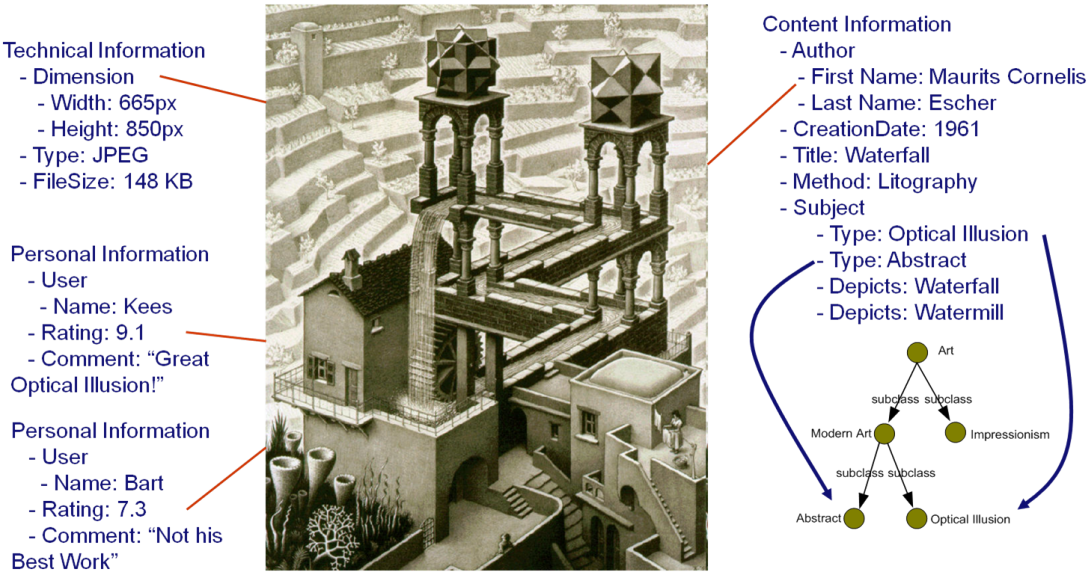

Figure 2.2: Meta-data illustration (2)

The usefulness of metadata becomes obvious as one sees how one can construct richer queries in the semantic web. In the regular Web one can use a keyword based search that works by matching the search string with term occurrences in textual Web pages. Structured metadata makes more complex queries possible. It allows answering queries like "which optical illusions in the late career of Escher get an average user rating of at least 8?". It allows for an enhanced browsing experience. One could for instance browse for "similar works" using external knowledge sources, which could be works with the same subjects, of the same artist, from the same time period or which are equally popular. Or for "more information about this artist" based on the author's full name. This is made possible without manual inclusion of links to other similar works, but by (semi-)automatically combining pieces of knowledge that are connected via links. For example, if two images of paintings have metadata that attribute the work to Escher and refer to an art thesaurus which describes Esher, these images are semantically linked.

The Semantic Web provides the technological foundation for structuring metadata and should eventually allow expressive query power on a Web scale as outlined in the previous paragraphs. This process is far from trivial [Shadbolt et al., 2006]. Making this technology a success on a Web scale means an ongoing effort in the standardization of the languages, adoption of the languages and ongoing process in supporting technologies. However, that this technology works is shown via running example applications as shown in Section 6.

2.2.1 Languages of the Semantic Web

The set of Semantic Web languages is organized in a hierarchical stack structure as shown in Figure 2.3 [Berners-Lee, 2003; Horrocks et al., 2005]. The technology of every layer of the stack builds upon the layer(s) below it.

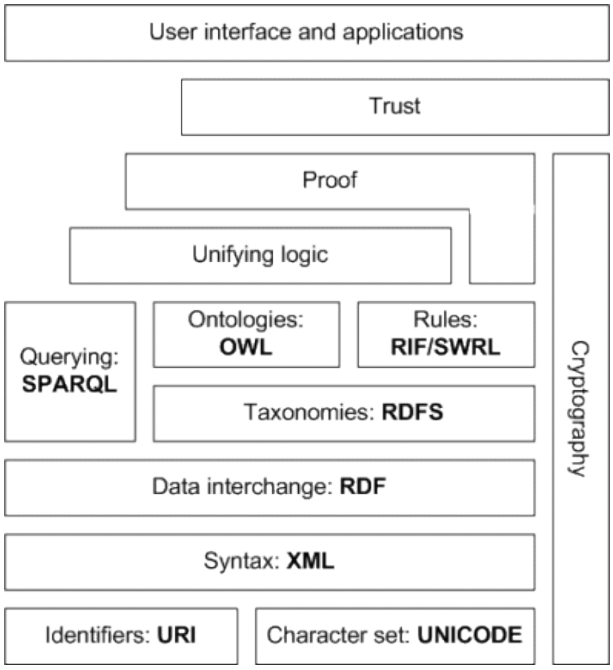

Figure 2.3: The Semantic Web Stack

The bottom layer contains the technology for writing text and resource identification. Unicode[1] is a standard for consistent text representation. Unicode has a clear multilingual focus, e.g. its character encoding allows the representation of over 100,000 characters.

A Uniform Resource Identifier (URI) uniquely identifies abstract or physical resources on the Web. The Uniform Resource Locator (URL) is a subset of the set of URIs, where a URL is a resource indication that also includes location details of how this resource can be looked up on the Web.

2.2.2 XML

The next layer in the Stack is syntactical. The most used format is Extensible Markup Language (XML)[2], however alternatives do exist and will be used in this thesis.

XML is a simple text-based data storage format. XML documents consist of a number of entities, which consist of nested element tags, which in turn can have attributes. The structure of an XML document can be formally defined using a DTD or an XML Schema[3]. URIs are used for the identification of documents and entities: each document and each entity has a unique URI which is normally built up with a namespace part (itself a URI) and a local name. Entities that have the same namespace URI are said to be in the same namespace.

XML is the most widely used serialization language in the Semantic Web. However, it is not the only syntax used for serialization. XML is a generic syntax that can be used for

[1] cf. `http://www.unicode.org/`
[2] `http://www.w3.org/XML/`
[3] cf. `http://www.w3.org/TR/xmlschema-0/`

encoding arbitrary datastructures, but especially the RDF/XML syntax is criticized for being verbose and hard to read by humans. Therefore a number of additional serialization syntaxes have been introduced that are especially tailored to RDF. The most often used XML alternative is N3[4], which as a number of derivatives, namely Turtle[5] (more on turtle in Section 2.2.3) , Ntriples[6], and Trig[7]. However note that these syntaxes are specifically used as a RDF serialization, while this does not apply to XML.

2.2.3 RDF(S)

Originally designed as a metadata representation language, RDF grew into the data interchange format that it is today in the Semantic Web Stack. RDF is a modeling approach for defining data using triples which have a subject, predicate and object. A collection of RDF statements represents a labeled directed graph that (possibly) has multiple edges and loops (i.e. a multigraph).

The main entity in RDF is called a resource. Resources can be anything, from Web pages and other online documents to physical entities and even abstract concepts. Therefore, RDF uses the URI mechanism to uniquely identify these resources, and the RDF language consists of statements that are made about the resources. Each statement consists of:

1. A subject: the resource about which the statement is made.

2. A predicate: an attribute or characteristic of the subject. In RDF, predicates are special forms of resources and thus also have URIs.

3. An object: the value of the predicate. An object can either be a resource or a literal value like a string or integer. RDF uses XML Schema data types to denote the type of these literal values.

Figure 2.4 is an example RDF graph describing my homepage. In RDF/Turtle syntax this is written as:

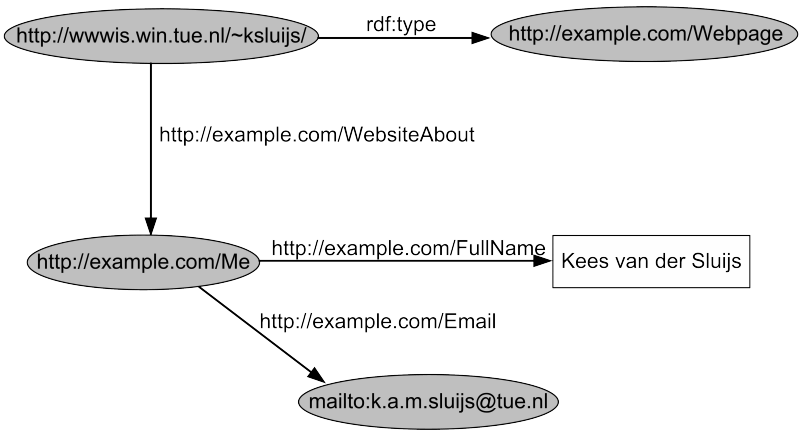

Figure 2.4: RDF Graph example

[4]cf. `http://www.w3.org/DesignIssues/Notation3`
[5]cf. `http://www.w3.org/TeamSubmission/turtle/`
[6]cf. `http://www.w3.org/TR/rdf-testcases/#ntriples`
[7]cf. `http://www4.wiwiss.fu-berlin.de/bizer/TriG/`

```
  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
  @prefix ex: <http://www.example.com/>.

<http://wwwis.win.tue.nl/~ksluijs/>
  rdf:type ex:Webpage;
  ex:WebsiteAbout ex:Me .

ex:Me ex:FullName "Kees van der Sluijs";
      ex:Email <mailto:k.a.m.sluijs@tue.nl> .
```

In addition to these basic constructs, RDF also supports more advanced constructs:

- **Blank nodes** Blank nodes are anonymous subjects or objects. Blank nodes can be either used for resources without URI identifier or for intermediate values, e.g. used for multi-values.
- **Typed literals** Literal values can have a datatype. RDF relies on the rich XML datatype [Biron et al., 2004] mechanism for this.
- **Containers and Collections** RDF allows grouping resources using several constructs.
- **Reification** With reification it is possible to describe properties of RDF triples.

While XML can have locally defined attributes, every RDF triple is a globally defined statement. But note that even though RDF uses URIs for global identification, this doesn't mean that every URI is HTTP accessible and can be looked up for more information. For example, the URIs in Figure 2.4 from the example.com domain are not HTTP accessible (the example.com domain is actually reserved for documentation).

For more specific information about RDF refer to [Manola and Miller, 2004].

Resource Description Framework Schema (RDFS) is a semantic extension of RDF that provides a structure for grouping and relating resources and their properties. RDFS provides a class hierarchy structure. Classes are resources themselves and classes can be defined as subclasses of other classes. Resources can also be defined as members (named instances) of a class. Semantically the class hierarchy defines the class extension of a class as the set of the instances of that class, where all instances of a class are recursively also instances of its superclass.

RDFS also defines a property hierarchy structure. The property hierarchy is a similar structure as the class-hierarchy, except for that property instances are typically used as predicates in triples. Furthermore, properties have the basic RDFS facilities rdfs:domain and rdfs:range for defining their domain and range. Domains are used to state that any resource that has a given property is an instance of one or more classes. Ranges are used to state that the values of a property are instances of one or more classes.

For both hierarchies the rdf:type is used to state that a resource is an instance of either a class or a property.

Other often used utility properties defined by RDFS are rdfs:label for defining human readable names for resources, rdfs:comment for human readable descriptions of resources and rdf:value that has no formal semantics, but is typically used to describe the main value (if there is one) of a structured value.

Figure 2.5 is an example of a graph containing RDFS. It builds upon Figure 2.4, but now with RDFS structure. It is stated 'me' is an instance of the class 'man', which on its

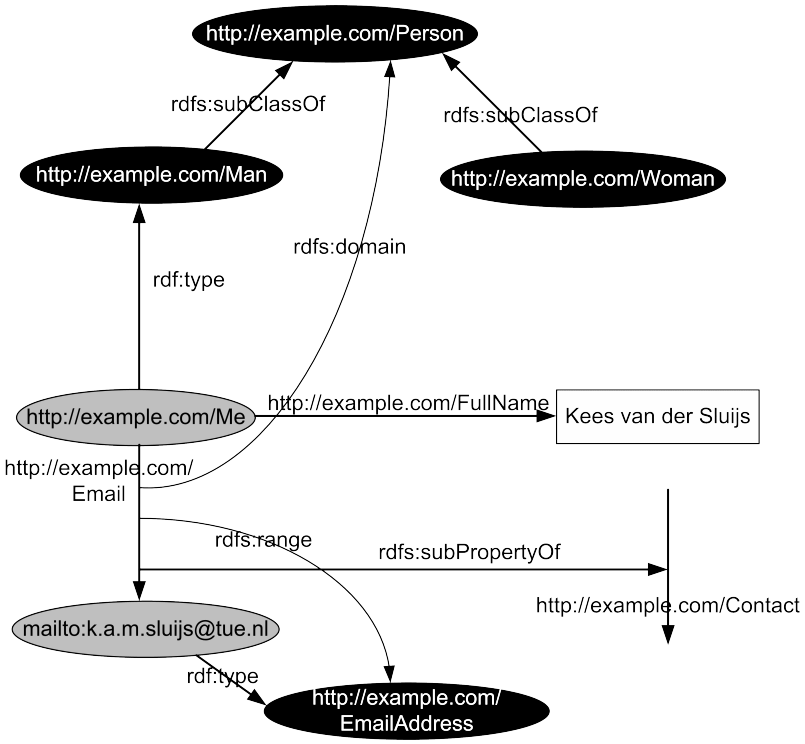

Figure 2.5: RDFS Graph example

turn is a subclass of 'person', as is 'woman'. It also shows that 'email' is a subproperty of 'contact'. Moreover, 'email' is a property that has 'person' as its domain class and 'email address' as its range class.

Applying the class extension to Person leads to concluding that I am also a 'Person'. This can also be concluded based on the fact that the 'Email' property is used with 'me' as a subject, because 'Email' has domain 'Person'. That 'mailto:k.a.m.sluijs@tue.nl' is an instance of 'EmailAddress' could also have been concluded based on the range of the 'Email' property.

Note that the Semantic Web languages are descriptive and not prescriptive. We could for example state that the university TU/e has the 'Email' address 'mailto:info@tue.nl', even if the subject of the property as used in this way does not have the domain 'Person'. This does not make the Resource Description Framework (Schema) (RDF(S)) description of the property 'Email' wrong, nor its usage in this case. Instead applying reasoning this will infer that TU/e is a 'Person'. If this is not desired we should not have described the property 'Email' to have 'Person' as a domain class.

Also note that the RDFS schema definition and its RDF instances are typically mixed. This is not usual in many data models, but for RDF it is.

For more information on RDFS refer to [Brickley and Guha, 2004].

Turtle

Terse RDF Triple Language (Turtle)[Beckett and Berners-Lee, 2011] is an RDF syntax format designed to be compact, easy to use, and easy to understand by humans. Although not an official standard, it is widely accepted and implemented in RDF toolkits.

In Turtle, each part of the triple is written down as a full URI or as a qualified name (using namespace prefixes). In our examples, we will mostly use the latter form, for brevity. For example,

```
my:car rdf:type cars:Opel .
```

denotes the RDF statement "my car is of type Opel". 'my:' and 'cars:' in this example are namespace prefixes that denote the vocabulary/ontology from which the term originates. Turtle also introduces the predicate 'a' as a shortcut for the 'rdf:type' relation.

```
my:car a cars:Opel .
```

denotes the same RDF statement as the first example. In order to list several properties of one particular subject, we can use the semi-column to denote branching.

```
my:car a cars:Opel ;
       my:color "White" .
```

denotes two statements about the car: that it is of type 'Opel', and that its color is white (denoted by a string literal value in this example). When the value of a property is itself an object with several properties, we can denote this by using square brackets ('[' and ']'). In RDF terms, such brackets denote a blank node (which can be thought of as an existential qualifier). For example:

```
my:car a cars:Opel ;
       my:hasSeat [
  a my:ComfortableSeat ;
          my:material "Cloth"
       ] .
```

This denotes that the car has a seat which is "something" (a blank node) which has as its type 'my:ComfortableSeat' and has cloth as the material.

Next to graphical figures we will mostly us Turtle as the RDF syntax in this thesis, because of its readability.

2.2.4 OWL

Web Ontology Language (OWL) is an RDF vocabulary that extends RDFS. It extends RDFS by defining semantically well-defined relations that can be used for knowledge representation. Using OWL one can define "ontologies". An ontology is defined as "an explicit specification of a conceptualization" [Gruber, 1993]. This basically means that an ontology is a formal way of describing (some aspects of) the real world.

The OWL constructs are based on the formal foundation of description logics [Baader et al., 2003]. The attractive property of description logic is that it ensures computable decidability (i.e. all reasoning steps on DL data finish within finite time). There are three distinct (so-called) species of OWL, namely OWL Lite, OWL DL and OWL Full. The three species of OWL balance expressive power against the time complexity of reasoning: OWL Full has more expressive power than OWL DL, which has in turn more expressive power than OWL Lite, but reasoning over OWL Lite is considerably less time complex than reasoning over OWL DL or OWL Full. Note that the different species of OWL are proper subsets of each other in the sense that every OWL Lite ontology is a valid OWL DL ontology, which is in turn a valid OWL Full ontology.

- OWL Lite encompasses RDFS and contains the main (simple version) constructs of OWL. The main OWL Lite language constructs are (in)equality (e.g. equivalentClass, sameAs, differentFrom), property charasteristics (e.g. inverseOf, TransitiveProperty), property restrictions (e.g. allValuesFrom, (0 or 1) cardinality) and versioning (e.g. versionInfo, DeprecatedClass).

- OWL DL is the subset of OWL that corresponds to description logic. Besides the OWL Lite constructs, OWL DL also contains class axioms (e.g. disjointWith, equivalentClass), boolean class expressions (e.g. unionOf, complementOf) and arbitrary cardinality. OWL DL restricts the use of its language constructs by requiring type separation (properties cannot also be classes or individuals and classes cannot also be individuals or properties). These restrictions are needed to ensure computable decidability of the language.

- OWL Full has equivalent language constructs as OWL DL and only differs from OWL DL by not having the type separation requirement. Removing this restriction means that reasoning over OWL Full data might not be a decidable and therefore not computable. However, it does simplify the use of OWL DL constructs for authors.

As an example, consider the following OWL/Turtle fragment that describes the class of 'Adult'. In the example 'Adult' is defined as the intersection of the Person class with the set of resources that an age in the range of 18-65 years.

```
:Adult a owl:Class ;
    owl:equivalentClass [ rdf:type owl:Class ;
        owl:intersectionOf ( :Person
            [ a owl:Restriction ;
              owl:onProperty :hasAge ;
              owl:someValuesFrom [ a rdfs:Datatype ;
                  owl:onDatatype xsd:integer ;
                  owl:withRestrictions (
                      [ xsd:maxExclusive "65"^^xsd:integer ]
                      [ xsd:minExclusive "18"^^xsd:integer]
                      )
                  ]
              ]
           )
        ] .
```

Note that different definitions of 'Adult' are possible and that the definition is descriptive. This means that a person without an hasAge attribute may still be labeled as an 'Adult' without problem. Even if the person does have an hasAge attribute with value 70 the qualification 'Adult' is fine unless there is a functional restriction on the hasAge attribute (i.e. defining that a person has exactly one age). In that case a reasoner will find a logical error.

For more information on OWL refer to [Bechhofer et al., 2004]. Also note that OWL is superseded by OWL 2, which extends OWL with additional features like keys, property chains, richer datatypes, dataa ranges, etc, and it provides additional profiles. For more information on OWL 2 refer to [Hitzler et al., 2009].

2.2.5 SPARQL

The Web nature of RDF implies that its data is inherently of a distributed nature. A query language in which data of different sources can be unambiguously queried is therefore essential. SPARQL Protocol and RDF Query Language (SPARQL) is the W3C query language recommendation for this purpose.

The SPARQL syntax is partly inspired by SQL by using a SELECT, FROM, WHERE syntax, extended with a FILTER operator.

Consider the following example SPARQL query:

```
PREFIX ex:    <http:/example.com/>
SELECT ?name, ?email
WHERE
 {  ?x ex:FullName ?name .
    ?x ex:Email ?email .
    FILTER regex(?name, "kees", "i" )
 }
```

This query returns a set of (name, email) pairs where the name contains the string "kees". The graph patterns are given with a Turtle syntax. The filter expression allows regular expressions, indicated with the keyword 'regex'. The regular expression "kees" will for instance match with "Kees van der Sluijs". The 'i' parameter indicates that the regular expression matches are made case-insensitive.

The resultset of a SPARQL query contains every possible solution for the query pattern in the RDF dataset. For SELECT queries the result set is a solution sequence that contains bindings for all variables in the SELECT clause. This result is itself not RDF, but in a dedicated XML format [Beckett and Broekstra, 2008] . An alternative is using CONSTRUCT queries instead of SELECT queries. The CONSTRUCT clause contains an RDF template with variables. The resultset of a CONSTRUCT query is an RDF graph that is the union of the instantiation of the template for all query solutions.

The SPARQL query language contains numerous additional operators, modifiers and query forms. For this and more information about SPARQL refer to [Prud'hommeaux and Seaborne, 2008].

2.2.6 Rules

The current status of the rules-layer in Figure 2.3 is that there is a rule language proposal called Semantic Web Rule Language (SWRL) [Horrocks et al., 2004a] from 2004. However, current work is focused on development of Rule Interchange Format (RIF) [Kifer and Boley, 2010].

SWRL is a specific rule language proposal that is based on RuleML [Hirtle et al., 2006], which itself has a Datalog foundation. SWRL has a high-level abstract syntax for Horn-like rules that consists of antecedent and consequents. The syntax is expressed in terms of OWL. Specific example of SWRL rules can be found in 5.4.3.

RIF is not an actual rule language, but an intermediate language that allows the exchange of rules. It was chosen as such because a multitude of rule languages exist. Known rule languages fall in three broad categories, namely first-order, logic-programming and action rules. Because of the extreme diversity between these categories RIF focused on extendable dialects that allow to express a specific rule language in one of the dialect languages. For more information on RIF refer to [Kifer and Boley, 2010].

2.2.7 Linked Open Data

The goal of the Semantic Web is not only about providing a data format that enables data exchange, but also about putting data on the Web that links to other data. This interlinked Web of data is envisioned to be an incredibly large knowledge source.

In order to achieve this, the Linked Open Data initiative [Berners-Lee, 2006] has been brought forward. RDF data is Linked Data if it fulfils four additional rules:

1. Use URIs as names for things.
2. Use HTTP URIs so that these things can be referred to and looked up.
3. When someone looks up a URI, provide useful information using the standards such as RDF.
4. Include links to other related URIs in the exposed data for improving discovery of other related information on the Web.

Using these rules the Semantic Web becomes a huge queryable knowledge source, in effect breaking down the classical data silos. This opens up possibilities for cross fertilization of data, which has several advantages. Combining knowledge for instance allows for richer applications. Given RDF exposure of airline databases, hotel data, and local (public) transport options, one could for example relatively simply build a total trip planning service. Reconciling separate data silos would make this much more complex and leaves the issue of making custom interfaces to connect the data.

For this vision to be meaningful and interesting for research, the Semantic Web of linked data needs to have a critical mass, i.e. its usage needs to be of Web-like proportions to be interesting. Therefore, the exposed datasets need to be big to have a considerable amount of usable data and there need to be many datasets exposed and interconnected in order to find meaningful cross fertilizations.

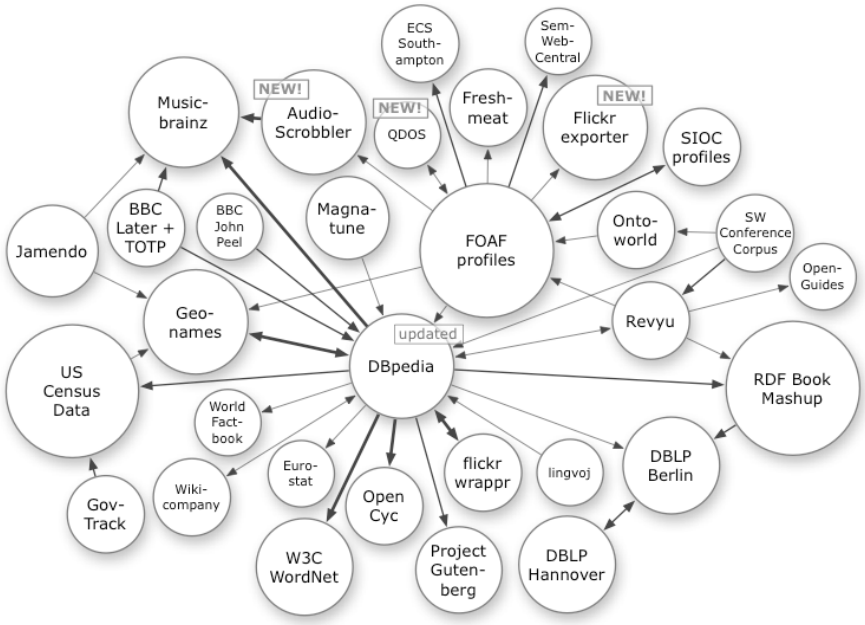

Figure 2.6: Linked Data cloud February 2008

Current developments point towards the Web size of the Semantic Web linked data becoming a reality. The number of datasets is growing quickly. This is illustrated by graphs of the linked datasets. Figure 2.6 shows the graph of the linked open datasets in February 2008, while Figure 2.7 shows the graphs of the linked open datasets in August 2011[8].

The datasets in the linked open datasets are in Web tradition about all sorts of domains and vary in size between tens of thousands of triples and billions of triples.

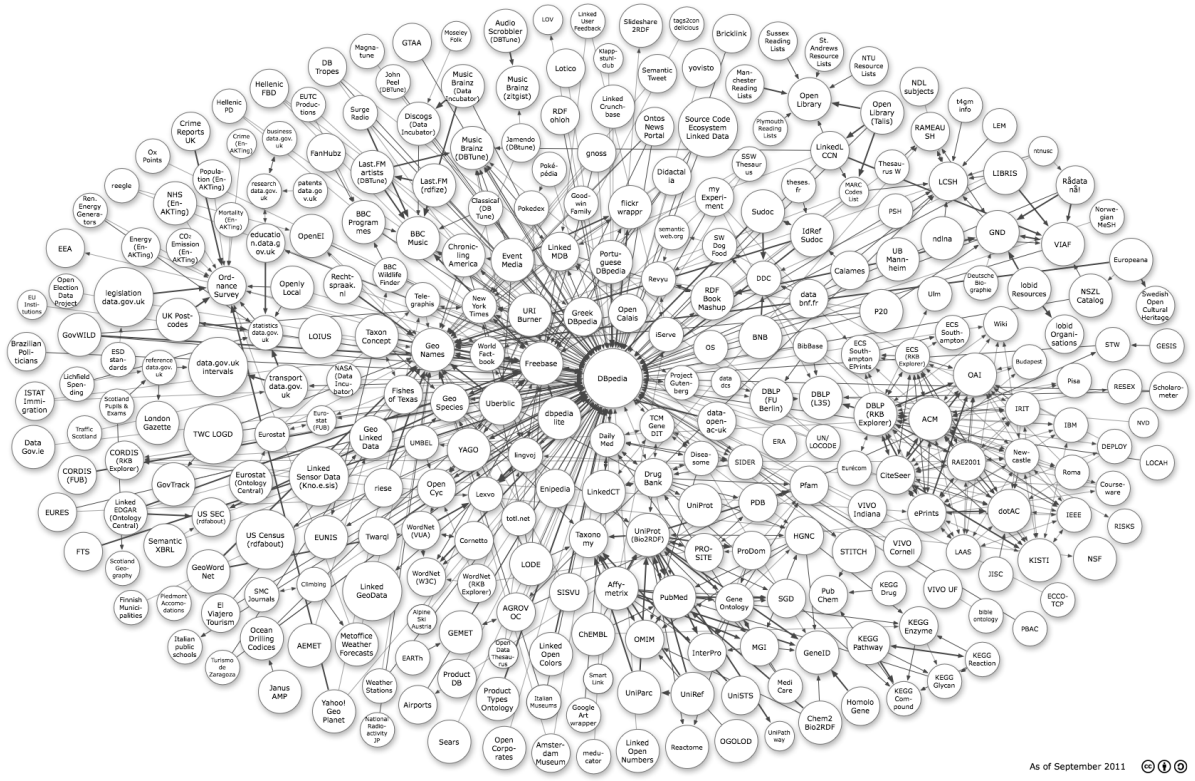

Figure 2.7: Linked Data cloud August 2011

Consider the following examples of datasets that are in the linked open dataset, for getting an idea of the diversity of the domains:

- Wordnet a popular English lexical database, has been available in RDF format[9] for some time now
- DBpedia[10] is a dataset abstracted from Wikipedia that contains millions of concepts in the RDF format.
- The Open Directory Project (DMOZ)[11] database, one of the Web's largest human-edited Web directories, is also accessible in RDF[12]
- DBPL is a large data set that contains bibliographic information about almost one million scientific papers and is also available in RDF[13].

[8]Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. `http://lod-cloud.net/`
[9]`http://www.w3.org/2006/03/wn/wn20/`
[10]`http://dbpedia.org/`
[11]The abbreviation is from directory.mozilla.org, its original domain name.
[12]`http://rdf.dmoz.org/`
[13]`http://dblp.l3s.de/d2r/`

- Uniprot[14] is a scientific database that stores protein sequences in RDF format.
- MusicBrainz is a large database that contains data about most artists and their albums and is available in RDF[15].

These are just a few examples that demonstrate the availability of many and large RDF datasets, many more are available on the Web[16]. This increased availability of semantic information leads to exciting new possibilities: semantics-based search engines , automated agents crawling and extracting relevant information, complex querying, (automatically) combining information from different sources, etc.

2.2.8 Tagging

Tagging is the process of a user assigning a simple keyword or a short sentence fragment to a resource (e.g. a multimedia document) and is an often used facility in Web 2.0 applications. It is therefore a special kind of annotation process. Tagging is used as a rudimentary form of metadata that can be used for searching and navigation features like tag-clouds. Even though tagging itself is not part of the Semantic Web, a part of its metadata goal is similar. We will show ways to exploit tagging data and relate it to Semantic Web sources in Chapter 6. Therefore, we will discuss tagging is some more detail here.

An inherent property of tagging is that it is schemaless. This means that the user does not need any prior knowledge of the domain for annotating resources, in the sense that the user does not need to know a given conceptual (knowledge) structure from which a concept needs to be chosen to annotate the resource. This simplicity is what makes tagging inherently easy to do for regular users.

Some popular systems that employ tagging mechanisms are for instance Del.icio.us [Golder and Huberman, 2006], Flickr[17] and Youtube[18]: these are systems that exploit collaborative tagging for web bookmarks, photos and videos respectively. In [Golder and Huberman, 2006] and [van Setten et al., 2006] it is observed that based on user behavior different kinds of tags can be distinguished. Tags that identify what (or who) the content is about are used the most by far. On the other hand, also tags that identify the kind of content, the owner, qualities or characteristics (for example regarding the photos or the cameras with which they were taken), and refinements are used. Moreover, users sometimes use unique tags for their own bookmarking purposes. Also in the research community tagging has received quite some attention. For example, in [Mika, 2005] a formal basis for collaborative tagging systems is laid out and both [Mika, 2005] and [Choy and Lui, 2006] describe how tagging can help in the search process. Another interesting paper is [Ames and Naaman, 2007], which explores the motivation of people for tagging photos. This paper provides evidence that the best incentive for users to using tagging facilities is a combination of personal and social motivator.

Tagging is a simpler than RDF for end-users as it is schema- and structureless, i.e. it doesn't require users to have knowledge about a particular structured vocabulary, nor forces them to design complex structures. However, the price for that simplicity is that the semantics of tags are unknown, e.g.:

[14] `http://dev.isb-sib.ch/projects/uniprot-rdf/`
[15] `http://dbtune.org/musicbrainz/`
[16] e.g. refer to [Heath, 2010].
[17] `http://www.flickr.com/`
[18] `http://www.youtube.com/`

- The user's intention of a specific tag is unknown.
- The context of the tag is unknown.
- The relationship between tags is unknown.
- Language ambiguity, e.g. homographs (words with different meanings), synonyms, syntax errors and word morphisms make it hard to parse tags.

2.3 Web Information Systems

The Web is *the* platform for ubiquitously accessible information, but also developed into an application platform with numerous applications of any kind [Gaedke and Meinecke, 2008]. However, even though the applications become increasingly complex, until recently most of the development process remained ad-hoc. Mixing document markup languages like HTML with server-side scripting languages like PHP, ASP and Java Servlets allow the creation of the typical Web applications that you can find currently on the Web. However a large gap remains between current ad-hoc programming and disciplined development approaches that allow for a good separation of concerns, i.e. that can address the right issues at the right level of abstraction. The Web poses unique requirements to applications in terms of usability, personalization, reusability and maintainability. As a result, many of the more complex web applications fail because of poor design and because its designers simply were not aware of the requirements to make their applications be accepted by the public. MDWE stands for a systematic approach for creating web applications, that allows the designer to manage the complexity and diversity that is inherent in such a task.

In the research field of Web Engineering several MDWE methods have been proposed, each with its own particular focus and its own strengths and weaknesses. Some of the most prominent methods include the Web Modeling Language (WebML) (Section 2.3.1), Object-Oriented Hypermedia Design Model (OOHDM) (Section 2.3.2), UML-based Web Engineering approach (UWE) (Section 2.3.3), and the Object-Oriented Web Solutions Approach (OOWS) (Section 2.3.4), which we will discuss more in detail in this section. What these methods share is that they dissect and describe Web applications on the following levels [Rossi et al., 2008]:

1. data/information, with issues relating to data representation.
2. navigation, with issues relating to navigation structure and behavior.
3. functionality, with issues relating to application functionality beyond navigation.
4. presentation/interface, with issues relating to interface and presentation design.

This extends traditional software engineering, which might typically place *functionality* as the core aspect of an application.

The first important notion is that of the subject matter of the application, i.e. what data or information is shown through the application. Most MDWE methods offer functionality for designing the information space of their application, e.g. via E/R-modeling (e.g. for data in relational databases), object-oriented approaches via UML or ORM, or using Semantic Web technology with languages like RDF. The information space is considered the basis of the application that is to be designed in most MDWE methods.

Secondly, MDWE allows designers to create the navigation structure and behavior of their application. In this context navigation is defined as a way to bring together (pieces of) information that is 'atomically' consumed (e.g. on one page). Navigation is typically expressed in terms of the underlying data of the application. It defines the things being navigated and the structure of the navigation space. Designing the navigation space is based on the user requirements. This means that the navigation space must be tailored towards its user base, typically by differentiating between specific users with their specific context and preferences by means of personalization. Most MDWE methods support the navigation specification process by means of graphical tools. Moreover, the use of well-defined models allows model checking techniques to detect logical errors in the specification like dead-ends (pages with no outgoing links) or unreachable pages.

Functional issues are built on top of the navigation structure. Functional behavior is basically all behavior that does not relate directly to the data navigation, even though it might influence it. The MDWE methods vary greatly in functional capabilities. One part of the application that is not about navigation is behavior within a navigation page. Since Web 2.0, Web applications are no longer static pages where navigation is the only type of behavior, but they also interact with the user. Other examples of functional behavior are workflows and activities. Consider for instance a shopping cart where products can be bought and checked out. This is clearly behavior that is beyond traditional navigation over the information set. Also the use of Web services to get external information, or to perform specific computations, is an example of the use of functional capabilities that a MDWE method could support.

The Web application will finally present its information to its user. Therefore, the MDWE method also has to describe how it is presented. One issue that some MDWE methods address is platform adaptation. An application will be typically translated to HTML, but also translations to for instance Wireless Markup Language (WML) and Synchronized Multimedia Integration Language (SMIL) have been reported in literature [Frasincar et al., 2006] as alternative platforms. Another issue is that of device adaptation. There are quite some presentation related differences between an application for a regular Web browser and one for a mobile phone or PDA. MDWE methods typically have an abstract specification of the presentation. This allows applications to be browser independent and forward compatible by using updates of the MDWE methods' implementing engine. It also allows designers to abstract from language-specific issues and allows designers to design applications for specific devices and platforms without having knowledge of the actual technologies and languages.

In chapter 3 we will discuss a MDWE method called Hera in detail. In this section we will shortly discuss some of the other most well-known methods, namely WebML in Section 2.3.1, OOHDM in Section 2.3.2, UWE in Section 2.3.3 and OOWS in Section 2.3.4. This will give a general idea of the common ideas and methodologies used in MDWE methods. We will also give short indications of how Hera differentiates from the other methods.

For a more comprehensive comparison between methods, please refer to [Schwinger et al., 2008].

2.3.1 WebML

WebML [Brambilla et al., 2008] is a mature web modeling language which has shown that MDWE methods can be commercially successful via its commercial tool implementa-

tion named WebRatio[19][Brambilla et al., 2010] which builds on the Eclipse framework[20]. According to [Ceri et al., 2002], WebML is a visual language for specifying the content structure of a Web application and the organization and presentation of such content in a hypertext. The WebML development process follows a repetitive cycle of requirements analysis, conceptual modeling of the application, implementation of the conceptual models and a testing and evaluation phase. Based on the outcome of these cycles the web application will finally be deployed and maintained. In the conceptual modeling phase the abstract application is defined, which is done in two steps: Data Modeling and Hypertext Modeling. The information space is specified in the data modeling phase of WebML. The data is designed using well-known ER-modeling techniques. This is in contrast with Hera's use of semantic web languages. Both modeling choices are of course valid; it depends on the specific situation and the wishes of the designer which approach is the most suitable for a specific application.

In the hypertext modeling phase the conceptual application is defined. In WebML the hypertext model defines the navigation structure between pages and page organization, but also functional specifications of e.g. services and content management operations, as well as presentation issues. WebML's hypertext model is a customized and particularly rich graphical language, which allows designers to add specific code bits on places where necessary. Hera's application model, in contrast, is written down in a RDF(S) vocabulary just like its data. Hera also has graphical builders for its application model, but these are not nearly as extensive as those of WebML. Where WebML is mostly focused on expressive graphical modeling, Hera is mostly focused on personalization and data selection and data reuse. WebML also does have an extension for context and user adaptation [Ceri et al., 2005].

WebML is extendable via model plugins. Throughout the years plugins for covering numerous issues have been added, e.g. with respect to context-awareness [Ceri et al., 2007], services [Manolescu et al., 2005], workflows [Brambilla et al., 2006], semantic web [Facca and Brambilla, 2007] as well as rich internet applications [Bozzon et al., 2006].

2.3.2 OOHDM

Object-Oriented Hypermedia Design Model (OOHDM) [Rossi and Schwabe, 2008] is one of the earliest web modeling approaches. The OOHDM development process discerns 5 steps, namely requirements gathering, conceptual design, navigational design, abstract interface design, and implementation.

In the requirements gathering phase the stakeholders of the applications are identified and the tasks that they must perform. Based on this information, scenarios are collected which describe the use cases in the form of User Interaction Diagrams (UID). The diagrams represent the interaction of the user and the application during execution of a task.

In the conceptual design the information space of the application is specified. OOHDM uses an extended version of UML to do that. The output of the requirements gathering phase serves as input for the conceptual design phase, i.e. the UIDs are used to initially derive a conceptual design (which can later be refined). Note that there is also a semantic web approach of OOHDM called Semantic Hypermedia Design Method (SHDM) [Schwabe et al., 2004] that, like Hera, is based on use of semantic web languages for the conceptual design.

[19] http://www.webratio.com
[20] http://www.eclipse.org/

The navigational design in OOHDM is considered to be a view on the conceptual design. This means that the objects (items) that the user navigates are not actually the conceptual objects, but other kinds of objects that are composed from one or more conceptual objects. The navigation objects are initially derived from tasks specified in the UIDs. Tasks in which a series of navigation objects needs to be visited are specified within in a certain context. In this way the user can follow different context paths to fulfill particular tasks and the same concept can end up in different navigational objects, which are separately defined, such that the particular context determines how a concept is presented in that context.

In the abstract interface design phase it is specified how the application is made perceptible to the user. The abstract interface in OOHDM focuses on the various types of functionality that can be played by interface elements with respect to the information exchange between the user and the application. The vocabulary used to define the abstract interface is established by an abstract widget ontology [de Moura and Schwabe, 2004] and can amongst others show elements to the user, collect mouse clicks and other events, and gather user information via form elements.

The final implementation phase in OOHDM is typically the rewriting of the models obtained via the method, to corresponding code on some particular platform, so the models only support the thinking process of the designer. However, for the SHDM part there is an engine that can directly execute the models, which is called HyperDE[Nunes and Schwabe, 2006].

2.3.3 UWE

UML-based Web Engineering approach (UWE) [Koch et al., 2008] is a web engineering approach completely based on UML. The UWE development process consists of phases for requirements engineering, analysis, design, and implementation. It also discerns three layers of an application, namely content, navigation and presentation.

In the requirements phase, UWE considers two levels of granularity. First, a rough description of the functional behavior of the application is drawn up with UML use cases. Second, a more detailed description of this use case is developed with UML activity diagrams in which responsibilities and actions of the users of the system are modeled.

In the analysis phase first a content model is described. This content model, expressed in a UML class diagram, captures the information space of the application. In a separate model also the user model data is modeled. The navigation structure is specified in the design phase, also specified in UML class-diagrams. In addition the navigation structure contains business logic rules. The classes in the diagram can be annotated with special semantic symbols to symbolize their function in the application, e.g. there are navigation classes, menus, access primitives, and process classes. UWE provides a facility to derive a first version of the navigation structure based on the content model, which can be later refined. The presentation is defined in classes that are associated with navigation classes in the navigation structure. Such a class is specified in terms of UI elements. The following UI elements are available in UWE: text, anchor, button, image, form and anchored collection.

UWE uses aspect-orientation for adaptive link hiding, adaptive link annotation, and adaptive link generation [Baumeister et al., 2005]. In order to use aspect-orientation the designer needs to (manually) put pointcuts in the model, where they want to integrate aspect considerations.

Current research in UWE is centered on rich internet application patterns[Koch et al., 2009] and modeling secure navigation[Busch et al., 2011].

2.3.4 OOWS

Object-Oriented Web Solutions Approach (OOWS) [Fons et al., 2008] is based on Model-Driven Architecture (MDA) as proposed by OMG[21]. OOWS is an extension of the OO-method [Pastor et al., 2001], where the OO-method approaches web modeling as "classic" software development. OO-methods provide a Platform-Independent Model (PIM) that defines a structural model, a dynamic model and a functional model. The structural model captures the system structure (its classes, operations, and attributes) and its relationships in a class diagram. The dynamic model describes sequence diagrams that express the communication between objects. And finally the functional model captures the semantics of state changes. OOWS extends the OO-method by providing a PIM for the user model, the navigation model and presentation model. Via automatic transformation these PIMs are translated to Platform-Specific Models (PSM) and from PSMs to application code. The application code is generated via a three-tier architectural style, namely the presentational tier (containing user interface components), the application tier (which contains functionality like business logic) and the persistence tier (that manages the data stores used in the application).

OOWS extends the OO-method so that it can express complete web applications. It does that by providing a PIM for the user model, the navigation model and the presentation model. The user models describe the users in a hierarchical way. On the top level of this hierarchy there are anonymous users, registered users and generic users. Navigation models are associated with users in the hierarchy and via inheritance this determines which users can access which part of the navigation. The navigation model specifies for each kind of user defined in the user model the view over the system in terms of classes, class attributes, class operations and class relationships. The presentation model depends on the links in the navigation model between system components and user types. For every such link the presentation is specified in terms of patterns. The basic patterns are information paging (i.e. breaking instances into logical blocks), ordering criteria and information layout (e.g. tabular or tree layouts). In the automatic transformation process these additional OOWS PIMs are orthogonally translated with the OO-method's PIMs to PSM and from PSMs to the complete web application code.

The OOWS also includes some extensions to deal with issues commonly encountered in modern web applications such as web requirements modeling. In this extension structural and behavioral requirements have been extended with navigational requirements. This can be achieved via a task taxonomy that specifies the tasks users should achieve when interacting with the Web application, activity diagrams that describe these tasks in more detail and information templates that describe information that is exchanged in each interaction point. Another extension is business process modeling. OOWS allows the designer to express the business processes that drive the web application. For this the designer needs to specify graphical user interfaces to launch and complete process activities, as well as the equivalent executable definition of the process. OOWS also implements an interface extension that allows interaction with semantic web tools. This interface is realized via two models. The first model specifies the system domain and the second model describes how external entities/agents should use the system functionality exposed in business settings.

[21] http://www.omg.org/mda

2.3.5 Related Fields

In this section we discussed approaches in the MDWE field. The MDWE is not an isolated field but is closely related to many other fields. In fact, many see MDWE as a specific branch of the Model-driven engineering (MDE) concept which focuses on creating applications by exploiting and reusing domain models. In this light many more related approaches exist. Two similar approaches for example include WebComposition and WSDM. WebComposition [Gellersen et al., 1997] is an object-oriented approach which allows to compose Web applications from smaller components. Even though the method is inherently object-oriented, there still are many similarities with MDWE fields in terms of reuse of information, maintainability concerns and the use of Semantic Web data. A recent evolution of WebComposition for example includes an subscribe-notification mechanism based on a Web-services implementation [Meinecke et al., 2007]. Similar approaches include portal-based approaches like [Trujillo et al., 2007] and [Bellas, 2004]. The Web Semantics Design Method (WSDM) [De Troyer et al., 2008] is yet another approach which shares many features with most MDWE approaches but differs in that it is based on an audience driven design philosophy. Instead of creating Web applications for a given dataset it centeres on modeling the specific target audience (and the resulting requirements) first.

Many other further related systems exist which share some origins have many related concepts, for example in the Hypermedia field. AHA! [De Bra et al., 2006] is an Hypermedia approach that allows adding adaptation and additional navigation primitives to existing Web documents. Its approach in terms of design certainly has similarities with some MDWE methods. AHA! will be discussed in more depth in Section 7.3. Related systems include for example Brusilovsky's Interbook [Brusilovsky, 2007], a goal driven approach like KBS Hyperbook [Henze et al., 1999], narrative approaches based on pagelets like APeLS [Hockemeyer et al., 2003].

Even though every mentioned approach has interesting relations with the MDWE field we will not discuss them in more depth here.

2.4 Summary

In this chapter we gave an overview of the evolution of the Web. From a large linked set of documents the Web grew into a large infrastructure for full fledged interactive Web applications. The next step in the evolution of the Web may be into a Web of reusable and open *knowledge* by using semantics to describe the meaning of data. For this the Semantic Web was developed. In this chapter we shortly summarized its foundational languages (XML, RDF and RDFS), extensional languages (OWL, rules and query language SPARQL) and the evolution of linked data. We also described tagging as a simple user friendly form of adding metadata to objects on the Web. Next to semantics we also described a parallel evolution of MDWE as a structured approach for developing WIS. These methods allow non programmers a method to simply design applications using models which is potentially less error-prone and better reusable and maintainable. We also discussed a number of existing MDWE methods that we can find in literature.

Chapter 3

Hera-S

This chapter presents the revision of an approach for designing adaptive Web information systems, called Hera. In Hera we distinguish models for specifying the domain, the navigation over the content from the domain, and the context for navigation. Hera is characterized by the use of the Semantic Web language RDF to express the models for the domain and context data and for the adaptive navigation. We present a short history of Hera including its fundamental ideas. Then, based on the experience with earlier Hera implementations we present a revised and more focussed conceptual revision named Hera-S, including an implementation called Hydragen and some additional tools for constructing and executing its models. The running example is expressed in terms of Hera models that use RDF query expressions to retrieve the data from RDF repositories for content and context data.

3.1 Introduction

This chapter presents a complete reworked version of Hera, called Hera-S. Hera is a MDWE method for Web information system design that found its origins in an approach for hypermedia presentation generation. It was also this focus on hypermedia presentation generation that gave the first engine complying with this method its name HPG [Frasincar, 2005]. The method distinguishes three main models that specify the generation of hypermedia presentations over available content data. With a model that specifies the content, a model that specifies grouping of content in pages and links between pages, and a model that specifies presentation details, the method enables the creation of a hypermedia-based view over the content. Originally, in the first generation of the method and its toolset, the models specified a transformation from the content to the presentation. The engine that was compliant with this definition was based on the EXtensible Stylesheet Language (XSLT) [Clark, 1999] and is therefore known as HPG-XSLT.

One of the characteristic aspects that HPG-XSLT supported was adaptation. Adaptation is as hypermedia-based systems exploit user information to adapt visible aspects of the system for the user [Brusilovsky, 1996].

As an illustrative example, we show in Figure 3.1 how different presentations could be produced by the engine out of a single design in which the "translation" to formats such as HTML, SMIL, and WML was dealt with generically. Note that the models which are mentioned in this figure will be explained in detail in the following sections.

Characteristic for the Hera models compared to other methods was not only their

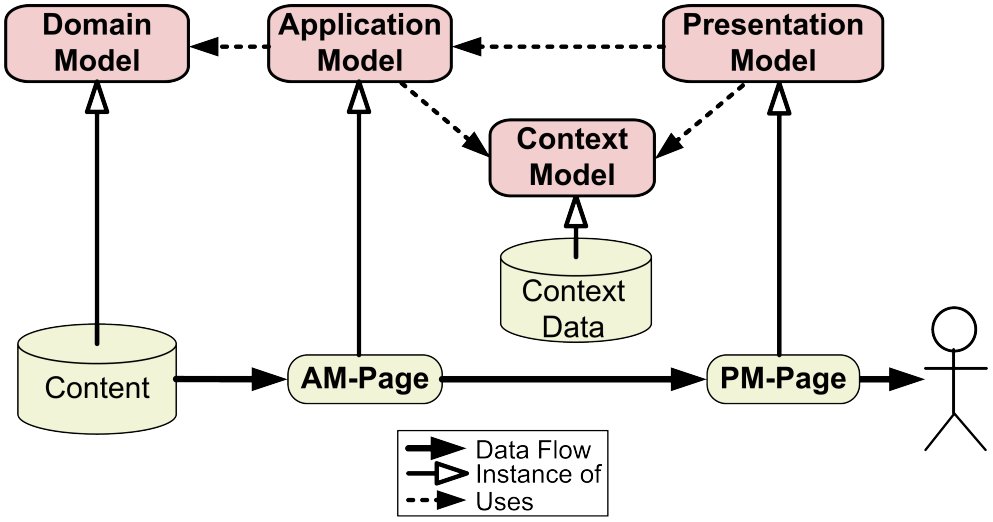

Figure 3.1: Hera Models

focus on user- and context-adaptation support, but also the choice to base the models on RDF and RDFS. For an overview of other MDWE methods see Section 2.2.3 and for a detailed comparison between these methods please refer to [Schwinger et al., 2008], especially table 48 in that paper. The use of Web standards such as RDF and RDFS as a modeling paradigm facilitates easy deployment on very heterogeneous data sources: the only assumption made is that a semi-structured description (in RDF) of the domain is available for processing. Not only is such a representation less costly to develop than any alternative, it also enables reuse of existing knowledge and flexible integration of several separate data sources in a single hypermedia presentation.

During the further research into the development of the method, the support was extended for more advanced dynamics. Where the first XSLT-based approach primarily transformed the original content data into a hypermedia document, with which the user could interact through following links with a Web browser, the subsequent engine version allowed the inclusion of form processing, which led to the support of other kinds of user-interaction, while retaining the hypermedia-based nature. Out of this effort, also a Java-based version of the engine became available which used RDF-queries to specify the data involved in the forms.

The experience out of these HPG-based versions and the aim for further exploitation of the RDF-based nature of the models have led to a further refinement of the approach in what is now termed Hera-S. The Hera-S-compliant models do combine the original hypermedia-based spirit of the Hera models with more extensive use of RDF-querying and storage. Realizing this RDF data processing using the Sesame framework [Broekstra et al., 2002] and the SPARQL query language (Section 2.2.5) caters for extra flexibility and interoperability.

We aim to exemplify also the characteristic elements included in Hera-S. As we mentioned before there is the RDF-based nature of the models. There is certainly also the focus on the support for adaptation in the different model elements. Adapting the data processing to the individual user and the context that the user is in (in terms of application, device etc.) is a fundamental element in WIS design and one that deserves the right attention: managing the different design aspects and thus controlling the complexity of the application design is crucial for an effective design and implementation.

In this chapter we first address the main characteristics of the method, before we

explain the models, i.e. the main design artifacts. We present the implementation of the hypermedia presentation generation process induced by the models. We also consider some extensions to the basic approach that can help the design process in certain scenarios.

3.2 Method

The purpose of Hera is to support the design of applications that provide navigation-based Web structures (hypermedia presentations) over semantically structured data in a personalized and adapted way. The design approach centers on *models* that represent the core aspects of the application design. Figure 3.2 gives an overview of these models. With the aid of an engine for running those models (e.g. Hydragen, refer to 3.6) we can serve the Web application, as depicted in this figure. Where Hera was originally a more monolithic approach where all models and data were specifically designed and served within its framework, Hera-S is more targeted on existing data and its models. Instead Hera-S only needs to define the application model, which most importantly defines the application logic. The actual data and presentation lie outside its core and are assumed as is. This allows a good separation of concerns and the reuse of external data sources.

Thus the appropriate pipeline of models captures the entire application design, leaving room for the designer to change or extend the implementation where desired. In this section, we only give a short overview over the different models and associated modeling steps, before each of them is presented in more detail in the subsequent sections.

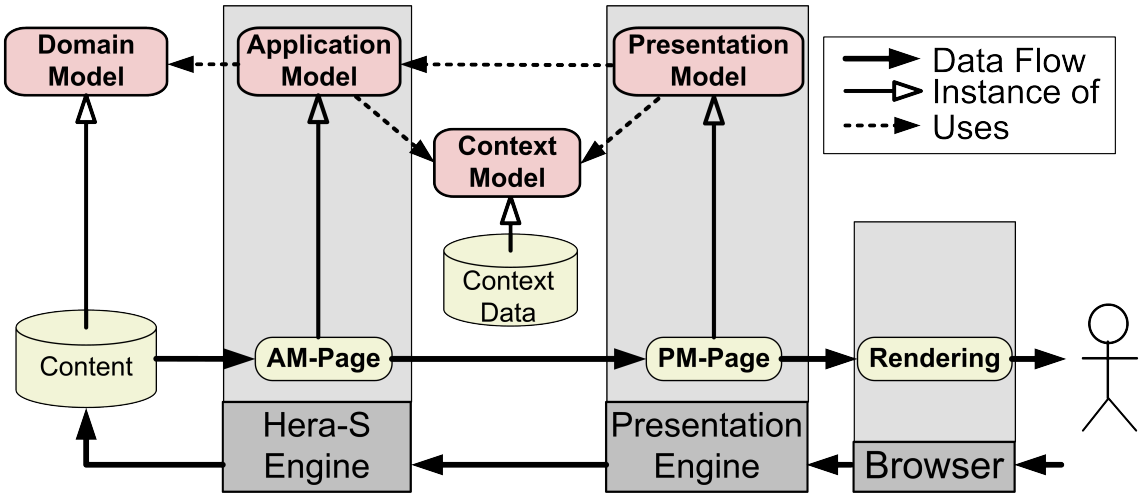

Figure 3.2: Hera-S Framework

Before we can create a model to specify the core design of the application, we need in Hera-S as a starting point a *Domain Model (DM)* that describes the structure of the content data. The sole purpose of the DM is to define how the designer perceives the semantical structure of the content data: it tells us what we need to know about the content over which we want the application to work. Based on this DM, the designer creates an *Application Model (AM)* that describes a hypermedia-based navigation structure over the content. This navigation structure is devised for the sake of delivering and presenting the content to the user in a way that allows for a (semantically) effective access to the content.

In turn, this effective access can imply the personalization or adaptation that is deemed relevant. Hera-S allows dynamic personalization and adaptation of the content, which means that user information will be gathered and stored during usage of the system and

the correct adaptation will be computed when a page is requested by the user based on the knowledge about the user currently available in the system. For this purpose, context data is maintained (under control of the application) in a so-called *Context Model (CM)*. This context data is typically updated based on the (inter)actions of the user as well as possibly on external information, e.g. by importing user data from other applications.

So, on the basis of DM and CM the AM serves as a recipe that prescribes how the content is transformed into a navigational structure. To be more precise, instantiating the AM with concrete content results in an *AM (instance) page (AMP)*. These AMP's can be thought of as pages that contain content to be displayed and navigation primitives (based on underlying semantic relations from the DM) that can be used by the user to navigate to other AMP's and thus to semantically "move" to a different part of the content. An AMP itself is not yet directly suitable for a browser, but can be transformed into a suitable presentation by a presentation generator, i.e. an engine that executes a specification, for example a Presentation Model (PM) of the concrete presentation design in terms of layout and other (browser-specific) presentation details. In Section 4.5 we demonstrate that both proprietary and external engines can be used for this task. For the Hera-S method this presentation generation phase itself is not specific and it may be done in whatever way is preferred. So, the AM specifies the (more conceptual or semantical) construction of the navigational structure over the content, while the subsequent presentation phase, possibly specified by a PM, is responsible for the transformation of this structure into elements that fit the concrete browsing situation.

AMP creation is conceptually *pull-based*, meaning that a new AMP is only constructed in the Hera pipeline at request (in contrast to constructing the whole instantiation of the AM at once, which was done in for example the implementation by the HPG-XSLT engine). Through navigation (link-following) and forms submission the user triggers the Hera-S engine, which results in internally adapting (updating) the website navigation or context data and the creation of a new AMP.

As indicated in the introduction, Hera models use RDFS to represent the relevant data structures. In the next sections we will see this for the specification of the data in DM, CM and AM. In the engines these RDFS descriptions are used to retrieve the appropriate content and generate the appropriate navigation structures over that content. In HPG-XSLT the actual retrieval was directly done by the engine itself, whereas in HPG-Java this was done with the aid of expressions that are based on SeRQL queries [Broekstra, 2005]. In Hera-S the actual implementation exploits the fact that we have chosen to use RDFS to represent the model data and allows to use native *RDF querying* to access data, for example the content and context data. For this Hera-S allows the application to connect to the content and context data through the Sesame RDF framework. This solution, combining Hera's navigation design and Sesame's data processing by associating SPARQL queries to all navigation elements, allows to effectively applying existing Semantic Web technology and a range of its solutions that is becoming available. We can thus include background knowledge (e.g. ontologies, external data sources), we can connect to third-party software (e.g. for business logic), and connect to services through the RDF-based specifications. We can also use the facilities in Sesame for specific adaptation to the data processing and to provide more extensive interaction processing (e.g. client-side, scripting). The dynamics and flexibility required in modern Web information systems can thus be met by accommodating the requirements that evolve from an increasing demand for personalization, feedback, and interaction mechanisms. Note that Hera-S models in principle are query and repository independent. We only require a certain type of functionality, and if a repository fulfils these requirements it can be used for

implementation of Hera-S models.

3.3 Data Modeling

Before the application design can consider the personalized navigation over the domain content, the relevant data needs to be specified. This can be an existing RDFS model of some external data source. But of course we also might want to start from scratch and design a completely new application. In this section we will look at this last scenario, so that we can see what is needed to create an application using the Hera-S method. For that we will look at creating an example application about movies using data from the International Movie Database (IMDb)[1].

As a necessary first step in the approach the *data modeling* step leads to the construction of the data models for the domain content and the context of the user. The modeling of the domain content uses RDFS and is primarily targeted towards capturing the semantical structure of the domain content. With the Hera-S engine we also allow the model to be an OWL ontology (without restrictions) as this is supported by the underlying Sesame database.

Looking at our example, we first need an RDFS or OWL definition of the IMDb dataset. In fact, several of these definitions can already be found in either RDF (e.g. [Taylor, 2008]) or in some other modeling paradigms like UML (e.g. [Koch et al., 2008]). For example, consider the UML model of the IMDb domain in Figure 3.3 as it gives a good visual overview of the domain. We can either reuse an existing RDF version, but can also choose for many other paradigms like UML if these model would be better suited for some reason as translations exist to convert those models to RDFS or OWL, e.g. for UML refer to [Falkovych et al., 2003] or [Hillairet, 2007]. However, for sake of the example, instead of reusing an existing source we can also create the model ourselves, e.g. by using the Protégé ontology editor [Protégé, 2011].

Figure 3.4 contains a screenshot of an OWL model of the IMDb database in Protégé. We divided the OWL model in four parts. One part, with the prefix 'imdb', contains the "core" of the movie domain, describing the movies and the persons involved in those movies. In another part, with the prefix 'cin', we extend the IMDb models with cinema information that show the movies that are modeled in the 'imdb' part.

In this figure we also see prefixes starting with 'cm'. They relate to the *context modeling*. The CM is modeled and implemented in a similar way as the DM. The main difference between the two is that the content data is meant to be presented to the user, while the context data is meant to support the context-dependent adaptation of the application. So, the content typically contains the information that in the end is to be shown to the user, while the context data typically contains information used (internally) for personalization and adaptation of content delivery. This distinction might not always be strict, but as it is only a conceptual distinction in Hera-S, the designer may separate content and context in whatever way he desires. As a consequence, we assume that context data is under direct control of the engine, while the content often is not. In the IMDb example the context model is modeled in the same way as the domain. We first maintain a model, with the prefix 'cm1', that contains users and their comments on movies in the 'imdb' part. The second part, with the prefix 'cm2', contains a description of tickets bought by the user for a particular movie showing in a particular cinema.

[1] `http://www.imdb.com/`

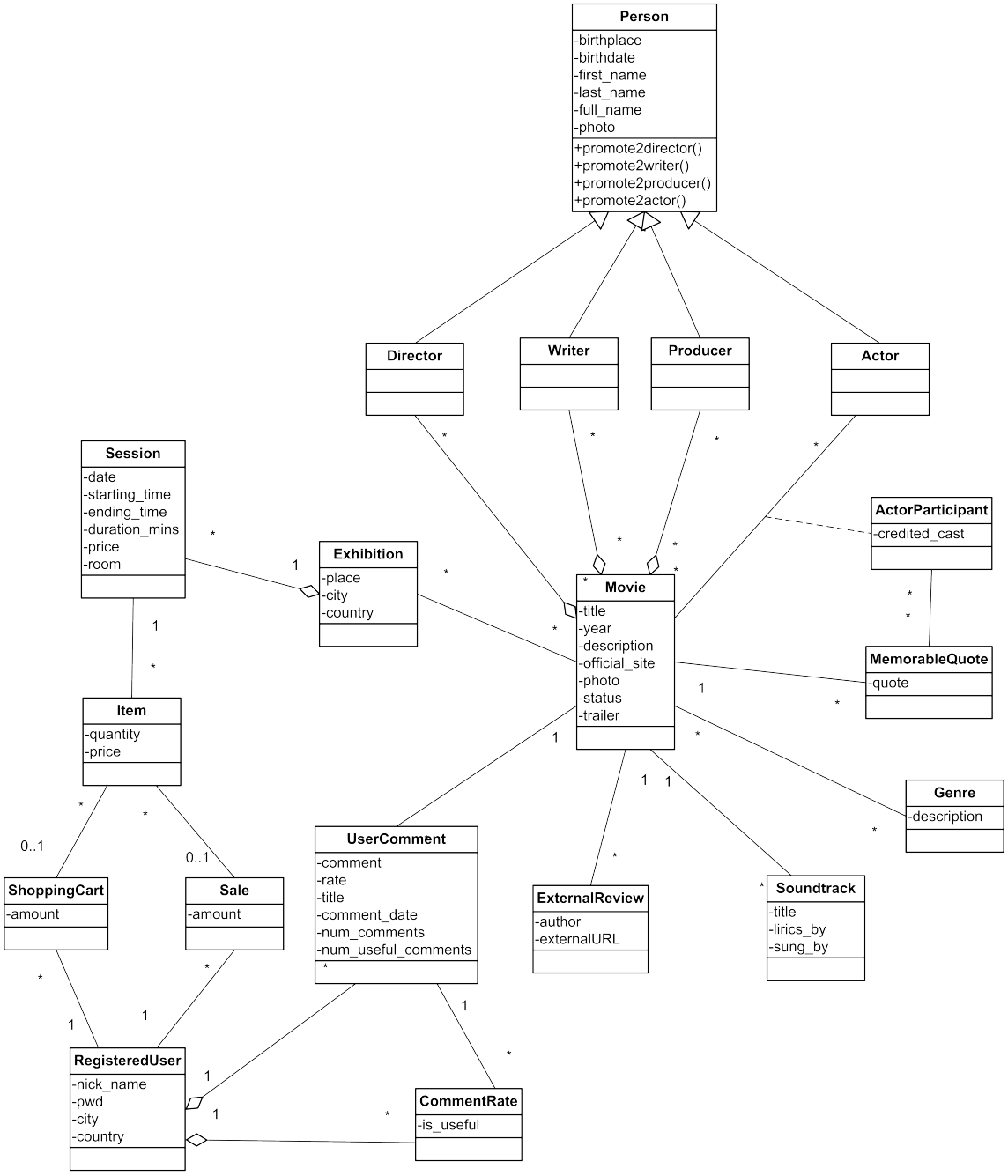

Figure 3.3: UML model of the IMDb domain

Considering the role and function of the context data, we can identify different aspects of context. We will come back to context data later when we discuss adaptation in the AM, but we now address the context data modeling. Even though the designer is free to choose any kind of context data, we in general discern three types: session data, user data and global data.

- Session data: Session data is data relevant to a certain session of a certain user. An example of such data is the current browsing context, such as the device that is used to access the Web application or the units browsed in the current session.

- User data: Data relevant to a certain user over multiple sessions (from initial user data to data collected over more than one session). User (profile) data can be used for

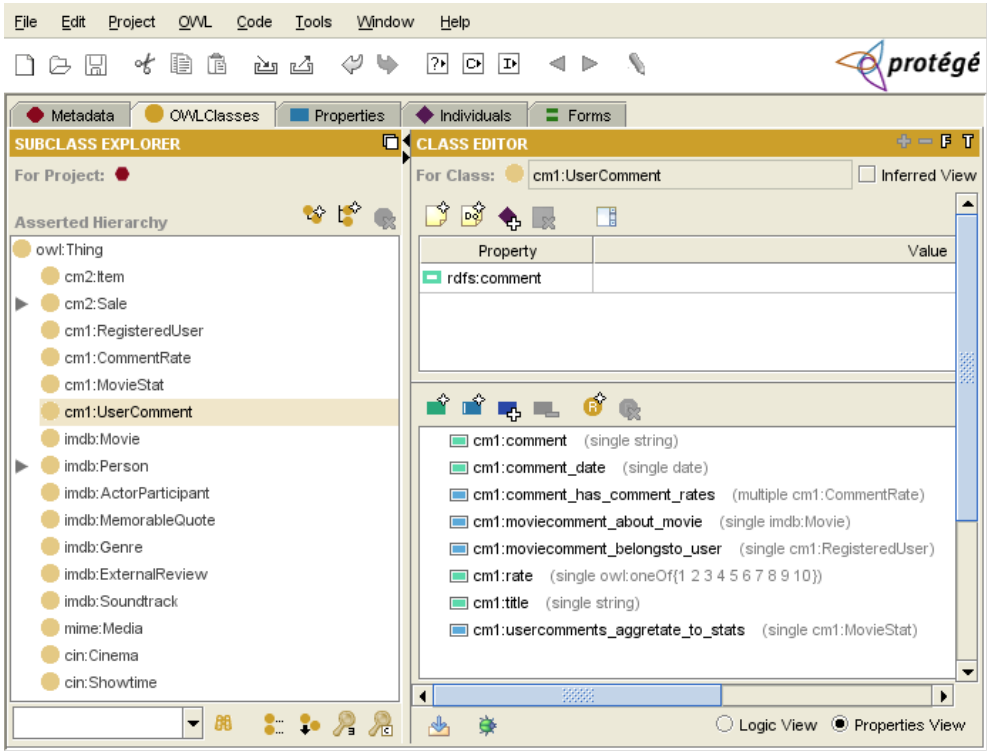

Figure 3.4: Protégé screenshot for the IMDb data modeling

personalization (even at the beginning of a new session). Note that for maintaining this user data over time the application needs some authentication mechanism.

- Global data: Usage data relevant to all users over all sessions. Global data typically consists of aggregated information that gives information about groups of people. Examples include "most visited unit" or "people that liked item x, also browsed item y".

In Figure 3.5 we show part of an RDF graph representation of the domain data model that we will use as a basis for the examples of this chapter. It shows the main classes and a selection of relationships between those classes, while omitting their datatype properties.

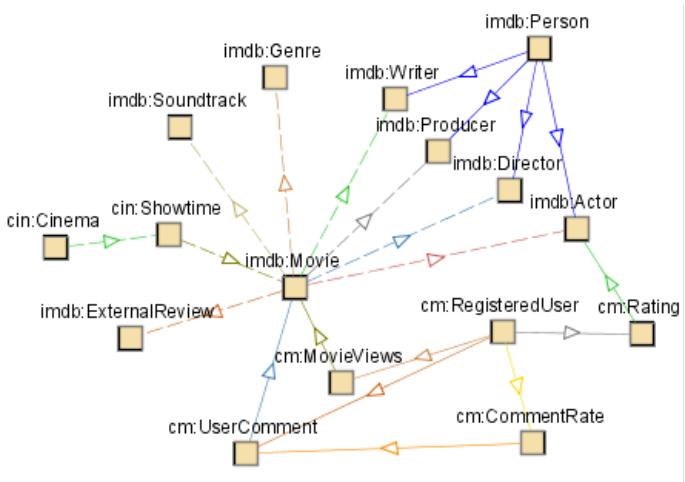

Figure 3.5: RDF-graph representation of IMDb domain and context data

Both the DM and CM data are in Hera-S implemented using a Sesame repository. For the CM which is under direct control of the application, this allows the application to manage and update the context as it perceives this context. Next to this, it also provides the means for other processes to use and update (parts of) this information. The context data could for instance be manipulated by business logic software for the sake of adaptation, or by external user profiling software.

By supporting unrestricted RDF, RDFS and OWL DM's, Hera-S is particularly suited to (re-)use existing domain ontologies. Moreover, many existing data sources that are not yet available in RDFS or OWL format can be used via Semantic Web wrapping techniques [Simile, 2008; Thiran et al., 2005]. In the latter case Sesame can be used as a mediator between such a data source and Hera-S. As such, Sesame can be used as part of the data integration layer which is further discussed in Section 4.2.

As we will see in detail in the next section, the access to the data from the DM or CM is part of the application definition. It means that the access to the RDF data is part of the model. In principle, we assume that the concepts from the DM and CM are associated with concrete data elements in the data storage structure. As we use the Sesame RDF Framework as our back-end repository, this data can be exploited and reasoned upon. Accessing the content data in Hera-S will be done via explicit SeRQL queries and by making them explicit in the models we support customizable access via customizable SeRQL queries. Thus, the full potential of the SeRQL query language can later be used in the AMP creation. For the purpose of defining the content and context, we can abstract from the SeRQL queries, but for the support of different types of adaptation, we benefit from making this SeRQL access explicit.

3.4 Application Model

Based on the domain definition, application modeling results in the AM that specifies the navigational behavior of the Web application. The AM enables designers to specify how the (navigational) access to the data (dynamically retrieved from the domain) is structured by describing which data is shown to the user and what Web pages the user can navigate to. At the same time, the AM allows this specification to be dynamic, such that the navigational access to the data can be personalized to a user and adapted for a specified context.

Please note that in the AM we use the Turtle (refer to 2.2.3) and SPARQL (refer to 2.2.5) syntaxes. The several AM constructs will be introduced using examples in our movie domain. We will start with the basic core constructs in Section 3.4.1, adaptation in Section 3.4.2 and more advanced modeling primitives in Section 3.4.3

3.4.1 Units, Attributes and Relationships

Now we will discuss the constructs that we provide in our AM. We will start in this section with the basic constructs that are sufficient to build basic Web applications and move on afterwards to more complex constructs for realizing richer behavior.

The AM is specified by means of navigational units (shorthand: units) and relationships between those units. The instantiation of units and relationships is defined by (query) expressions that refer to the (content and context) data as explained in Section 3.3.

The unit can be used to represent a "page". It is a primitive that (hierarchically) groups elements that will together be shown to the user. Those elements shown to the user are called attributes and so units build hierarchical structures of attributes.

An attribute is a single piece of information that is shown to the user. This information may be constant (i.e. predefined and not changing), but usually it is based on information inside the domain data. If we have a unit for a concept `C`, then typically an attribute contained in this unit is based on a literal value that is directly associated with `C` (for example as a datatype property). Note that literals may not only denote a string type, but also other media by referring to a URL. Furthermore, we offer a built-in media class (denoted by 'hera:Mime') that can be used to specify an URL and the MIME-type of the object that can be found at the URL. This can be used if the media type is important during later processing.

Below we give an example of the definition of a simple unit, called 'MovieUnit', to display information about a movie. We mention two elements in this definition:

- From the second until the seventh line, we define which data instantiates this unit. This data is available as input ('am:hasInput') from the environment of this unit, e.g. passed on as link parameter or available as global value. In this case we have one variable '`M`': the fourth line specifies the (literal) name of the variable, while the fifth line indicates the type of the variable. In this case, a value from the 'imdb:Movie' class concept from the domain will instantiate this unit.

- In the eighth until the fourteenth line, starting with 'am:hasAttribute', we decide to display an attribute of this movie, namely a title. We label this attribute with 'Title' (such that later we can refer to it), and we indicate (with 'am:hasQuery') how to get its value from the data model. This query uses the 'imdb:movieTitle' (datatype) property applied to the value of '`M`'. Note that in the query '`$M`' indicates that '`M`' is a Hera-S variable, i.e. outside the scope of the SPARQL query itself. The output of the SPARQL query result is bound to the Hera-S variable '`T`' (implicitly derived from the SELECT list).

In our RDF/Turtle syntax the definition looks as follows:

```
:MovieUnit a am:NavigationUnit ;
   am:hasInput [
      am:variable [
         am:varName "M" ;
         am:varType imdb:Movie
      ]
   ] ;
   am:hasAttribute [
      rdfs:label "Title" ;
      am:hasQuery
         "SELECT ?T
         WHERE {$M a imdb:Movie;
                   imdb:movieTitle ?T .}"
   ] .
```

For the attribute value instead of the simple expression (for which we can even introduce a shorthand abbreviation) we can use a more complicated query expression, as long as the query provided in 'am:hasQuery' returns a datatype property value.

Relationships can be used to link units to each other. We can use relationships to contain units within a unit, thus hierarchically building up the "page" (we call these

aggregation relationships), but we can also exploit these relationships for navigation to other units (we call these navigation relationships).

As a basic example, we include in the unit for the movie not only its title but also a (sub)unit with the name and photo of the lead-actor and a navigational relationship that allows to navigate from the lead-actor information to the full bio-page (unit) for that actor. Note that from now on we omit in our text namespaces when they appear to be obvious.

- We have separated here the definitions of 'MovieUnit' and 'ActorUnit' (which allows later reuse of the 'ActorUnit'), but we can also define subunits inside the unit that contains them.

- In the definition of the 'MovieUnit' one can notice compared to the previous example, that we have an additional subunit with its label 'LeadActor', with its type 'ActorUnit', and with the query that gives the value with which we can instantiate the subunit.

- In the definition of the 'ActorUnit' one can notice its input variable, two attributes, and a navigation relationship. This navigation relationship has a label 'Actor-Bio', targets a 'BioUnit', and with the query based on the 'imdb:actorBio' property it determines to which concrete 'BioUnit' this 'ActorUnit' offers a navigation relationship. Note that in this case the variable '$B' is passed on with the navigational relationship (it is also possible to specify additional output variables that are passed on with the relationship).

```
:MovieUnit a am:NavigationUnit ;
   am:hasInput [ am:variable [ am:varName "M";
                               am:varType imdb:Movie]] ;
   am:hasAttribute [ rdfs:label "Title" ; ... ] ;
   am:hasUnit [
      rdfs:label "LeadActor" ;
      am:refersTo :ActorUnit ;
      am:hasQuery
         "SELECT ?L
         WHERE {$M a imdb:Movie;
                   imdb:movieLeadActor ?L .
                ?L a imdb:Actor .}"
] .

:ActorUnit a am:NavigationUnit ;
   am:hasInput [ am:variable [ am:varName "A" ;
                               am:varType imdb:Actor]] ;
   am:hasAttribute [
      rdfs:label "Name" ;
      am:hasQuery
         "SELECT ?N
         WHERE {$A a imdb:Actor;
                   imdb:actor_name ?N .}" ] ;
   am:hasAttribute [
      rdfs:label "Photo" ;
      am:hasQuery
```

```
        "SELECT ?P
        WHERE {$A a imdb:Actor;
                  imdb:actorPhoto ?P .}" ] ;
   am:hasNavigationRelationship [
      rdfs:label "Actor-Bio" ;
      am:refersTo :BioUnit ;
      am:hasQuery
        "SELECT ?B
        WHERE {$A a imdb:Actor;
                  imdb:actorBio ?B .}"
   ] .
```

So, in these examples we see that each element contained in a unit, whether it is an attribute, a subunit, or a navigational relationship, has a query expression ('hasQuery') that determines the value used for retrieving (instantiating) the element.

Sometimes we know that in a unit we want to contain subunits for each of the elements of a set. For example, in the 'MovieUnit' we might want to provide information for all actors from the movie (and not just the lead actor). Below we show a different definition for the 'MovieUnit' that includes a set-valued subunit element ('am:hasSetUnit'). In its definition, one can notice

- the label "Cast" for the set unit,
- the indication that the elements of the set unit are each an 'ActorUnit', and
- the query that determines the set of concrete actors for this movie to instantiate this set unit, using the 'imdb:movie_actor' object property.

```
:MovieUnit a am:NavigationUnit ;
   am:hasInput [ am:variable [ am:varName "M" ;
                               am:varType imdb:Movie]] ;
   am:hasAttribute [ rdfs:label "Title" ; ... ] ;
   am:hasSetUnit [
      rdfs:label "Cast";
      am:refersTo ActorUnit ;
      am:hasQuery
        "SELECT ?A
        WHERE {$M a imdb:Movie;
                  imdb:movieActor ?A .}"
   ] .
```

So, we see that a set unit is just like a regular unit, except that its query expression will produce a set of results, and this will cause the application to arrange for displaying a set of (in this example) 'ActorUnits'.

Likewise, we can have set-valued query expressions in navigational relationships, and with 'am:tour' and 'am:index' we can construct guided tours and indexes respectively. With these the order of the query determines the order in the set and with the index an additional query is used to obtain anchors for the index list.

3.4.2 Adaptation Examples

Adaptation and personalization are important aspects within the Hera methodology. For this purpose the query expressions can be used to include conditions that provide control over the instantiation of the unit. Typically, these conditions use data from the CM and thus depend on the current user situation. For example, we can use 'U' as a (global) variable that denotes the current (active) user for this browsing session (typically this gets instantiated at the start of the session). Let us assume that in the CM for each 'Actor' there is a 'cm:actorRating' property that denotes 'U's rating of the actor (from 1 to 5 stars) and that the user has indicated to be only interested in actors with more than 3 stars. We could then use this rating in adapting the cast definition in the last example:

```
am:hasSetUnit [
   rdfs:label "Cast";
   am:refersTo ActorUnit ;
   am:hasQuery
      "SELECT ?A
      WHERE {$U cm:actorRating _:rating cm:stars ?V;
                               cm:ratingOnActor ?A .
             ?A imdb:playsIn $M .
             FILTER (?V > 3)}"
] .
```

Here we see how we can influence (personalize) the input to an element (in this case a set) by considering the user context in the query that determines with which values the element gets constructed. To be precise, in this example we state inside the movie unit what actors of the cast this user will be provided with, i.e. which values we "pass on".

Another user adaptation example would be that the user has indicated not to be interested in photos from actors. We could then change the query for the photo attribute accordingly:

```
:ActorUnit a am:NavigationUnit ;
   am:hasInput [ am:variable [ am:varName "A" ;
                               am:varType imdb:Actor]] ;
   ...
   am:hasAttribute [
      rdfs:label "Photo" ;
      am:hasConditionalQuery [
         am:if "SELECT *
                WHERE {$U cm:showElement _:el .
                       _:el cm:showAbout imdb:actorPhoto .}"

         am:then "SELECT ?P
                  WHERE {$A imdb:actorPhoto ?P .}"
      ]
   ] ;
   ... .
```

Here we see that with 'am:hasConditionalQuery' the attribute becomes "conditional", i.e. the photo attribute is only shown when the condition ('am:if') query produces a

non-empty result. We can also add an 'am:else' part here and for example display the string "no photo displayed".

Finally, we present a more complex example of adaptation. Consider again the 'ActorUnit' from the previous section, which showed an actor's name, his picture, and a link to his bio. Now imagine we would like to add the list of movies in which the actor played. However, because some movies are age-restricted, we would like to restrict this list so that adult-rated movies are only shown to registered users that are 18 or older. As in the previous adaptation examples, this adaptation can be achieved by tweaking the SPARQL query that computes the list of movies:

```
:ActorUnit a am:NavigationUnit ;
   am:hasInput [ am:variable [ am:varName "A" ;
                               am:varType imdb:Actor]] ;
   ...
      am:hasSetUnit [
      rdfs:label "Movies Played In";
      am:refersTo MovieUnit ;
      am:hasConditionalQuery [
      am:if "SELECT *
             WHERE {$U cm:age ?G .
                    FILTER (?G > 17)}"
      am:then "SELECT ?M
               WHERE {$A imdb:actorMovie ?M .
                      ?M a imdb:Movie .}"
      am:else "SELECT ?M
               WHERE {$A imdb:actorMovie ?M .
                     ?M a imdb:Movie;
                        imdb:mpaaRating ?R .
                     FILTER (?R != 'NC-17')}"
      ]] .
```

First of all, notice in the code-excerpt the 'am:hasSetUnit', which represents the list of movies for the active actor ('A'). This list is defined by a conditional query, in which it is verified whether the active user ('U') is registered and his age is over 17 ('am-if'). If this condition holds ('am:then'), all movies of the particular actor are computed. If the condition does not hold ('am:else'), the computed movie list is restricted to movies which are not MPAA (Motion Picture Association of America)-rated as "NC-17" (No Children Under 17 Admitted).

3.4.3 Other constructs

Before, the basic constructs were explained. In this section we will look at some additional features of Hera-S that also allow designers to use some more advanced primitives in order to construct richer applications.

Update Queries

For the sake of adaptation we need to maintain an up-to-date context model. In order to do so, we need to perform updates to this data. For this, we have the functionality to specify an 'am:onLoad' event update query and an 'am:onExit' update query within

every unit, that are executed on loading (navigating to) and exiting (navigating from) the unit. Furthermore, we allow attaching an update query to a navigation relationship so that the update query is executed when a link is followed. In all cases, the designer may also specify more than one update query. In our example we could for instance maintain the number of page views (visits) of a certain 'movieUnit', and update this information if the 'movieUnit' is loaded using the 'onLoad' query:

```
:MovieUnit a am:NavigationUnit ;
   ...
   am:onLoad [
      am:updateQuery
         "UPDATE {?V cm:amount (?views+1) .}
          WHERE {$U a cm:RegisteredUser;
                    cm:userMovieViews ?V .
                 ?V cm:amount ?views;
                    cm:viewsOfMovie $M .}"
   ];
   ... .
```

Frame-based Navigation

We explained earlier that units can contain other units. The root of such an aggregation hierarchy is called a top-level unit. The default semantics of a navigational relationship (that is defined somewhere inside the hierarchy of a top-level unit) is that the user navigates from the top-level unit to the top-level unit that is the target of the relationship. In practice this often means that in the browser the top-level unit is replaced by the target unit. However, we also allow specifying that the navigation should only consider the (lower-level) unit in which the relationship is explicitly specified, so that only that unit is replaced while the rest of the top-level unit remains unchanged.

This behavior is similar to the frame construct from HTML. We specify this behavior by explicitly indicating the source-unit for the relationship. Inspired by the HTML frame construct we allow the special source-indications "**_self**" (the unit that contains the relation), "**_parent**" (the unit that contains the unit with the relation) and "**_top**" (the top-level unit - the default behavior). Alternatively, relations may also indicate another containing unit by referring to the label of the contained unit. An example of a navigational relationship with source indication looks like:

```
am:hasNavigationRelationship [
   ...
   am:source am:_self ;
   ... ]
```

Forms

Besides using relationships (links) for navigation, we also support applications that let the user provide more specific feedback and interact. For this we provide the form unit. A form unit extends a normal unit with a collection of input elements (that allow the user to input data into the form) and an action that is executed when the form is submitted. In a form a navigational relationship typically has a button that activates the submission.

Below we give an example of a form that displays the text “Search Movie:” (line three) with one text input-field (line five to eleven) to let the user enter the movie he wants to browse to. If the user enters a value in this field, it is bound to the variable ‘movieName’ (line nine). After submitting the form via a button with the text “Go” (line fourteen and fifteen), the user navigates to the ‘MovieUnit’ (line fourteen) that will display the movie for which the name was entered in the input-field, which is specified in the query (starting in line twenty) using the variable ‘movieName’.

```
:MovieSearchForm a am:FormUnit ;
   am:hasAttribute [
      am:hasValue "Search Movie: "
   ];
   am:formElement [
      rdfs:label "Search Input";
      am:formType am:textInput;
      am:binding[
                am:variable [am:varName "movieName" ;
                             am:varType xsd:String ]]
   ];
   am:formElement [
      rdfs:label "Submit Button";
      am:formType am:button;
      am:buttonText "Go";
      am:hasNavigationRelationship [
         rdfs:label "Search Form-Movie" ;
         am:refersTo :MovieUnit ;
         am:hasQuery
            "SELECT ?M
             WHERE {?M a imdb:Movie;
                       imdb:movieTitle ?X .
                    FILTER (?X = $movieName)}"
      ]
   ].
```

Scripting objects

Current Web applications offer users a wider range of client-side functionality by different kinds of scripting objects, like Javascript and VBscript, stylesheets, HTML+TIME timing objects etc. Even though WIS methods like Hera concentrate more on the creation of a platform-independent hypermedia presentation over a data domain, and these scripts are often (but not always) browser/platform specific, we still provide the designer a hook to insert these kind of scripting objects.

The designer can specify within a scripting object whatever code he wants, as this will be left untouched in generating the AMP’s out of the AM. Furthermore, the designer can add an ‘am:hasTargetFormat’ property to specify one or more target-formats for format-specific code, e.g. HTML or SMIL. This allows later in the process to filter out certain format-specific elements if these are not wanted for the current presentation. The scripting objects can use the variables that are defined within the scope of the units. Scripting objects can be defined as an element within any other element (i.e. units and attributes). Furthermore, it can be specified if the script should be an attribute of its

super-element or not (e.g. similar to elements in HTML that have attributes and a body). The need to place some specific script on some specific place is of course decided by the designer.

Service Objects

An application designer might want to use additional functionality that cannot be realized by a client-side object, but which involves the invocation of external server-side functionality. Therefore, we provide so-called service objects ('am:serviceObject') to support Web services in the AM. The use of a service object and the reason to provide support for it is similar to that of scripting objects. The designer is responsible for correctness and usefulness of the service object.

Think of utilizing a Web service from a Web store selling DVDs in order to be able to show on a movie page an advertisement for buying the movie's DVD. A service object needs three pieces of information:

- a URL of the Web service one wants to use,
- a SOAP message that contains the request to the Web service, and
- a definition of the result elements.

A service object declaration can be embedded as a part of every other element. If a unit is navigated to ("created"), first the service objects will be executed. The results of the service object will either be directly integrated into the AM and treated as such, or the result can be bound to variables. Service objects can use unit variables in their calls.

3.5 Presentation Model

The PM is defined by means of so-called regions and relationships between regions. Regions are abstractions for rectangular parts of the user display and thus they satisfy browsing platform constraints. They group navigational units from the AM, and like navigation units, regions can be defined recursively. They are further specified by a layout manager, a style, and references to the navigational units that they aggregate. We note that the usage of layout managers was inspired by the AMACONT project's component-based document format[Fiala et al., 2003], adopting its abstract layout manager concept in the Hera-s PM. As will be explained later (Section 4.5), this enables to use AMACONT's flexible presentation capabilities for the generation of a Web presentation.

Figure 3.6 shows a visual excerpt of a PM for the running example, i.e. the regions associated to the 'MovieUnit' navigational unit and its subregions.

The 'MovieUnit' navigational unit is associated with the region called 'MovieRegionFull'. It uses the layout manager 'BoxLayout1' for the arrangements of its subregions ('MovieRegionLeft' and 'MovieRegionRight'), and the style given by 'DefaultStyle'. 'BoxLayout1' is an instance of the layout manager class 'BoxLayout' that allows to lay out the subregions of a region either vertically or (as in this case) horizontally. The style describes the font characteristics (size, color), background (color), hyperlink colors etc. to be used in a region. The definition of styles was inspired by Cascading Style Sheets (CSS) [Bos et al., 2011]. We chose to abstract the CSS formatting attributes because (1) not every browser supports CSS at the current moment and (2) we would like to have a representation of the style that can be customized based on user preferences.

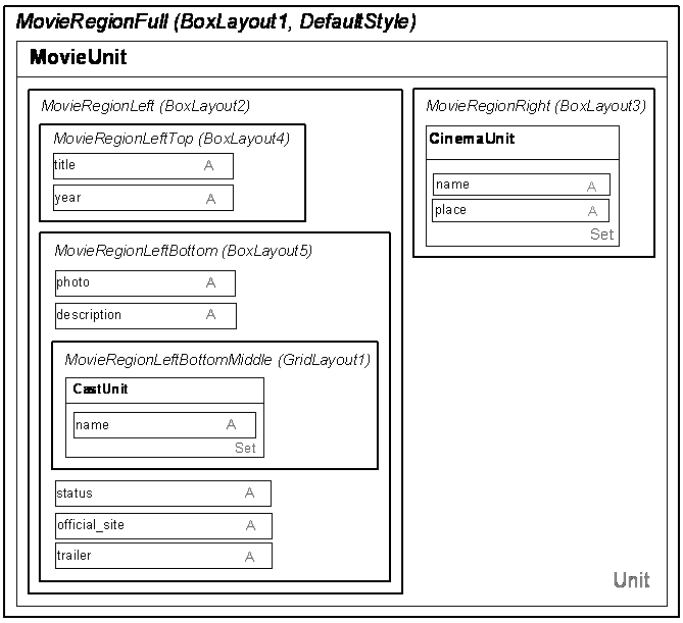

Figure 3.6: Presentation Model for the 'MovieUnit' Navigational Unit

Both 'MovieRegionLeft' and 'MovieRegionRight' use 'BoxLayout's with a vertical organization of their inner regions. For the 'title' and 'year' attributes 'BoxLayout4' is used, which specifies a horizontal arrangement. For 'photo', 'description', the region containing the names in the cast, 'status', 'official_site' and 'trailer' it is used 'BoxLayout5' which states a vertical arrangement. The names in the cast are organized using 'GridLayout1' (an instance of the layout manager class 'GridLayout'), a grid with 4 columns and an unspecified number of rows. The number of rows was left on purpose unspecified as one does not know a priori (i.e. before the presentation is instantiated and generated) how many names the cast of a movie will have. The regions that do not have a particular style associated with them inherit the style of their container region. Note that in Figure 3.6 we have omitted constant units (e.g., '(', ')', 'Cast', etc.) in order to simplify the explanation.

Besides the layout manager classes exemplified in Figure 3.6, the definition of PM supports additional ones. 'BorderLayout' arranges subregions to fit in five directions: north, south, east, west, and center. 'OverlayLayout' allows to present regions on top of each other. 'FlowLayout' places the inner regions in the same way as words are placed on a page: the first line is filled from left to right and the same is done for the second line etc. 'TimeLayout' presents the contained regions as a slide show and can be used only on browsers that support time sequences of items, e.g., HTML+TIME[Schmitz et al., 1998] or SMIL[Bulterman et al., 2008]. Due to the flexibility of the approach, this list can be extended with other layout managers that future applications might need.

The specification of regions allows defining the application's presentation in an implementation-independent way. However, to cope with users' different layout preferences and client devices, Hera-S also supports different kinds of adaptation in presentation design. As an example, based on the capabilities of the user's client device (screen size, supported document formats etc.), the spatial arrangement of regions can be adapted. Another adaptation target is the corporate design (the "look-and-feel") of the resulting Web pages. According to the preferences and/or visual impairments of users, style elements like background colors, fonts (size, color, type), or buttons can be varied. For a thorough elaboration of presentation layer adaptation the reader is referred to [Fiala et al., 2004].

3.6 Implementation

Different from previous versions, the Hera-S implementation called *Hydragen* concentrates on the application model level. For data management Hydragen mainly relies on Sesame. On the front end, Hydragen provides a simple presentation generation module, but this can be easily replaced by a specialized front end as will be shown in Chapter 4.

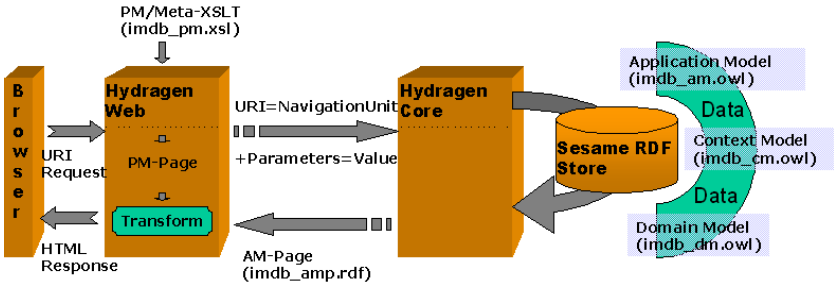

Figure 3.7: Hydragen Information Flow

Hydragen is composed of two main components: the *HydragenCore* and *HydragenWeb*. HydragenCore encapsulates the core engine functionality. For each request of a *NavigationalUnit* it is able to generate a new AMP. *HydragenWeb* on the other hand is implemented via a Java Servlet, which forwards requests to *HydragenCore* and transforms the generated AMP by applying XSLT transformations into an HTML response, i.e. HydragenWeb acts as the presentation generator of the Hydragen framework. In principle, the format of response is decided by XSLT specification used for presentation generation. The information flow between these components is depicted in Figure 3.7.

Figure 3.8 depicts the HydragenCore class diagram (abstracting away some of the lesser important classes and methods). In the execution phase, the HydragenWeb servlet receives an 'HttpServletRequest' everytime the user executes a navigation action (e.g. click a link, submit a form etc). The request is in the form of a URI along with a set of parameters. The parameters passed with the request are retrieved, and stored in an internal data-structure (a TreeMap) along with their corresponding values. The URI of the requested NavigationalUnit comes as value of a predetermined HttpServletRequest parameter - URI. Since during generation of HTML response, some encoding is done for URI's that are generated as part of response, the parameters need to be decoded before being stored and passed on to the CoreEngine for request handling. A typical example of a URL request for a NavigationalUnit would appear as:

```
http://localhost:8080/herasWeb/HydragenWeb?
    URI=http://wwwis.win.tue.nl/~hera/Hera-S/imdb_am.owl^LoginUnit
```

The 'generateAMP()' method from the CoreEngine class is responsible for processing the request for a specific NavigationalUnit and generating the corresponding AMP in a session specific file. The CoreEngine forwards the 'generateAMP' request to the 'AMParser' class which initializes query transactions with the AM, DM and Content repositories before processing the request further. The AMParser then initializes the AMP output file as an RDF file. Although a request is always for a specific NavigationalUnit, the application meta-model allows a NavigationalUnit to contain more NavigationalUnits and in order to distinguish between handling of the root NavigationalUnit and the internal NavigationalUnits, the 'handleRootUnit' method is called. To give an idea of the unit handling process consider the activity diagram in Figure 3.9.

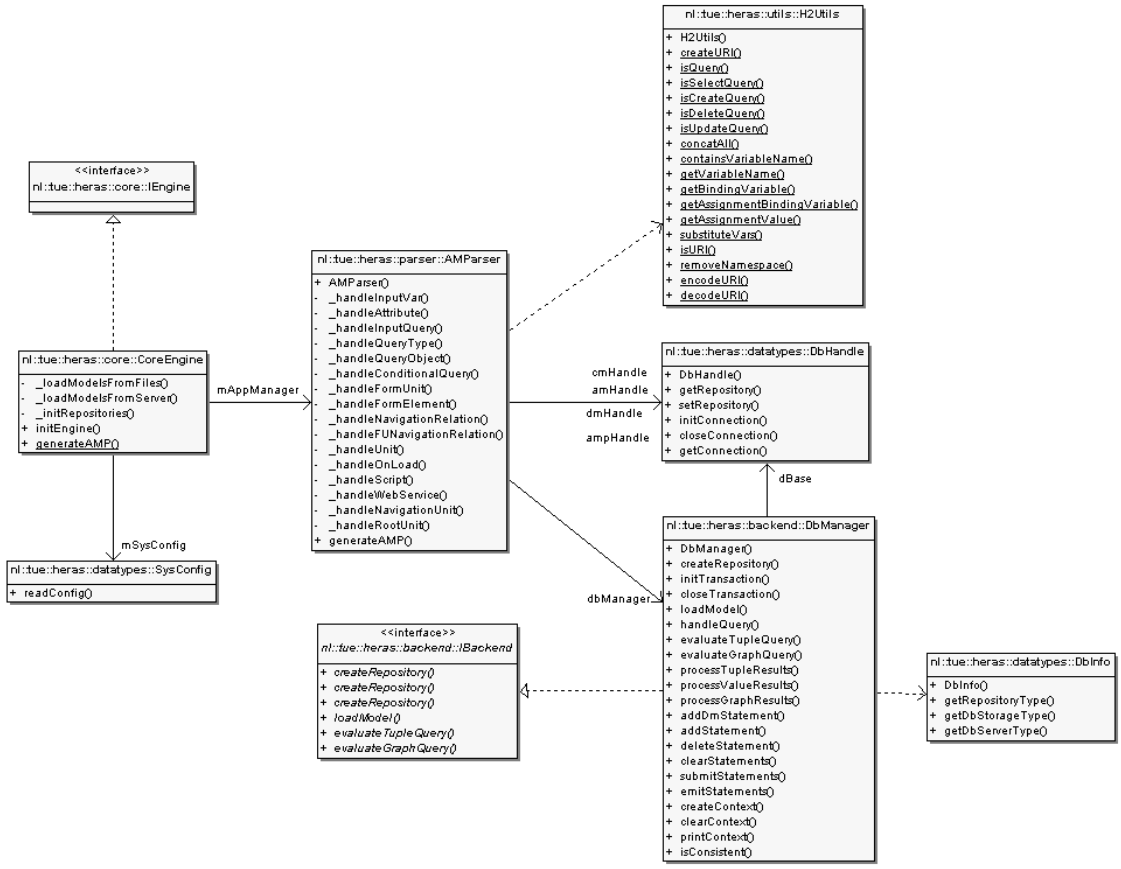

Figure 3.8: HydragenCore Class Diagram

Since Hydragen supports creating new nodes, deletion and update of statements, the statements that are generated/modified during processing of a *NavigationalUnit* are captured in a set of internal lists: an 'add_list', 'delete_list' and 'amp_list'. As a last step before dumping the AMP to a file, these modifications are submitted. The addition and deletions (from the add_list and delete_list) are submitted to Sesame, while the AMP statements (from the amp_list) are first stored locally in the session specific context of the AMP repository for performance reasons. Afterwards the contents of that session context of *AMP* repository are also sent to the Sesame server.

Hydragen does include some model checking facilities. Model checking can be used to check things like if the content adhers to the its DM definition. But it can also be used to check if the AM satisfies certain conditions, like that is has a starting unit, it lacks a cyclic sub-unit structure, etc, to ensure that the AM produces a meaningful Web application. Model checking is implemented by using Pellet[Sirin and Parsia, 2004]. Pellet is an open source OWL DL reasoner in Java. It is flexible and has a reasonable performance and can be easily configured and extended to do various static or run-time model checking activities.

HydragenWeb is deployed as a Web application on a servlet container such as Apache Tomcat. In addition, the Sesame server and *Sesame* Web-client also need to be installed (though, possibly on a different machine). Sesame must then be configured to contain the data definitions and (possibly remote) data. All paths to application, data and context servers and initialization options can be configured by using the Hydragen Configuration File. Sesame allows setting up of several properties of the repository such as use of inference layer, type of storage, persistence, etc.

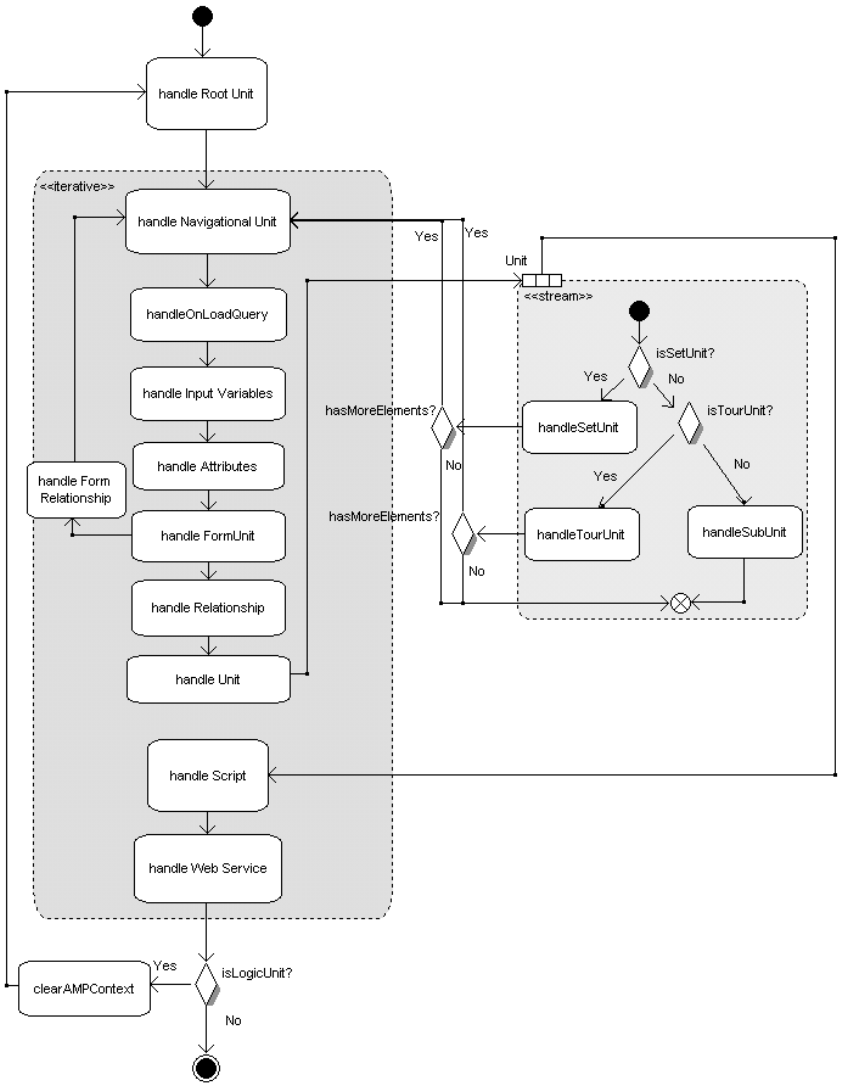

Figure 3.9: Activity Diagram for handling Units

3.7 Summary

In this chapter we discussed the revised version of the Hera methodology, called Hera-S. Progress was made by making Hera-S more flexible than previous incarnations, by relying on runtime querying of both models and data by using Sesame. Sesame can act as a data mediator and query external data outside of Hera-S' control and mix this with locally updated data by Hera-S. The Hera-S engine, called Hydragen, has focussed its attention to application modeling. Both back end (e.g. data integration) and front end (e.g. presentation generation) processes can be plugged in by specialized engines, which allows us to utilize their specific strengths.

Hera-S came with a new AM structure, which we described in detail. We described core functionality like units, attributes and relations, we also described conditional adaptation and additional constructs like forms, scripts, etcetera. We also described the Hera-S presentation model, which we will discuss in more detail in the next chapter. Finally we described some of the implementation details of Hydragen engine.

Chapter 4

Extending Hera-S

This chapter presents a number of extension points of Hera-S. We will discuss the steps needed to allow for data integration in Hera-S, more of which we shall see in the following chapters. Next are the Hera-S graphical builders which support designers to (graphically) design domain, context and application models. Similarly, we will look at maintainability and separation of concerns by using aspect-orientation in Hera-S. We offer mechanisms that allow to insert functionality that changes the entire application independently of the application model, for example for adding application-wide adaptation to mobile devices. After presenting a simple approach for Hera-S presentation model before, in this chapter we present an implementation that uses the external and specialised AMACONT engine to do the presentation generation phase.

4.1 Introduction

Chapter 3 described the core Hera-S framework. In this chapter we look at several natural extension points, with which we also demonstrate the flexibility and extendibility of Hera-S. First we discuss how data integration fits together with Hera-S in Section 4.2. Next, we describe a graphical toolset for building DM and AM models in Hera-S in Section 4.3. In Section 4.4, we discuss maintainability and how we implemented an aspect-oriented approach for separately implementing pervasive features that influence the entire Web application. Another important aspect is the actual presentation generation. In Section 3.5 we described the PM, while in Section 4.5 we discuss an implementation of the presentation generation component that uses the specialised AMACONT framework, with an advanced PM approach. This presentation generation component can also use aspect-orientation and we describe the use of external semantic information for improving personalisation.

4.2 Data Integration in Hera-S

An interesting feature of Semantic Web languages is that it allows to interlink datasets by using URIs as global identifiers. This allows people to point to elements in datasets that they actually do not control by linking to these URIs. Because links are the only way to relate concepts there is afterwards no real difference between links within a dataset or links between datasets. These links thus allow applications over these domains to navigate the combined dataset as one.

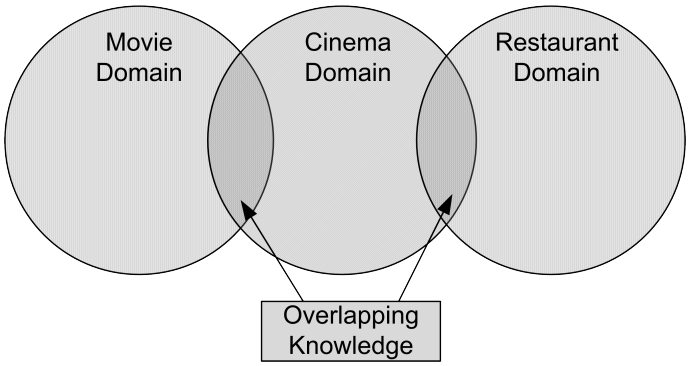

Figure 4.1: Intersecting domains

As an example consider Figure 4.1. It is clear that different descriptions, e.g. one that describes movies and one that describes cinemas, describe a lot of different things but can have also some things in common. In this case the movie domain descriptions describes a lot of information about movies, while the one for cinemas shows some of the cinemas that show these movies. This is a clear example of overlapping domains. Consider also the example in the same figure of overlap between cinemas and restaurants. They mostly describe very different things, but can also share things such as information about how they might be situated in the same geographical location. By combining these three domains we can effectively start answering questions like 'Which cinemas show 'the Matrix' this evening and are within 500 meters from a Greek restaurant?'.

Now, suppose we have a two or more related domain datasets, an important question to answer is how do we integrate them using Semantic Web techniques. Generally, the following steps have to be taken:

1. For the datasources that are not yet expressed in RDF, either convert them into RDF, e.g. using one of many RDFizers[1], some other converter, or use a wrapper that allows SPARQL querying over, for instance if it is in a relational database [Thiran et al., 2005].

2. Link overlapping knowledge concepts between sources and indicate which elements in the different datasources are the same.

3. Provide a 'seamless' platform that allows to query the combined datasources in a single query.

Many solutions already exist for the first of these steps. We will look at steps 2 and 3.

Let us first consider step 3. Suppose that we have a number of interlinked datasources. How can we make them available to the user in one integrated view? The simplest solution would be to retrieve all of these datasources as RDF files and put them in one RDF-database like Sesame. Actually, any RDF-database is capable of storing all of these RDF files. This solution is rather inflexible, however, especially if you consider that some sources might not be controlled by the designer and change rapidly over time. Or consider that you might actually want to store different sources separately, specially for performance reasons.

Sesame[2] does offer an answer for this issue. Figure 4.2 contains the layered architecture of Sesame. The Sail API defines a set of interfaces for RDF stores and inferencers. It

[1] http://simile.mit.edu/wiki/RDFizers

[2] `http://www.openrdf.org`

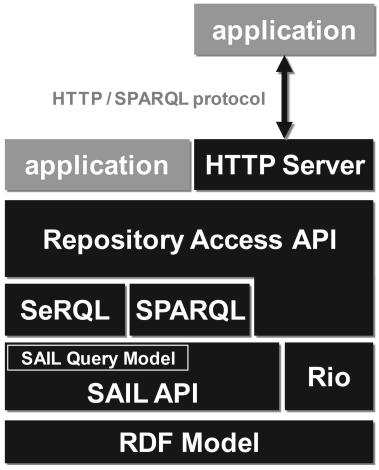

Figure 4.2: Sesame Architecture[a]

[a] http://www.openrdf.org/doc/sesame2/system/ch02.html

allows designers to write custom storage mechanisms by implementing an interface that allows Sesame to send queries to it. This does not only allow for specialized local storage mechanisms, but for example also for the possibility to define external datasources. Sail also provides a custom inference mechanism via a specific Sesame rule language which allows designers to express a set of axiomatic triples and inference rules.

The Sail layer expects querying and (optionally) storing and inferencing directives of how to connect with a new datasource. It is left over to storage mechanism to take care of creating search indexes for the data. This has a drawback in terms of speed, e.g. when combining several datasources there is no overal index that knows which data is where. This means that generally all queries will have to be sent to all underlying sources and the results have to be retrieved and combined locally. This can introduce serious performance issues when combining several large sources that are not solvable by Sesame. Therefore we made a start at looking at strategies for using several datasources with the explicit purpose of efficiency and improving performance [Bellekens et al., 2008; Verheijen, 2008]. In the first of these papers we presented a specific solution for the SenSee TV-recommender (refer to Section 6.2) where we also developed a number of general guidelines for speeding up large intergrated RDF datasets. Verheijen's Master thesis was an introductory study into high performance for quering distributed RDF-databases, which would be an interesting starting point for future work. To keep this PhD-thesis focused it was chosen to not go into detail of these studies here.

Instead, this thesis focuses more on the second step, namely linking overlapping knowledge concepts. We quickly found that this is not a Hera-S specific issue, but a general issue that is relevant for a large domain of Semantic Web applications. Therefore, in Chapter 5 we will specifically look at this generic issue and in the following chapters we will show how our solution to this issue can be used to improve applications in various domains.

4.3 Model Builders

In most RDF serializations it can become difficult to see which structures belong together and what the general structure of the document is; especially if the documents get larger. This also applies to the Hera-S models and has the consequence that manually creating

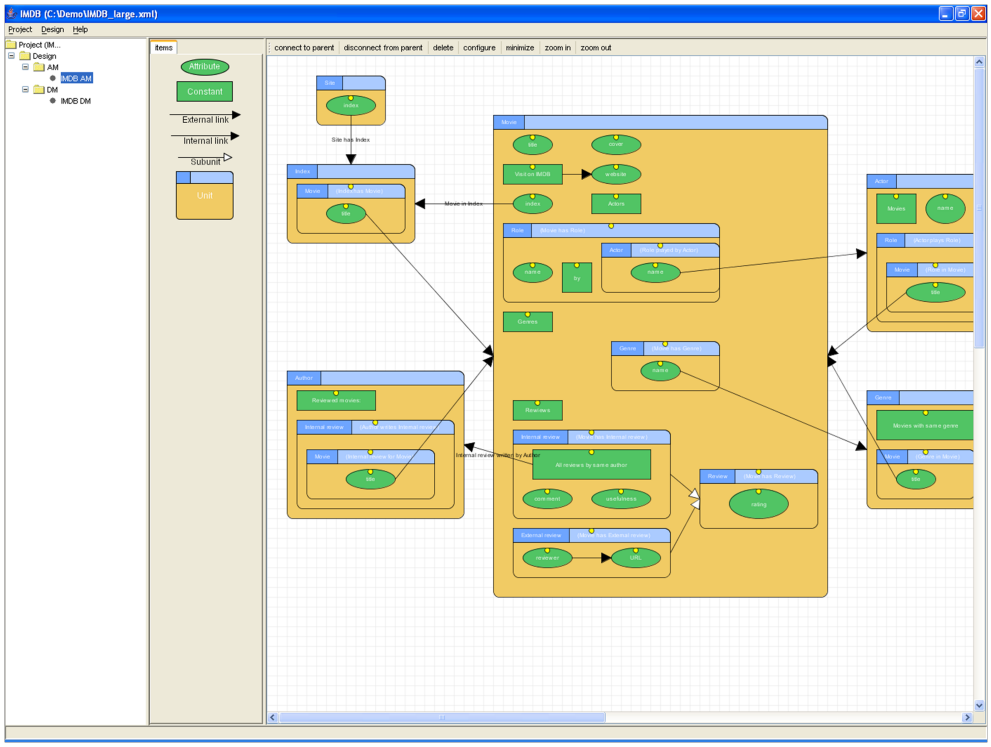

Figure 4.3: Hera Studio

them can become error-prone. It is therefore beneficial to offer the designer tool-support for creating those models graphically. Figure 4.3 is a screenshot of Hera-S Studio. Hera-S Studio contains a domain, context and application model editor in which the designer can specify the models in a graphical way.

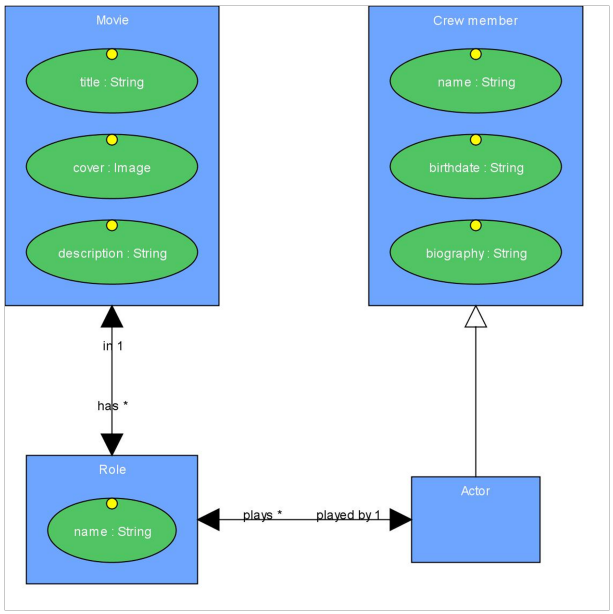

Figure 4.4: DM example

Note that Hera-S Studio is not a general purpose OWL or RDFS-editor like for instance Protégé, rather a custom-made version specialized for Web applications designed through Hera-S models. In the DM-editor (figure 4.4), designers can define classes and object properties between those classes. For every class a number of datatype properties can be

given that have a specified media type (e.g. String, Image etc). Furthermore, inheritance relations for classes and properties can be denoted. In addition, instances of the classes and properties can be specified. Note that if more complex constructs are needed, the designer could also use a general purpose OWL or RDFS editor like Protégé. The DM-editor can also be used to create CM-models as these are identical from a representation point of view.

The AM-editor provides a graphical way for specifying an AM (figure 4.5 gives an example). It specifically allows organizing the units and the relationships between them. Per unit, elements can be defined and displayed. Units and elements can be dragged and dropped and connected with other units. Selected elements on the canvas can also be copied and pasted. Detailed information in the models like queries is hidden in the graphical view, and can be configured by double-clicking the elements. For the simpler constructs, the editor provides direct help: for example when defining a datatype property, the editor gives the designer a straightforward set if choices out of the (inherited) datatype properties of the underlying context and domain models. However, for the more complex constructs, the designer has the freedom to express his own queries and element properties. In addition, the designer can control the level of detail of the model to get a better overview of the complete model.

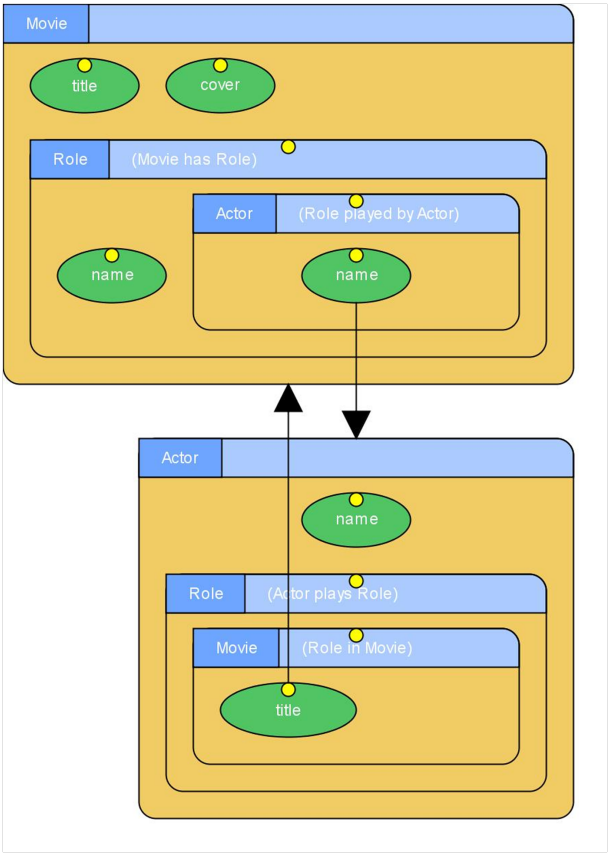

Figure 4.5: AM example

Hera-S Studio is a Java program. Its data is stored in an XML format that not only stores the used elements, its properties, associated queries and conditions and the connections between the elements, but also the size and position of the different elements. Next to that, it maintains the connection between elements in AM and DM via the simpler query constructs (i.e. constructs refering to single datatype properties). This means that if a designer chooses to change the DM and deletes a property that is used in

the AM he will be notified by the program. Hera-S Studio also has some modest model checking capabilities using the Pellet [Sirin and Parsia, 2004] library that we also use inside Hydragen. It can do some basic consistency checks and find logical errors in the DM model. Next to that it checks for dead-ends and consistency in the AM. Resulting DM, CM and AM models can be exported to their RDF(S) and OWL equivalents that can be used in the Hydragen-engine. Next to that, it can be configured to directly contact a Sesame server and remotely upload or update the models that are used by the Hydragen engine.

4.4 Modeling Aspects in Hera-S

4.4.1 Introduction

Adaptation support in Hera-S, as detailed in Section 3.4.2, is based on conditional element inclusion. The conditions can be either integrated using an 'if-then-else' construct or by directly integrating the conditions in the SPARQL expressions underlying every element of the Application Model, making the instantiation of the particular AM element dynamic. If the condition references the CM (denoted by the "cm:" prefix), it thus expresses user- or context-dependency, and so the resulting Web application becomes adaptive to the current user or context. As an example, reconsider the ActorUnit from Section 3.4.2, and imagine we don't want to display the actor's picture when the user is using a small-screen mobile device that has a horizontal resolution of 800 pixels or less, and furthermore imagine the desire that actor movies are only shown when the actor is in the user's favorite actor list. The ActorUnit would then look as follows (for clarity, the adaptation conditions that were added are shown in bold; non-altered parts were omitted):

```
:ActorUnit a am:NavigationUnit ;
   ...
   am:hasAttribute [
      rdfs:label "Photo" ;
      am:hasQuery "SELECT ?P
                   WHERE {$A imdb:actorPhoto ?P .}
                         {$U cm:userDevice ?device .
                          ?device cm:hres ?horresolution .
                          FILTER (?horresolution > 800)}"
    ] ;
   am:hasSetUnit [
      rdfs:label "Movies Played In";
      am:refersTo MovieUnit ;
      am:hasQuery
         "SELECT ?M
          WHERE {?M a imdb:Movie;
                    imdb:movieActor $A .}
                {$U cm:favActor $A .}"
      ] .
   ...
```

It shows that using SPARQL, arbitrary user or context conditions can thus be written into any regular selection query present in the AM. While SPARQL is both versatile and

expressive and therefore allows arbitrary and powerful personalization, inherently the way in which adaptation is specified in the global Web application design suffers from some drawbacks:

- *Adaptation specification is localized*: In the above example, we restricted the display of pictures from actors. However, adaptation is typically not localized: for example, for small-screen users it makes more sense to eliminate all pictures from their application instead of only actor pictures. While certainly possible with the above approach, it would require restricting all pictures, wherever their appearance is specified in the AM. This forces the designer to (manually) identify all attributes of units in the AM where pictures are shown, and subsequently alter the corresponding queries in all these units.

- *Adaptation specification is hard-coded*: as mentioned in the example of the previous bullet, the designer needs to identify (over the entire AM) which queries will return pictures, and subsequently alter these queries. The adaptation specification is thus hard-coded and not re-usable. Furthermore, when extending the AM later on (for example, by adding songs or games to the Web site) and possibly introducing new pictures, these pictures would still be shown to the user, unless adaptation conditions are also (manually) added for these new parts of the AM.

- *Semantic information is ignored*: valuable semantic information, present in the Domain Model, is ignored. For instance, in the previous example of omitting pictures for small screen users, the semantic (type) information for media elements (present in the DM) could be exploited to automatically select all pictures. Instead, it is left to the designer to interpret which queries involve pictures.

- *Global (structural) properties are not usable*: because adaptation conditions are limited to one particular element (i.e. integrated in a particular query), they cannot express restrictions based on several elements or, more general, on global properties of the AM. For example, expressing that pictures can be shown to small screen users, but only on pages where no other pictures appear, is currently impossible.

The above problems leave room for human error, with possible inconsistencies in the adaptation behavior as a result. Additionally, adaptation engineering is also totally intertwined with the (classical) Web engineering (concerns), complicating the overall design.

A more systematic and high-level approach, treating adaptation as a design concern in its own right, is thus called for. We address this issue by presenting an aspect-oriented and semantic-based approach for adaptation specification. Our approach effectively separates adaptation engineering from (regular) Web engineering, and tackles the fact that adaptation concerns are typically spread over the entire Web application. In order to add strength, flexibility and robustness to the adaptation specification, we exploit the valuable semantic information typically present in the underlying domain, in addition to (global) structural properties of the Web application.

4.4.2 Related work

As mentioned, adaptation design is typically spread all over the Web application design. A similar observation was made in the programming community. When studying certain design concerns, it gradually became clear that some concerns cannot be localized to one

particular function or module. A clear example is logging, the code of which is typically spread over the entire application code. Such a concern is called cross-cutting. The answer of the programming community to address such a concern while maintaining good abstraction and modularization is aspect-orientation [Kiczales et al., 1997]: instead of duplicating the required logging code in different places in the application code, it is specified as an aspect. An aspect consists of a pointcut and an advice. The pointcut performs a query over the programming code, selecting exactly those locations where the cross-cutting concern comes into play. The advice consequently specifies which action should be taken whenever this cross-cutting concern is detected (i.e. which piece of code should be 'injected'). Exactly the same paradigm can be applied for adaptation specification: some (adaptation) action(s), i.e. the advice, is typically required to be in multiple places in the Web application, i.e. the pointcut. The pointcut will thus consist of a query over the AM, selecting exactly those elements which require an action to be taken for adaptation. The action itself will typically consist of injecting adaptation-conditions to particular places selected by the pointcut. However, we will not limit ourselves to only adding adaptation-conditions. In fact, as we will see, an adaptive action will consist of an arbitrary transformation applied to the AM elements selected in the poincut, and will thus also allow adding, deleting or replacing elements, based on a certain (context- or user-dependent) condition.

Some related work has also been done on aspect-orientation in other MDWE design methods. In the context of UWE (refer to Section 2.3.3), aspect-orientation has been used to specify some specific types of adaptive navigation [Baumeister et al., 2005], namely adaptive link hiding, adaptive link annotation, and adaptive link generation. However, our approach as will be described below, is on a different level of granularity. Instead of very specific types of adaptation like link hiding, we will focus on more flexible arbitrary transformations both influenced by design time and runtime information. Also, the approach in [Baumeister et al., 2005] uses manual pointcut specifications, where we will focus on an approach that is completely abstracted from the application model. Another approach of aspect-orientation in MDWE was found in WebML. [Schauerhuber et al., 2007] describes the extension of WebML with aspect-oriented techniques. That research was initiated and performed independently and in parallel to our work as detailed below. In order to extend WebML with aspect-orientation the orginal WebML language was re-modeled (with some changes) in UML, so classical aspect-oriented tools and techniques could be used. Much as in our approach adaptation specification was separated from the regular design using aspect-orientation. However, this approach does not combine semantic information with aspect-orientation, which, as we will show, provides additional strength for our approach. Also, it is not able to taken into account global application properties when selecting elements, which, as we will also show, can help to provide for more flexible, robust and reusable adaptation aspects.

4.4.3 Aspect-Oriented Adaptation Engineering

As part of a collaborating effort [Casteleyn et al., 2009] we devised own our aspect language named Semantics-based Aspect-Oriented Adaptation Language (SeAL), which is custom-made to provide adaptation support in the context of Hera-S. Such languages, specifically aimed at supporting certain tasks in a specific domain, are called domain-specific languages [van Deursen et al., 2000]. Their advantages are well-documented, and clearly motivate our choice for a newly designed language here, as opposed to using a general-purpose language. Most important benefits are reduced complexity (constructs take into account

the peculiarities of Hera-S models, and thus allow easy and quick access and handling of these models) and ease of use (domain experts may readily understand and use our domain-specific language, and do not need to learn a more complex general-purpose language).

We will now describe the most important constructs in the SeAL language.

Pointcuts

Pointcut expressions select exactly those elements from the Application Model where adaptation needs to be inserted. The basic pointcut statement selects all AM elements. All subsequent language constructs restrict (condition) this selection in some way.

Using the *type* construct, selection of elements is restricted to those of a certain type. The type can refer to every element of the AM metamodel, e.g. "NavigationUnit", "subUnit", "Attribute", "NavigationRelationship", "Query", etc. Elements of different types may be selected by specifying all desired types, separated by a ",". Typical examples include:

- *;* (selects all AM elements)
- *type NavigationUnit, subUnit;* (selects all units and sub-units)
- *type Attribute;* (selects all attributes)

Next to restricting the element selection according to type, it may also be restricted according to name, value (a property of an AM-element should have a certain value), aggregation (an element contains another element, or is contained in another element), or aggregate function (using a numerical condition on the amount of contained elements, specified using count). The navigational structure of the AM can also be exploited to restrict element selection: for relationships, restrictions on the target may be specified (using refersTo), while units can also be selected based on the navigational relations between them (besides refersTo and subUnit constructions also the inverse of these relationships). Where appropriate, string pattern-matching is allowed when conditioning values. Logical expressions may be negated or combined using logical conjunction or disjunction; parentheses are used to alter default logical operator precedence. Typical examples include:

- *type NavigationUnit or type subUnit;* (select all units and sub-units; this in fact is equivalent to a previous example, namely "type unit, subUnit")
- *type NavigationUnit and hasAttribute "name" "movie*";* (selects all units that have a name that starts with "movie")
- *type subUnit and contains (2 < count(type Attribute) < 5);* (selects all sub-units which have between 2 and 5 attributes specified)
- *type Attribute and containedIn (type NavigationUnit and contains (type Attribute and hasLabel "title"));* (selects all attributes contained in a unit, which has an attribute labeled "title"; note that 'contains' and 'containedIn' are special aggregation keywords)
- *type NavigationRelationship and from (type NavigationUnit and contains (count(type NavigationRelationship) > 3);* (select all relations which originate from a unit containing more than 3 relationships; note that 'from' is again a special keyword indicating reverse links)

- *type NavigationUnit and refersTo (type NavigationUnit and contains (type tour));* (selects all units containing a relationschip that navigates to a unit containing a guided tour)

- *type NavigationUnit and from (type NavigationUnit and hasName "*movie*");* (selects all units to which a(ny) relation points to, navigating from a unit with a name containing "movie")

SeAL also supports calls to the native underlying query language (SPARQL) to retrieve AM elements. This support has been provided so that the expert user can still specify intricate queries that exploit the full expressiveness of SPARQL (and of which the semantics are not expressible in SeAL). Such queries can be specified by using the '< `SPARQL` >' keyword at the beginning of a pointcut specification. A trivial example includes:

- *< `SPARQL` > "SELECT ?A WHERE {?U am:hasAttribute ?A . ?A rdfs:label ?L . FILTER (?L = 'title')};* (selects all attributes with a label "title")

The < `SPARQL` > construct can also be used to query the DM. In this way semantic information from the DM can be exploited to retrieve relevant elements from the AM, using SPARQL as the native query language to access the DM. For example, to select all subunits whose corresponding domain concept (e.g. person, movie) is connected via a property to an age rating:

- *type subUnit and hasInput in [< `SPARQL` > SELECT ?I WHERE {imdb:hasAgeRating rdfs:domain ?I}];*

This condition selects subunits with an input variable type that appears in the domain of the imdb:hasAgeRating property. The square bracket notation is used to indicate a DM query.

Note that this pointcut effectively exploits domain knowledge, and alleviates the burden of the adaptation engineer to (manually) find out which domain concepts have an age rating specified. Such a pointcut cannot be specified using AM information alone, unless the designer explicitly made sure the AM rigorously "marks" the places to be selected in the pointcut. Naming conventions such as adding a label named "age-rating" could for example do this. In that case, the following pointcut would achieve the desired result:

- type subUnit and contains (type Attribute and hasLabel "age-rating");

Advice

Advice specifies exactly what needs to be done to the element(s) selected in the pointcut. SeAL supports adding conditions, as is commonly done in many approaches, yet also allows arbitrary transformations of (elements of) the AM.

Conditioning elements is the common way of adapting. Conditions (condition expressions) typically reference the context data to express a kind of context or user-dependency. In analogy with Hera-S notational conventions (see chapter 3), referencing the user's context is done using the "cm:"-prefix; more specifically, the current user node is used as a starting point to navigate through the context data. Analogously, referencing the domain data is done using the namespace-prefix from the Domain Model (e.g. "imdb:"); in this case, the domain resource(s) corresponding to the element(s) selected in the pointcut is used as a starting point for navigation. Navigation through the data can be done using the

"."-operator: for example, imdb:hasAgeRating.minAllowedAge returns the minimum age of a domain resource corresponding to an (AM) element that was selected in the pointcut. Note that by just providing the CM namespace prefix (e.g. "cm"), the user (node) can be referenced directly. Using "this" in the pointcut allows to explicitly reference the elements selected in the pointcut. Both are illustrated in the third example below.

As usual, condition expressions can be combined using logical conjunction, disjunction or negation, and can have parentheses altering the standard operator precedence. The semantics are such that depending on the truth value of the condition, the particular element is then shown or not. In adaptation engineering the most common practice is to include (add) conditions when a new adaptation concern is considered. Typical examples include:

- *ADD CONDITION cm:age >= 18;* (adds a condition to the elements selected in the pointcut denoting that the age of the user should be 18 or above, i.e. user is not a minor)
- *ADD CONDITION imdb:hasAgeRating.minAllowedAge <= cm:age;* (adds a condition to the elements selected in the pointcut which specifies that the age of the current user should be higher than the minimum allowed age for these elements).

Note that in the 'ADD'-part of the above example, navigation starts from the elements selected in the pointcut, while the actual condition specifies navigation in the context data (denoted by the "cm:" prefix) for the current user. In some cases, this notation will be insufficient. For example, consider the following case where we do not want to show elements for which the current user has given a low rating. Using only the constructs introduced above, this would result in:

- *ADD CONDITION cm:givenRating.about != this or cm:givenRating.rating > 5;* (adds a condition to the elements selected in the pointcut, stating that either the user has not yet given a rating to this resource, or that the user gave a rating over 5)

It can be seen that the above condition is flawed, because it states that the user either never rated the resource or that a(ny) rating over five was given: the rating could belong to a completely different resource. Therefore, it has to be checked whether the rating that is being checked actually belongs to the resource in question (i.e. resource returned by the query). In order to combine several checks in one navigational path, we use the ";" notation:

- *ADD CONDITION not cm:givenRating.about = this; rating < 5;* (this condition states that in case the user has entered a rating for the resource, that rating should not be below 5)

(New) elements can be added to the elements selected in the pointcut if a certain condition is fulfilled, or existing elements selected in the pointcut can be deleted. When adding elements, plain RDF(S) elements can be added (the designer is responsible for validity), or it is possible to use ADD something statements where something is any of the AM elements. Typical examples include:

- *if (cm:age < 18) { DELETE };* (simply deletes the elements selected in the pointcut if the current user is a minor, i.e. cm:age < 18)

- *if (cm:bandwith >= 1000) {ADD Attribute containing rdfs:label "Trailer", hasQuery "SELECT ?T WHERE {$var a imdb:Movie; imdb:movieTrailer ?T .}"; };* (adds, to the element(s) selected in the pointcut (movies), an AM-attribute showing the trailer, with the label "Trailer" and the corresponding query, if the user's bandwidth is above 1000 Kbps)

(Parts of) existing elements selected in the pointcut can also be replaced, if a certain condition is fulfilled. Only the explicitly specified parts of the elements are replaced; parts which do not appear in the replace-statement are simply copied. As with the pointcut expressions, pattern matching symbols may be used to match (part of) the element to be replaced (see the first example below). Typical examples include:

- *if (cm:userDevice="mobile") { REPLACE refersTo :LargeUnit BY refersTo :SmallUnit; };* (if the user uses a mobile phone, let relationships/subunits refer to a smaller version of the unit)
- *if (cm:userDevice = "mobile") { REPLACE subUnit BY NavigationRelationship; };* (replaces the subunit elements by navigational relationships, i.e. part of the content of a page is shown on a seperate page)

An arbitrary number of adaptation actions and conditions can be combined in a single aspect advice, as to avoid having to specify multiple aspects with the same pointcut.

Some Examples

Given the description of pointcuts and advice in the previous sections we can now consider some examples for our movie application. For each example, we will first state the adaptation requirement, and subsequently formulate the adaptation aspect realizing this requirement, followed by a small explanation. We will gradually show the strength of our approach, and illustrate and motivate how we benefit from our aspect-oriented approach, and how we exploit semantic information and structural properties of the AM when specifying adaptation aspects.

We start off with a simple adaptation requirement, affecting only one particular element in the design: "For users who specified that they do not want any spoilers, don't show the plot outline for a movie."

```
POINTCUT: rdfs:label ‘‘Plot" and
        containedIn (type NavigationUnit and hasName :MovieUnit);
ADVICE: ADD CONDITION cm:spoilerInfo = true;
```

The pointcut selects the AM-element (in our case an attribute) which is labeled "Plot" and is contained in the "MovieUnit" unit. The advice adds the relevant condition to only show this "plot" attribute if the user doesn't mind spoilers (i.e. cm:spoilerInfo = true). This first example is somewhat trivial, and corresponds to typical condition-based approaches: the desired adaptive behavior is specified on one particular attribute from a specific unit. If this is our only adaptation need we might as well integrate this in our original AM. However, imagine there is another movie unit present in the AM (e.g. an elaborated version), which also shows the plot. In this case, another aspect specification would be required to also restrict visibility of this plot information, similar to typical condition-based approaches which require manual specification of conditions on all affected elements. The only advantage we have gained here is the fact that our adaptation specification is separated

from the (regular) Web design; for the rest, the same drawbacks as for condition-based approach exist.

The use of aspect-orientation becomes clearer if we look at more advanced examples. For example, consider that we want to restrict the navigation for non-registered users to top-level links.

```
POINTCUT: type NavigationRelationship
        and from (type subUnit) and to (type NavigationUnit);
ADVICE: ADD CONDITION cm:isRegistered = true;
```

This pointcut selects all relationships, i.e. links, which originate from a subunit and target a unit. The advice indicates to add to these relationships the condition that the (current) user should be registered. Thus, the above adaptation aspect will present the non-registered user with a restricted view on the Web application: only top-level links (i.e. those appearing in units) will be shown, any link that originates from a sub-unit and targets a top-level unit, and thus typically presents an elaborated view on the particular concepts represented in the sub-unit, will be hidden. This adaptation concern is clearly cross-cutting: it is not localized, yet spread over the entire AM. In our movie application execution of the advice affected fifteen sub-units linking to top-level units. Note that this adaptation aspect exploits the structural (i.e. navigational) properties of the AM, yet is perfectly re-usable over (different) Web applications, as the adaptation is not specified in an application-specific way.

The previous example basically hides links according to a specific property of the user as stored in the Context Model. Many web applications use a slightly different pattern, namely showing the link but clicking it will lead to a registration page instead of the desired content. This adaptation requirement is depicted below.

```
POINTCUT: type NavigationRelationship and
                from (type subUnit) and to (type NavigationUnit);
ADVICE: if (not cm:isRegistered) {
                  REPLACE refersTo * BY refersTo :RegistrationUnit;
                };
```

The pointcut remains unchanged, selecting all relationships originating from a sub-unit and targeting a unit. However, the advice doesn't simply add a condition to the selected relationships. The actual adaptation that is performed is thus not filtering, as is typically done in condition-based approaches. Instead the advice replaces, if the user is not registered, the value of the 'refersTo' attribute (whatever it was) to the ":RegistrationUnit" unit, causing all selected relationships to be redirected to the registration unit. In this way, the elements of the AM are actually (conditionally) altered, completely changing their behavior.

The next example demonstrates exploitation of metadata present in the DM to perform cross-cutting adaptation. Here we try to make sure that we do not show any age-restricted information to minors.

```
POINTCUT: type subUnit and hasInput in
            [SELECT ?node1 WHERE {?node1 imdb:hasAgeRating ?node2}]
ADVICE: ADD CONDITION imdb:hasAgeRating.minAllowedAge <=  cm:age;
```

The pointcut selects all sub-units that have as input a certain concept which has an "imdb:hasAgeRating" property specified in the DM (the latter part is represented by the native SPARQL expression). The advice adds a condition to these sub-units, denoting that they should only be shown if the age of the user (specified in the context data) is higher than the minimum allowed age (specified in the domain data) for that resource. Note that in this example, no specific AM elements are specified in the pointcut. In other words, at specification time, it is not known which elements will or will not have an 'hasAgeRating' property. Only at runtime it will be determined which elements correspond to concepts that have an 'hasAgeRating' property specified in the DM and subsequently the corresponding elements will be selected from the AM. This clearly illustrates that the burden of identifying the place(s) in the AM where a certain adaptation needs to be performed is alleviated from the application engineer. Instead, these places are specified in a declarative way, using semantic meta-data present in the DM. This adaptation aspect is rather robust. Even when new resources are added to the Web application (imagine for example that we also add games to our application), and therefore adding new units and/or sub-units describing these resources, the adaptation aspect will subsequently also identify these new resources and their corresponding sub-units, and restrict their access/visibility accordingly. This results in consistent adaptation throughout the application even when the application changes.

Finally, in the last example we will perform some adaptation based on a complex, global property of the AM and combine multiple advice actions and conditions in a single aspect. It models that for users who use a mobile device we move content from pages that are (too) large to dedicated pages and notify the user about this adaptation.

```
POINTCUT: type subUnit and contains (count (type Attribute) >=  10) ;
ADVICE: if (cm:userDevice.type = ‘‘mobile") {
  REPLACE hasUnit BY hasNavigationRelationship;
  ADD attribute containing label "Adjusted to device!",
      query "SELECT ?L WHERE {?V a imdb:MobileImage;
                                imdb:imageURL ?L}";
  if (cm:userDevice.screenSize < 250) {
    REPLACE refersTo :LargeUnit BY refersTo :VerySmallUnit;
  } else {
    REPLACE refersTo :LargeUnit BY refersTo :SmallUnit;
  }
};
```

The pointcut selects all subunits which have ten or more attributes (i.e. "large" sub-units). Firstly, the advice replaces these sub-units (actually their in-page occurrence) by relationships to the corresponding units, to avoid crowding small screens. In this way, users will only see information directly relevant for the current page, and be presented with links to related information instead of seeing it in-page. Secondly, the fact that adaptation to the user's device has occurred is indicated by adding an attribute that displays an image of a mobile phone. Finally, the links of the relationships are replaced by links to smaller versions of the target page. If the screen is very small even smaller versions will be served.

Note that this adaptation aspect exploits characteristics of the entire AM, not just one single element, in this case the amount of attributes. We stress that this kind of behavior cannot be reached by current approaches which simply rely on specifying adaptation (conditions) to a single element. Furthermore note that the actual adaptation that is performed is not filtering (as is commonly done), but instead alters AM elements.

4.4.4 Implementation

The parser for the pointcut part of the SeAL language was constructed using the JavaCC parser generator[3], while the Javacc JJTree tool was used to automatically generate AST (Abstract Syntax Tree) classes. This tool also provides support for the Visitor design pattern, which is used here to traverse the AST corresponding to a given pointcut expression. Each pointcut condition is translated into a SPARQL query that extract the elements from the AM that satisfy the pointcut conditions. The queries are sent to Sesame as Sesame is used in the Hydragen implementation to store all the models. Aggregate functions do not exist yet in the current SPARQL specification nor in Sesame's current SPARQL implementation. This also holds for subqueries. Therefore, SeAL generates several SPARQL queries of wich the result were post-processed in Java objects. This results in some possible performance penalties for pointcuts with big resultsets, which is not typical however. The result of this process is a set of RDF subgraphs stored in Java objects.

The implementation of the advice uses a similar approach: the JavaCC parser generator and the JJTree tool were used to generate AST classes. These classes allows us to rewrite the RDF graphs and the SPARQL queries (even though these are implemented as separte packages). The result of an advice is an RDF graph. This graph is used to update the pointcut result, i.e. the RDF subgraphs. This results in updated subgraphs that can be stored back into Sesame. Given that SPARQL nor Sesame natively support update queries, a proper update is done by first deleting the result subgraph and then storing the updated subgraph. Luckily Sesame has a rudimentary support for transactions, which allows for applying these updates atomically.

An important design decision had to be made concerning the execution time of the aspects. We finally decided to apply all aspects during runtime. This has some advantages and disadvantages. The advantage is that for aspects that use volatile information (e.g. from the CM or DM) always the most recent information is used. Especially for aspects that depend on the typically volatile CM, runtime evaluation is the only viable option. Another point of concern tackled by runtime execution is reversibility. Only applying aspects on runtime data and not the actual AM models allows to remove the aspects without the issue of reversing all AM data in Sesame. Disadvantages are however a performance penalty and the need to do some integration work to let Hydragen and SeAL work together. The advantages do outweight the disadvantages, which is why we chose runtime evaluation. Some of the disadvantages could be alleviated by optimization techniques where aspects that influence only the AM are executed at compile time.

4.5 Presentation Generation

Hera-S' main focus is on generating AMP's (refer to Section 3.2). AMP's contain a logical structure of the data that is to be presented to the users, but does not contain actual presentation details like ordering, style, etc. In Section 3.5 we presented a simple PM model which allows us to model presentation details for our Web application. However, Hera-S also allows other specialized components to take over this task. We chose to create an implementation using the AMACONT presentation engine [Fiala et al., 2003].

This approach aims at implementing personalized ubiquitous Web applications by aggregating and linking configurable document components. These are instances of an

[3] `http://javacc.java.net/`

XML grammar representing adaptable content on different abstraction levels. As Figure 4.6 shows, components in AMACONT can be classified into three different layers, with an orthogonal fourth Hyperlink layer.

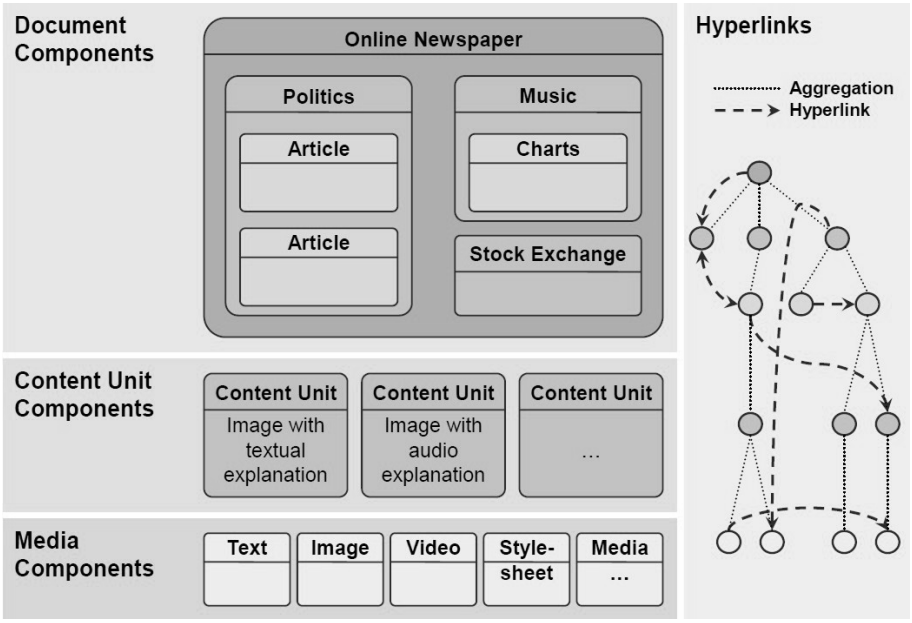

Figure 4.6: AMACONT Component Model

The most basic components stem from the bottom layer of Media Components. Examples of such elements are text fragments, images, CSS stylesheets and videos. Often there are fragments that are closely related and typically used in conjunction, like an image with a textual caption. By defining such semantic groups on the Content Unit layer of AMACONT, authors can structure their applications. Finally, there is the Document Component layer. It allows to structure a web application further by grouping Content Units into larger components and further into web pages. AMACONT allows arbitrarily complex subcomponents, so that authors have the freedom to organize their page structure in any way they want. Note that web pages in AMACONT can be either modeled by structuring one document accordingly or by splitting the definition up into a number of documents, similar to HTML files. Orthogonal to these three layers, the Hyperlink layer is spanned. Hyperlinks in AMACONT can reference components from all three levels. Thereby, it is easy to create a link starting from a composed page element. Links can target either an AMACONT file or an arbitrary component - or any custom URL. AMACONT also has an Eclipse-based authoring tool, the AmaArchitect, which is a graphical editor for creating presentation models.

Whereas the AMACONT document model provides different adaptation mechanisms, here we focus on its presentation support. For this purpose it allows to attach XML-based abstract layout descriptions (layout managers) to components. Document components with such abstract layout descriptions can be automatically transformed to a Web presentation in a given Web output format. The PM was specified by adopting AMACONT's layout manager concept to the model level. This enables the automatic translation of AMP's to a component-based Web presentation based on a PM specification. The corresponding presentation generation pipeline is illustrated in Figure 4.7.

In a first transformation step (AMP to Component) the AMP's are translated to hierarchical AMACONT document component structures. Thereby, both the aggregation hierarchy and the layout attributes of the created AMACONT components are configured

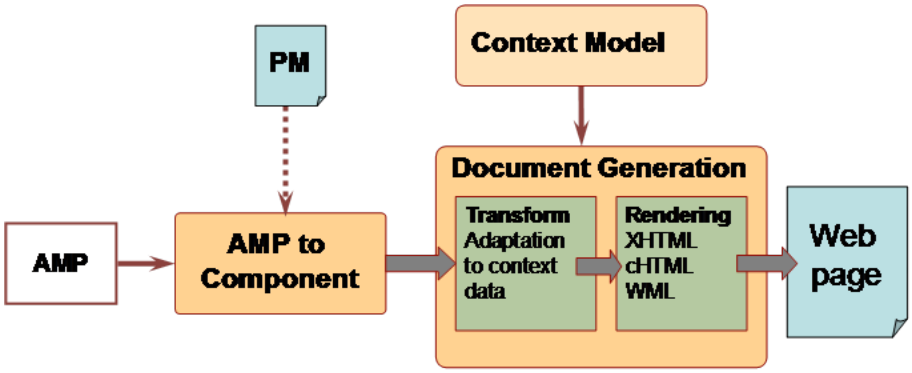

Figure 4.7: The AMACONT Presentation Generation Pipeline

according to the PM configuration. Beginning at top-level document components and visiting their subcomponents recursively, the appropriate AMACONT layout descriptors (with adaptation variants) are added to each document component. This transformation can be performed in a straightforward way and was already described in detail in [Fiala et al., 2004]. The automatically created AMACONT documents are then processed by AMACONT's document generation pipeline. In a first step, all adaptation variants are resolved according to the current state of the context model. Second, a Web presentation in a given Web output format, e.g. XHTML, XHTML Basic, cHTML or WML is created and delivered to the client. For example, Figure 4.8 is a screenshot of the movie Unit (example from Section 3.4.1) for the movie 'The Matrix'.

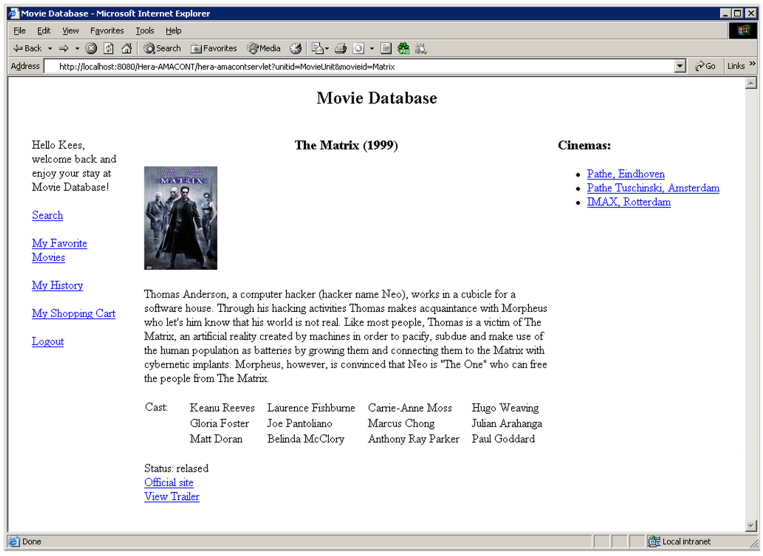

Figure 4.8: AMACONT Presentation on a PC

The rules executed by AMACONT can also be parameterized by the adaptation context data containing up-to-date information on both the user and his browser device (client). AMACONT relies on CC/PP, an RDF grammar for describing device capabilities and user preferences [Klyne et al., 2004a], and its related specification UAProf [Wireless Application Protocol Forum, 2001]. Many devices send a URL containing a link to its CC/PP or UAProf profile. This profile contains some device capability characteristics that can be used for adaptation. Note that newer devices that have browsers that have extensive scripting capabilities typically do not send these profile links anymore. Instead a combination of Javascript and AJAX technologies could be used in the browser to obtain

the same information.

In this way, by setting up the right rules, AMACONT allows for adapting to the user context and associated preferences. For example the following code depicts an excerpt from the adaptation context used in our example for a PDA. Figure 4.9 is the resulting generated presentation for a PDA.

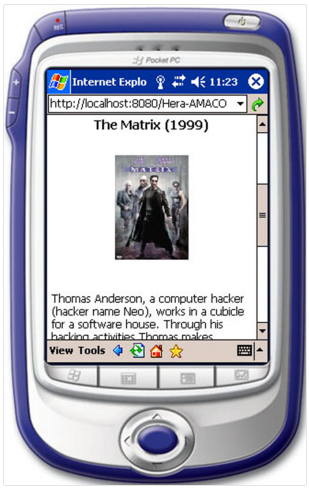

Figure 4.9: AMACONT Presentation on a PDA

```
<AdaptationContextData>
    <Description ID="DeviceProfile">
        ...
        <Description ID="BrowserUA">
            ...
            <Description ID="Window">
                ...
                <InnerSizeX>1024</InnerSizeX>
                ...
        ...
    <Description ID="UnitVisited">
    <Unit.movie.imdb_ID1>true</Unit.movie.imdb_ID1>
    <Unit.movie.picture_ID3>true</Unit.movie.picture_ID3>
...
</AdaptationContextData>
```

4.5.1 Aspect-Orientation in AMACONT

AMACONT employs traditional AOP concepts: the base is a regular web application, that is, typically without adaptivity. On top of such existing web applications, authors can instantiate adaptation concerns (Figure 4.10) that group adaptation on a requirement-level. Typical examples of adaptation concerns are device independence, location awareness, internationalization or role-based access. Because concerns offer only a high level view, adaptation is further subdivided into adaptation aspects. These are medium-sized adaptation fragments that allow authors to impose an order on the weaving process, which is needed for a predictable result, as the weaving order generally plays a decisive role in AOP (e.g., see [Nagy et al., 2005]).

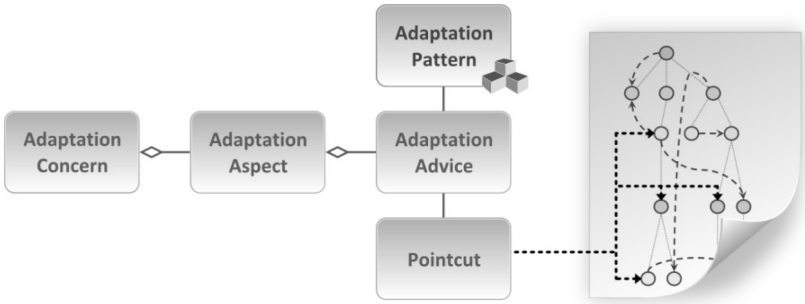

Figure 4.10: Aspect Model of AMACONT Adaptation

The smallest possible piece of adaptation is an advice. An advice describes how to transform a given component within an AMACONT document. To this end, they may instantiate adaptation patterns, i.e., reusable transformations that resemble typical adaptation techniques like the ones identified by the hypermedia community [Brusilovsky, 2007]. The adaptation patterns available to the author differ, depending on the adaptation concern he is working on. Several patterns have been implemented and are available for AMACONT users, e.g. first sentence elision, image rescaling or layout change for instant use. Still, authors are free to define their own patterns to achieve any adaptation effect they want. An advice also contains an adaptation precondition that specifies the circumstances under which to perform the transformation. Finally, an advice is connected to the web application via the pointcut. The pointcut, expressed in XPath, is a reference to one or more AMACONT components. Therefore, it is possible to easily apply one advice to an arbitrary number of components. An adaptation model based on aspects can be stored either for every AMACONT document or for the whole web application. The aspect weaver then has the task to merge this model with the base application.

For weaving aspects, AMACONT offers two general choices: weaving at design time and weaving at runtime. Design time weaving offers one obvious advantage: Because it is a one-time process of generating alternative variants, it creates only negligible overhead. Then, at run time the server only needs to decide which (pre-made) variant to choose for a user. However, the greatest disadvantage of design time weaving is the inability to cover dynamic adaptation. For example, it is not possible to insert a user's name into the document, because that data is not known at design time. In contrast, run time weaving shows opposite characteristics: Working with dynamic data is no longer a problem, while the downside is that it requires weaving at every request, thus reducing performance. But weaving at run time offers some more advantages: depending on the server load, it becomes possible to "deactivate" costly aspects temporarily. Moreover, it is possible to create alternative aspect configurations that are chosen dynamically based on the context. For example, authors could model three different levels of location awareness and base the selection on whether the user is a paying customer. With aspect-oriented adaptation, web engineers are able to nicely separate adaptation from the rest of the web application.

For aspect-orientation, the document generation pipeline was extended by adding another transformation component before the adaptation transformer. This way adaptation aspects can be weaved at runtime, while still allowing authors to use the more generic static variants, which are then processed by the existing adaptation transformer. The aspect weaver takes as input an AMACONT document together with an adaptation description. It then tries to match the advice's pointcuts as specified in the adaptation description with the components in the input document. The following code snippet shows an extract

from the aspect-oriented adaptation definition, resizing all images in the navigation bar if the user accesses the application with his PDA.

```
<aspect id="RI1" concern="ResolutionIndependence">
    <advice>
        <pointcut>
            <condition>
                <adaptationclass>Device_PDA</adaptationclass>
            </condition>
            <target>
                <xpath>//*[@id="nav"]/aco:AmaImageComponent</xpath>
            </target>
        </pointcut>
        <pattern id="ReduceImageSizeByTranscoding">
            <parameter id="ratio">0.5</parameter>
        </pattern>
    </advice>
</aspect>
```

If a match is found, the advice's transformation is executed, modifying the affected components. Every Advice is executed one after another, depending on the order specified by the author.

4.5.2 Semantics-Based Adaptation

One useful way for improving and making adaptation more sophisticated is to incorporate semantic web technology in AMACONT, allowing us to specify the meaning of the data by using meta-data and ontologies, and define their relationship. This enables applications to couple, exchange, and reuse data from different external data sources, including data resources on the Web. Consequently, this is useful to reduce the knowledge load for adaptation conditions, as well as to simplify queries, or even use knowledge that is not available at design time. The semantic web technology also makes reasoning about the data by applications possible, allowing the semantic gap between data to be bridged. Therefore, by exploiting the semantic web technology, AMACONT is able to provide adaptation in a richer and more flexible way (only somewhat related work we know of is [Krištofič and Bieliková, 2005]).

To illustrate the power of semantic-based adaptation consider the following example. Suppose we want to select the language in which we present pages in our application, based on where a user lives. In this way we could choose to give someone who lives in the Netherlands a Dutch version of the page, and other users the English version of the page. However, suppose we have a user model that contains the following location information about a user (in abbreviated Turtle syntax):

```
user:Kees :livesin _:cityX .
_:cityX a :city;
        :cityname "Eindhoven" .
```

In other words, we only know in this case that user "Kees" lives in a city called "Eindhoven". The question is now where we get the knowledge from to understand that Eindhoven is in the Netherlands so that we should give user "Kees" the Dutch version of

the page. One possible solution would be of course to extend the user model so that it also contains the information that user "Kees" also lives in the country of Netherlands, and ask the user for this information. However, this would bother the user with additional questions while we could already deduce the country from the city if we would combine knowledge. Even though it might still be possible to ask the user in this concrete example, one can easily think of cases where this is not viable or even possible (e.g. if the information is not known at design time). Another possibility would be to extend the adaptation query with an enumeration of all Dutch cities, which obviously is also not really viable as it would be extremely laborious to write such queries and their performance would suffer dramatically. We take another approach by exploiting the knowledge of external ontologies to solve our granularity problem. Many of such ontologies are freely available on the Web and can be reused for our purposes. Consider the following aspect-oriented adaptation rule in AMACONT:

```
<aspect id="ML1" concern="Internationalization">
    <advice>
        <pointcut>
            <condition>
                <aada:Sparql>
                  SELECT ?country
                  WHERE { $CurrentUser :livesin ?country .
                          ?country a :country ;
                                   :name "Netherlands" . }
                </aada:Sparql>
            </condition>
            <target>
                <xpath>//AmaTextComponent</xpath>
            </target>
        </pointcut>
        <pattern id="ReplaceComponentByName">
            <parameter id="replacePattern">nl/%name%</parameter>
        </pattern>
    </advice>
</aspect>
```

This aspect-oriented adaptation condition expresses that if the user is from the Netherlands (denoted by the SPARQL query to the user model), we want to replace all (English) text elements by their Dutch translation (with path nl/%text%). If the user is not from the Netherlands we just present the default English text (and thus do not adapt anything). Please note here that $CurrentUser is an AMACONT variable (denoted by the $-sign) that is substituted at run time with the actual identifier for the current user. We now have to deal with the fact that the {geo:livesin ?country} pattern is not in our user model. Therefore, we first select an appropriate ontology that allows making the semantic connection we need, in our case between a city and the country it resides in. A good candidate for this case is the GeoNames Ontology[4]. This ontology simply provides us with additional knowledge:

```
geo:2756253 geo:Name "Eindhoven" ;
            geo:inCountry geo:NL .
```

[4] http://www.geonames.org/ontology/documentation.html

In plain English this means that we have an object with id "2756253", which apparently represents the city of Eindhoven and has a direct property which connects it to Netherlands ("NL") via the geo:inCountry property. This is a very fortunate simple example, as every location within a country has this inCountry element. However, it could also be necessary to follow more complex paths. If we need more information about the country 'the Netherlands' (e.g. its population), we would have to follow the complete chain

Netherlands ← North Brabant ← Municipality Eindhoven ← Eindhoven

to find the Netherlands URL, namely geo:2750405.

Based on this ontology we can conclude that the user lives in Eindhoven, which is part of the Netherlands, and therefore also lives in the Netherlands. In order to automatically derive that fact, we of course need to do some configuration in AMACONT. We first need to align the location name in our local ontology (e.g., "Eindhoven") with the location name in the geo ontology (e.g. also "Eindhoven" or "Municipality Eindhoven"). If these concepts are identical (as in our example), then no further configuration is needed in AMACONT. Otherwise, we can make a manual alignment for every user, or a (semi-) automatic alignment. The last can be configured by specifying two SPARQL queries. We then first specify the following construct in our AMACONT configuration to indicate an alignment step:

```
:align [ sourceConceptQuery "query1 details" ;
         targetConceptQuery "query2 details" ; ]
```

Now we fill in the query details. For the source concept we specify which concept in the user model we want to connect with a target concept of our specialized ontology. In simple cases we allow for simple declarations like for "query1" in this case:

```
?city a um:City
```

Here we specify that we want to select all concepts of type city of the user model vocabulary, i.e., the only variable. For "query2", the target concepts, we specify a (somewhat simplified) SPARQL query of how to find the aligned city in the geo ontology. In this query we can align the result by reusing it in query1 and refer to it with the variable $city. The "query2" could then look like this:

```
PREFIX geo: <http://www.geonames.org/ontology#>
SELECT ?feature
WHERE {
        ?feature a geo:Feature ;
                 geo:featureCode geo:P.PPL ;
                 geo:name ?name ;
                 geo:population ?population .
        FILTER (regex(?name, $city))
}
ORDER BY DESC(?population)
```

In this query we are looking for a "feature" (in GeoNames terminology) that has the featureCode geo:P.PPL (code for city) and the name should contain the original city name indicated by the AMACONT variable $city in the FILTER clause. Note that we assume that the target location here contains the city name of the source as a substring (i.e., this

might be a limitation for more complex searches that cannot be expressed like this). We also include "?population" in our search to solve ambiguity issues. Suppose that there are more cities in the world that are called Eindhoven (in this example there are none, but there are many examples of cities that have the same name), so we have to find a heuristic to determine the most probable target concept. For our example, we use the heuristic that the most populated city is the most probable one. We order the results by population and, in the case of more than one result, we use the first result as the target concept.

We have now aligned concepts in our user model to concepts in our helper ontology. After alignment we want to specify a reasoning rule that defines how knowledge in the helping ontology extends the knowledge in our user model. In our case, we define an entailment rule that specifies that if a user lives in a city, he also lives in the country in which that city is located. This rule is basically of the form:

$$\left.\begin{array}{r} \langle ?X : livesin \;\; ?Y \rangle \\ \langle ?Y \;\; a \;\; : city \rangle \\ \langle ?Y \;\; geo : inCountry \;\; ?Z > \rangle \end{array}\right\} \Rightarrow \left\{\begin{array}{l} \langle ?X : livesin \;\; ?Z \rangle \\ \langle ?Z \;\; a \;\; : country \rangle \end{array}\right.$$

This (syntactically simplified) inference rule specifies that if X lives in Y of type city, then X also lives in the country in which Y is located. By applying this entailment rule, we can deduce that user 'Kees' who lives in Eindhoven, also lives in the Netherlands and thus should get the Dutch welcome page. With configurations like this, AMACONT can now effectively use the external ontology to solve granularity issues in our user model.

4.6 Summary

In this chapter we discussed several extensions of the Hera-S toolset.

First, we investigated the issue of data integration. It was explained how well-suited RDF is for data integration, and that datasets can be coupled by a number of simple links. We showed that existing data sources can be integrated in a number of steps. First, they have to be transformed into RDF, e.g. using one of the existing approaches for that. Next, the overlapping knowledge needs to be linked together. This is a topic which we will discuss in the coming chapters. And finally they have to be presented to the user in an integrated way. Partially this can be achieved within Sesame, but for more robust performance custom solutions have to be created.

We also presented the Hera-S model builders. These builders provide designers (partial) graphical support to create the domain, context and application model. They also provide some model checking facilities to make the process of creating these models a bit less error prone.

Next we described an extension of Hera-S Web applications via aspect-orientation with a domain-specific language called SeAL. Aspect-orientation provides a better separation of concerns and better maintainability by plugging in functionaly by querying and adapting the application model using pointcuts and advice. This method provides compacter and better understandable AM code by separating the code for seperate functionality, like adding device adaptation or shopping carts, which would otherwise would modify the entire AM model. This functionality can be easily turned off or on, without disturbing the rest of the application.

Finally we zoomed into presentation generation with AMACONT. AMACONT can be plugged in together with Hydragen and works as a seperate functional component that offers specialised presentation generation functionality. Also AMACONT can be

extended with aspect-orientation similar to our AM aspect-orientation. Next to that we presented how to use semantic based adaptation by combining user model data with external knowledge sources.

Chapter 5

Data Integration Using Semantic Web-based Techniques

One of the most interesting aspects of Semantic Web techniques is the power to exploit connected data sources. Many existing data sources would get a lot richer if coupled to others. In this chapter we look into various aspects of coupling existing data sources. We look at creating links between similar concepts in different datasets, as well as merging data sets together into one data set. We also look at the process of transforming a data set which is an instance of a certain schema to a data set which is an instance of another schema, by using a transformation language. The expression for transforming data between schemas can be generated semi-automatically by using schema matching techniques. Therefore we apply schema matching techniques, as well as more semantical techniques, and techniques that exploit the knowledge of a user base. We also show that matching techniques can be applied in relating user-generated data to semantically well structured datasets. User-generated data is important to overcome cold start problems, i.e. the lack of metadata of some systems that contain multimedia objects. A common characteristic of user-generated data is that it is structureless which differs from structured data found in e.g. relational databases.

5.1 Introduction

The power of the Semantic Web might not be the novelty of techniques used for data modeling, but rather the most powerful feature of the World Wide Web applied to (semi-)structured datasets: linking data. Traditionally, links between web pages are created manually. A similar approach could be assumed for the Semantic Web for relationships between concepts. However, the manual approach is problematically laborious for the enormous amount of existing well-structured datasets that are currently available online. These existing structured data sources can be quite large which makes that finding and explicitly creating links between datasets is very laborious and in most cases unfeasible. Therefore an automated approach would be very helpful in these cases.

The Semantic Web provides a generic data structure, which was especially designed for expressing metadata. Many datasources however lack any (meaningful) metadata, like many multimedia collections. One way to bridge this gap is by obtaining this metadata from users who are interested in browsing through these collections. One cannot expect

well structured metadata from users though, nor even ask the lay users to use a graph-based data modeling paradigm like RDF. In many situations these users would prefer a mechanism like the well-known tagging facilities that are available in many web 2.0 applications to provide metadata. A characteristic of user tags is that they are structureless. As many of the interesting Semantic Web applications exploit the semantic structure of metadata it is interesting for this purpose to bring in (existing or specifically fabricated) ontologies. The challenge here is to find the relation between user tags and the conepts in these ontologies so that through these relations the underlying structure of the ontology can be exploited on behalf of the underlying Web application.

In this chapter we explore ways to integrate data sources. First we look at various integration strategies. In Section 5.2 we look at connecting two datasets by making relations between concepts in the ontologies. Then, in Section 5.3 we look at coupling datasets by using a standardized or common ontology between them. Finally, we look at a transformation strategy that directly mappes data between two data schemas in Section 5.4. For this last strategy we define a data transformation language and in Section 5.5 we present ways for semi-automatically deriving transformations.

5.2 Data Coupling

In this thesis we are interested in building (Semantic) Web applications. In these application the main reason for using Semantic Web techniques to couple data sources is solving the data integration issue: how to couple two or more data sources so that we can exploit them as if they were one homogenous datasource. In this chapter we discuss three data integration strategies, namely: coupling (this section), merging (Section 5.3) and transformation.

The Semantic Web offers ways to connect concepts in different data sets via relationships by using RDF statements. Consider the following code fragments:

```
<rdf:Description rdf:ID="m13541">
   <movie:title>The Good, the Bad and the Ugly</movie:title>
</rdf:Description>
```

and

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:films="http://films.nl/prog/film/"
   xml:base="http://films.nl/prog/film">
  <rdf:Description rdf:ID="x234">
   <films:film_properties rdf:parseType='Resource'>
    <films:original_name>Il buono, il brutto, il cattivo</films:original_name>
    <films:aka_literal>The Good, the Ugly, the Bad</films:aka_literal>
    <films:aka_us>The Good, the Bad and the Ugly</films:aka_us>
   </films:film_properties>
 </rdf:Description>
</rdf:RDF>
```

Even though both fragments refer to the same movie both the RDF structure and the used syntax are completely different. The first source uses one simple title attribute only, while the second source has a more complex structure in which original titles and translated and international titles of movies are distinguished. If we want to exploit and

combine the information from both data sources about this film we somehow have to know that these code fragments denote the same concept. The most straightforward way to do this reconciliation is using the mechanisms in the OWL language that support this. By using the OWL sameAs property one can denote equality of individuals. If we would alter the first code fragment as follows:

```
<rdf:Description rdf:ID="m13541">
   <movie:title>The Good, the Bad and the Ugly</movie:title>
   <owl:sameAs rdf:resource="&films;x234"/>
</rdf:Description>
```

we effectively coupled the two RDF movie fragments by claiming they are about the same movie.

The biggest advantage of this approach is its simplicity. By adding just one triple one can specify equality between concepts. A second advantage is that the triple can be written down independent of the two original sources. This coupling-statement can be written in a third RDF document independent from the first two sources, which might not be under our control nor locally present. This for example allow us to couple datasources which might not be under our control, similar as we are able to make our webpage in which we claim or review other web resources.

A disadvantage of this approach is illustrated by Figure 5.1. When we explore the three datasets (the two sources, plus the coupling statement) we find that there is one concept which has two identifiers and which has two different ways of expressing a "title" property, namely one direct datatype property, and a composite datastructure via the "Film Properties" property and a blank node. So even though we have now expressed that two concepts are identical, this still leaves graph structure differences in which similar properties are expressed. In simple datatype property cases this can be overcome by also specifying a 'sameAs' relationship between properties. However, this does not apply to our example as the structures are different and moreover the properties 'title' and 'Original Name' might even be semantically incompatible. This is a realistic issue that applies to many independently developed datasources.

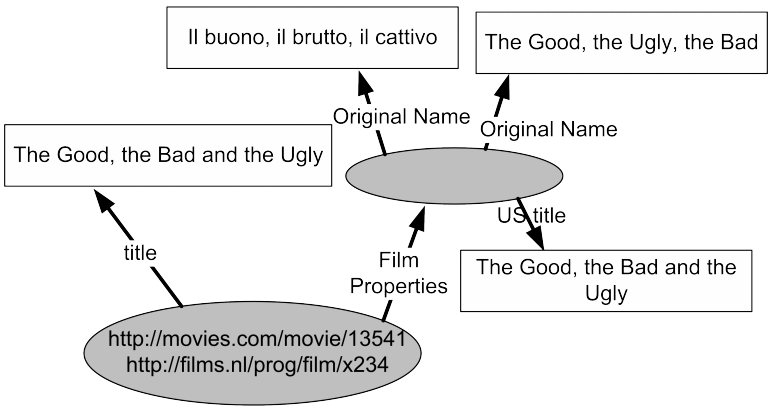

Figure 5.1: Integrated view over coupled dataset

In practice the mentioned structure incompatibility is even worse as available reasoners will not perform inter related reasoning between datasets because of the distributed nature of the data, but only within datasets. So the mentioned sameAs relationship between movie:m13541 and films:x234 will only lead to the deducted node as shown 5.1 if all RDF

fragments are in one repository. In all other cases actual semantic integration of this data still needs to be done by manually storing the data in one datastore.

These issues have consequences for querying datasources that are coupled in this way. It means that in order to query for a certain property in our integrated dataset one needs to know the schema descriptions of all original datasources and enumerate every possible path to the property. Moreover, either the original datasources need to be queried separately where the results for those queries still need to be integrated, or one combined query expression needs to be written that searches and enumerates all possible structures and returns an integrated result.

A SPARQL query that queries for all possible titles of a movie in our example is:

```
SELECT ?title
WHERE  { { ?movie movie:title ?title } UNION
         { ?movie films:film_properties _:propnode1.
           _:propnode1 films:original_name ?title } UNION
         { ?movie films:film_properties _:propnode2.
           _:propnode2 films:aka_literal ?title } UNION
         { ?movie films:film_properties _:propnode3.
           _:propnode3 films:aka_us ?title }
}
```

This query enumerates all possible paths from a movie concept to its title and integrates the results. The answer to this query for our example RDF fragments is:

?title
"The Good, the Bad and the Ugly"
"Il buono, il brutto, il cattivo"
"The Good, the Ugly, the Bad"
"The Good, the Bad and the Ugly"

In the example the query is not overly complex and for simple examples and applications using OWL relationships for data integration this solution is fine. But given an integration of many sources the query can quickly get more complex and become inefficient both in terms of query design and its performance. Especially for user applications of considerable scale this will be unfeasible.

The biggest drawback of this approach is however that its lacks for ways of reconciling semantical differences of literals in different sources. Literals usually have implicit semantics, meaning and value ranges. Consider for example the following different RDF fragments for a name property:

```
<rdf:Description rdf:ID="personX1">
   <person:hasName>Kees van der Sluijs</person:hasName>
</rdf:Description>
```

and

```
<rdf:Description rdf:ID="personX1">
   <person:firstName>Kees</person:firstName>
   <person:lastName>van der Sluijs</person:lastName>
</rdf:Description>
```

Both examples express the name of a person. However, they do it on a different level of granularity. Where one source uses only the full name, the second distinguishes the first and last name. For mapping the first name and last name to full name we need a concatenation function. For the other way around we need a parsing function that can correctly divide the full name into first and last name. Even though SPARQL does have comparison operators in queries, both neither SPARQL nor OWL have operators for literal manipulation. For complete integration and harmonization we will need string based functions, number based operations, date operations, range mappings, and also data aggregation.

5.3 Vocabularies

One way to deal with data heterogeneity and semantic mismatch between applications is using a shared vocabulary. In some cases the application can directly use the shared ontology. However, in general it is difficult to decide on a common world view and many applications have their own specific needs. Another way to use a shared ontology is to use it as an intermediate ontology. This means that every application has its own internal way for representing and storing data, while for sharing data they use an intermediate ontology as a common vocabulary. In this way, for every application a two-way mapping needs to be created to and from the intermediate ontology (as depicted in Figure 5.2). In terms of complexity this means 2N mappings for N applications. As a consequence, a new application can be added to a swarm of `n` applications by adding two mappings, one to and one from the shared ontology.

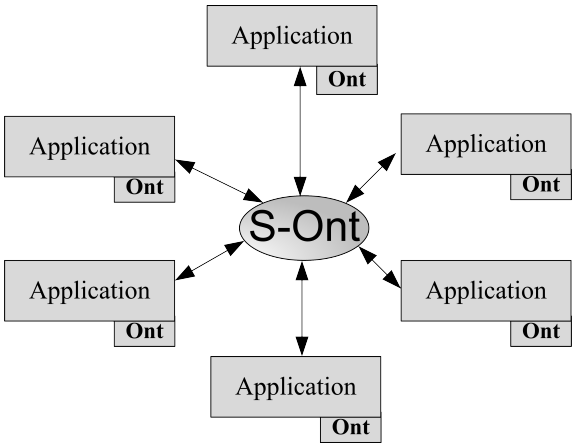

Figure 5.2: Shared-UM approach

Using a standardized ontology is a good solution in many cases, but it is not always a viable possibility. There are actually not many domains for which everyone worldwide can agree on one single vocabulary. This is even more evident for application specific data structure, i.e. the structure that is maintained that represents the state of the application. As every independently developed web application is unique, with unique requirements, design and implementation, one cannot expect to find unique vocabularies that all these applications can agree on. However, the Semantic Web has a major advantage: by design it allows authors complete freedom when designing their own specific ontologies and offers constructs to relate and connect them together with other ontologies. However, in some cases consensus can be reached. There can be good reasons for that:

- Some vocabularies are generic and domain independent and are used to describe often used properties and concepts that are not application specific. In these cases it can be easier to use a standard vocabulary than to come up with one yourself.
- Standard vocabularies typically get developed by domain experts, which pretty much guarantees a well defined structure and semantics.
- There are tools and search engines that interpret and use the fixed semantics of these well-known vocabularies. If a developer chooses such a well-known vocabulary, this tool support comes for free.
- It is easier to interoperate with other parties that use the same standards. Also for domains that do not completely overlap.
- Semantic Web vocabularies are by definition easily extendible and can easily be mixed with application specific vocabularies, i.e. choosing a standard vocabulary is not a limitation for a designer.

Standardized vocabularies certainly offer benefits in terms of interoperability. Therefore a number of standardized RDF vocabularies have been developed or are under development. Commonly used vocabularies are e.g. SKOS, OWL Time, Dublin Core, FOAF and SIOC. Other frequently used sources are the thesaurus WordNet, and the Getty Thesaurus of Geographic Names.

SKOS

SKOS[1] is a data model for knowledge organization systems. SKOS provides structural primitives that can help to structure, in addition to OWL, a thesaurus or other structure datasets. SKOS adds the notion of collections, but also properties like broader, narrower, related, membership, alternative and preferred label, change note and history, etc. SKOS is itself defined in an OWL Full ontology[2]. SKOS itself does not have a formal semantics of its constructs, in the form of formal axioms or facts, only OWL relationships that define their meaning partially. SKOS is currently a W3C Recommendation.

OWL Time

OWL Time[3] is as its name implies a time vocabulary in OWL. It allows not only to express dates, time instants and more imprecise notions like day of the week or day of the year in RDF, but also intervals and duration of time. Besides, it also considers expressing timezone information and using (partial) temporal ordening constructs like 'before', 'after', 'inside'. Together this allows time based reasoning and conflict detections. A typical example of OWL Time supported reasoning is given in the W3C OWL-time document, namely "Suppose someone has a telecon scheduled for 6:00pm EST on November 5, 2006. You would like to make an appointment with him for 2:00pm PST on the same day, and expect the meeting to last 45 minutes. Will there be an overlap?". The OWL Time vocabulary is itself defined by a OWL DL specification[4]. OWL Time is a W3C Draft (and is not intended to become a W3C Recommendation).

Dublin Core

[1] cf. `http://www.w3.org/TR/skos-reference`
[2] cf. `http://www.w3.org/2009/08/skos-reference/skos.rdf`
[3] `http://www.w3.org/TR/owl-time/`
[4] `http://www.w3.org/2006/time`

Dublin Core is a metadata vocabulary developed by the Dublin Core Metadata Initiative (DCMI)[5]. It is targeted to provide an open set of vocabulary terms for which a consensus can be found. The vocabulary is extensible, but also contains a basic vocabulary set, also called the simple set. This contains basic constructs like 'title', 'creator', 'subject', 'description', 'publisher' and 'date'.

FOAF

FOAF[6] is an ontology that can be used for describing persons (e.g. name, title, age, email), the relationships between them (e.g. knows, group membership) and the things they create (document, images) and do (interest, projects, workplace). It allows people to structurally describe who they are, what they do, and via relationships with other people it results in a machine-interpretable web of people. This Web is created by people that place their FOAF profiles on their websites and link to FOAF profiles of their friends and colleagues. Exploiting the semantic structure of FOAF profiles enables queries like "find recent publications by people I've co-authored documents with."

SIOC

SIOC[7] is an ontology that can be used for describing online community websites like weblogs, message boards and wikis. It allows for the basic description of such a site and the relationships between that site and other online community sites. It describes concepts like 'site', 'forum', 'post', 'user', 'role', etc. The SIOC vocabulary can also be mixed with for example FOAF data for describing the persons on the online community website. SIOC exporters for many weblogging and other frameworks exist.

WordNet

WordNet[8] is an English lexical database. WordNet groups sets of synonym word in so called synsets that represent distinct concepts. These synsets are linked together via semantic relationships like hyponyms (subclass), holonyms (part of), antonyms (opposite meaning), polysemy (words with several senses), etc. WordNet is freely available, and is considered useful for natural language analysis applications. There is also an RDF version available of WordNet[9]. Besides the English variant, variants in other languages exist as well[10].

Getty Thesaurus of Geographic Names

The Getty Thesaurus of Geographic Names (TGN) is a vocabulary and ontology that describes information about places and locations. For example, most cities in the world are in the thesaurus and contains descriptions like coordinates, number of inhabitants, a description, alternative names, etc. Moreover, it also contains hierarchical relationships like province, country, continent, etc. There are two different views on places, a physical and a political one. Politically, places are typed with concepts as capital, tourist center, trade center, river port, etc. Geographically it defines location types like ocean, seas, rivers, waterfalls, etc.

[5] cf. `http://dublincore.org`
[6] cf. `http://www.foaf-project.org`
[7] `http://sioc-project.org/`
[8] cf. `http://wordnet.princeton.edu/`
[9] cf. `http://www.w3.org/2006/03/wn/wn20/`
[10] for an overview of available languages in WordNet structure refer to `http://www.globalwordnet.org/gwa/wordnet_table.htm`

5.4 Data Transformation

5.4.1 Data exchange

In many cases applications are specialistic and have unique data modeling needs that cannot be captured by a common vocabulary. In other cases applications with their own internal models already exist, need to cooperate with other applications and a complete remodeling of the original application is not feasible. Many times the data that needs to be exchanged is user related. Think for instance about hospital applications that want to exchange patient data to get a better understanding and more complete picture of a patient and his disease. Another example is the learning domain, in which students use several Web applications to understand the concepts of a course and in which the teacher wants to gather the information from these application to assess the knowledge of his students, e.g. so that he can adapt the lectures to focus on the student's weak points. Also in general exchanging knowledge is interesting. Think for instance of a television guide application that wants to reuse the information of a movie database so that it can show data about movies on television. Or an application for booking plane tickets that also wants to show information about additional transportation to and from the airport from a public transport application. In other words, many scenarios exist for which exchanging data between applications is beneficial and data interoperability is a necessity.

When a common vocabulary for a group of applications is not feasible the scenario as depicted in Figure 5.3 presents itself. This means that every application needs to create a specific mapping for every other application it wants to share data with. This is more costly than the shared vocabulary approach as it now takes n^2 mapping steps for n applications to share data. However, in most cases this solution will be used to solve a specific data exchange need, which means that n is typically low.

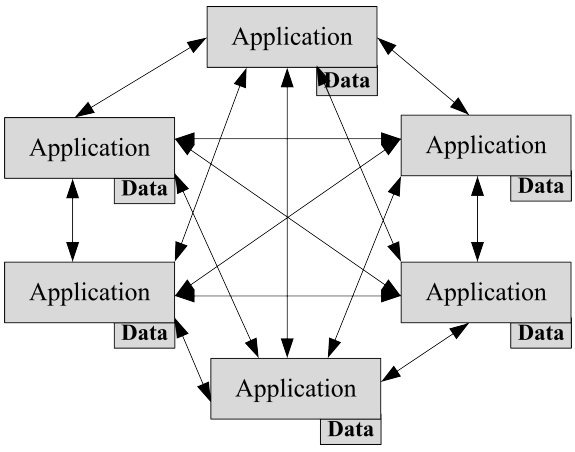

Figure 5.3: Direct connection approach

The actual solution for specific scenarios is not necessarily a choice between either the solution in Figure 5.2 or the one in Figure 5.3, but the solutions can be mixed. A set of applications can find some common ground and develop a shared ontology for generic data, but develop specific mappings for specific data translation between specific applications. This means that for most cases in reality a custom implementation needs to be developed based on the circumstances.

5.4.2 Transformation scenario

In this scenario we look at applications that already have there own data models and for which remodeling is unfeasible. As a data exchange format we choose to use the Semantic Web languages RDF(S) and OWL. OWL allows a high expressivity, while also providing a semantic structure to exploit relationships between concepts. The fact that we have chosen to use such a language does not automatically mean a limitation for the data models of the existing (Web) applications. Already many Semantic Web wrappers are out there[11], and for most data models such a wrapper can be produced rather easily.

Let's look at the following two example OWL schemas between which we want to map the instances.

The first schema is:

```
<owl:Class rdf:ID="User">
<owl:Class rdf:ID="Name"/>
<owl:ObjectProperty rdf:ID="hasName">
    <rdf:type owl:FunctionalProperty"/>
    <rdfs:domain rdf:resource="#User"/>
    <rdfs:range rdf:resource="#Name"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="firstName">
    <rdfs:domain rdf:resource="#Name"/>
    <rdfs:range  rdf:resource="XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="infix">
    <rdfs:domain rdf:resource="#Name"/>
    <rdfs:range  rdf:resource="XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="lastName">
    <rdfs:domain rdf:resource="#Name"/>
    <rdfs:range  rdf:resource="XMLSchema#string"/>
</owl:DatatypeProperty>
```

In this schema a user's name is built up from a first name, a middle name or name prefix and a last name.

Also consider the following schema:

```
<owl:Class rdf:ID="User">
<owl:DatatypeProperty rdf:ID="hasFullName">
    <rdfs:domain rdf:resource="#User"/>
    <rdfs:range  rdf:resource="XMLSchema#string"/>
</owl:DatatypeProperty>
```

We are looking for a language that allows to express and run a mapping between the instances of these schemas. We will now look at several transformation languages that allow us to do that.

5.4.3 Transformation using SWRL

One language that gives us expressive power we need for instance transformation is SWRL [Horrocks et al., 2004b]. SWRL is a rule based proposed extension of OWL and RuleML[12]

[11] e.g. Refer for a collection of wrappers to `http://simile.mit.edu/wiki/RDFizers` .

[12] `http://ruleml.org/`

that allow the expression of Horn-like rules. A SWRL rule consists of an antecedent and consequent, which both are OWL patterns with variables. If the antecedent holds this means that the consequent should also hold. We can use a SWRL engine in practice such that if the pattern expressed in the antecedent exists in the instance data set, the consequent pattern gets generated and added to the data set.

Consider the following abstract SWRL rule:

$$\left.\begin{array}{r}\langle ?user : hasName\ \ \$n\rangle \wedge \\ \langle \$n : firstName\ \ \$fn\rangle \wedge \\ \langle \$n : infix\ \ \$in\rangle \wedge \\ \langle \$n : lastName\ \ \$ln\rangle\end{array}\right\} \Rightarrow\ \langle ?user : hasFullname\ \ concat(\$fn,\text{`}\ \text{'},\$in,\text{`}\ \text{'},\$ln)\rangle$$

This abstract rule expresses that if there is a user who has a firstname, infix and lastname, this implies that the user also has a fullname which is a concatenation of the three elements.

In actual SWRL syntax[13] this is written as[14]:

```
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#mapName"/>
  <ruleml:_body>
    <swrlx:individualPropertyAtom  swrlx:property="hasName">
      <ruleml:var>user</ruleml:var>
      <ruleml:var>n</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom  swrlx:property="firstName">
      <ruleml:var>n</ruleml:var>
      <ruleml:var>fn</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom  swrlx:property="infix">
      <ruleml:var>n</ruleml:var>
      <ruleml:var>in</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom  swrlx:property="lastName">
      <ruleml:var>n</ruleml:var>
      <ruleml:var>ln</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:builtinAtom swrlx:builtin="&swrlb;#stringConcat">
      <ruleml:var>fullName</ruleml:var>
      <ruleml:var>fn</ruleml:var>
      <ruleml:var>in</ruleml:var>
      <ruleml:var>ln</ruleml:var>
    </swrlx:builtinAtom>
    <swrlx:individualPropertyAtom  swrlx:property="hasFullName">
      <ruleml:var>user</ruleml:var>
      <ruleml:var>fullName</ruleml:var>
```

[13] refer to `http://www.w3.org/Submission/SWRL/` for details on the SWRL syntax

[14] Note that the SWRL-code is a bit simplified, as the concatenation of adding spaces between name elements is left out in this example, and the user identification is assumed to be the same in both applications.

```
    </swrlx:individualPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

This SWRL rule checks for the pattern first name, infix and last name in the source data. If this pattern occurs, a new full name is generated that is a concatenation of the three subparts, and an RDF triple is generated that has the user as a subject, has a hasFullName property which was computed as a result of the concatenation operation.

The other way around, i.e. mapping instance data from the second schema to the first is also possible, even though it is a bit more complex. Consider the following abstract SWRL rule:

$$\langle ?user : hasFullname \ \$fulln \rangle \ \} \Rightarrow \begin{cases} tokenize([\$fn, \$in, \$ln], \$fulln, \text{`}\ \text{'}) \wedge \\ \langle ?user : hasName \ \_n \rangle \wedge \\ \langle \_n : firstName \ \$fn \rangle \wedge \\ \langle \_n : infix \ \$in \rangle \wedge \\ \langle \_n : lastName \ \$ln \rangle \end{cases}$$

Here we look at users with the hasFullname property. This full name is tokenized in a first name, infix and lastname, using a space as the subelement separator. Note we used the (simplified) assumption that every name consists of these three subelements and that every subelement do not themselves contain spaces. Then a structure is built that is an instance of the first schema, where _n denotes a newly created blank node for a name composite structure.

Typically in implementations of reasoning engines like SWRL newly computed triples are inserted into the source dataset as derived facts. However, for our purposes we typically want to communicate the mapped data to the target datasource. In our implementation we used the Bossam[15] reasoning engine. Bossam uses the source data and SWRL rules as its input, and is then able to output all deduced facts. These facts are communicated with and stored in the datastorage of the target data source.

5.4.4 Transformation using customized SPARQL queries

Given that the SWRL rule language was readily available and gave us the basic expressivity we needed, it was our first approach for specifying mappings between schemas. However, using this approach also lead to some practical problems, namely:

- **Expressivity** Even though SWRL is quite expressive using its built-in data operators, more complex data operations cannot be performed. Consider our example for tokenizing a full name into a first name, infix, last name, where we do want to consider the possibility of a missing infix and multiple first or last names, based on heuristics. This is not possible in SWRL.

- **Syntax** Even though SWRL is expressive, its syntax is rather verbose and complex to understand. This makes developing these rules more time consuming and hard to maintain.

- **Implementation availability** Several SWRL implementations are available. However, as of this writing there is no SWRL implementation that supports its full

[15] http://projects.semwebcentral.org/projects/bossam/

specification because of undecidability issues. Especially the data operations we are after where not available during our research, e.g. Bossam has partial support for textual and mathematical operators.

To overcome these issues, we constructed a new RDF transformation language based on SPARQL that we call SPARQL Transformation Language (STL).

Language description

SPARQL allows pattern matching of an RDF graph with triple templates, variables and filtering restrictions. It also allows 'CONSTRUCT' queries, meaning that the result of a SPARQL query itself is an RDF graph, based on a RDF template in which the variables have been substituted. Together, this forms a basis to map one RDF structure to another. However, as SPARQL is a query language and not a transformation language, we found that it lacks some functionality that we need for our goals, namely value transformation and aggregate functions.

Therefore we implemented STL such that it extends the SPARQL CONSTRUCT, FROM, WHERE syntax with the TRANSFORM operator. The TRANSFORM keyword is followed by a piece of arbitrary JAVA-code encapsulated by an opening '${' and a closing '}$'. In the TRANSFORM one can refer to variables in the SPARQL WHERE clause by using the '?' sign. These variables will automatically be transformed to strings. If one of the variable values is a URI this must be explicitly cast from String to URI in the TRANSFORM clause. The TRANFORM clause can also generate variables by using the '$' sign that can be referred to in the CONSTRUCT clause. In this way, value transformation can be performed, only limited by the Java programming language.

The following Query constructs a user fullname based on its first name, infix and last name parts.

```
CONSTRUCT {?user app2:hasFullName ?fullname .}
WHERE     {?user app1:hasName ?nameN .
           ?nameN app1:firstName ?fn ;
                  app1:infix ?in ;
                  app1:lastName ?ln . }
TRANSFORM ${ if (?in.equals("")) $fullname = (?fn + " " + ?ln);
             else $fullname (?fn + " " + ?in + " " + ?ln);
          }$
```

This transformation differentiates betweens name that have an infix or not, and prevents the extra space in case no extra infix exists. Note that neither references to variables in the WHERE clause, nor the new variables that can be used in the CONSTRUCT clause need to be specifically initialized.

The following transformation works the other way around and decomposes the full name into its separate components.

```
CONSTRUCT {?user app1:hasName _nameN .
           _nameN app1:firstName ?fn ;
                  app1:infix ?in ;
                  app1:lastName ?ln .}
WHERE     {?user app2:hasFullName ?fullname .}
TRANSFORM ${
            String[] decomp = ?fullname.split(" ");
```

```
    if (decomp.length>0) $fn=decomp[0];
    if (decomp.length>1) $ln=decomp[decomp.length-1];
    if (decomp.length>2) {
        for (int i=1;i<decomp.length-2;i++){
            if (i>1)$in+=" ";
            $in+=decomp[i];
        }
    }
}$
```

In this illustrative example we assumed that if a full name contains no space it is only the first name, if a full name has one space between components it contains a first name and last name, and if a name has more components every component between the first (first name) and the last (last name) is an infix. To cover more complex cases this can of course be easily adapted.

Within the TRANSFORM JAVA code one can use the methods:

- *SparqlQuery*, which allows to query external datasources and use the results within the query.
- *CountSparqlQuery*, is an aggregation function that counts the number of results for some (sub)query
- *MaxSparqlQuery*, is an aggregation function that computes the maximum number of a set of elements of a (sub)query (default value: -maxint, non-numbers are ignored)
- *MinSparqlQuery*, is an aggregation function that computes the minimum number of a set of elements of a (sub)query (default: maxint, non-numbers are ignored)
- *SumSparqlQuery*, is an aggregation function that computes the numerical sum of a set of elements of a (sub)query (default: 0, non-numbers are ignored)

These methods allow to query external data sources, and also provide a rudimentary way for data aggregation.

For external repository queries an optional repository location can be passed to the methods. The default repository is the repository that is queried in the FROM-WHERE clause. But the repository parameter can also be used to indicate an helper dataset. For instance, there is a direct relationship between postal code and a city, but it requires an external datasource to make this relationship explicit. Given that we have such an external datasource we could convert postal codes to city names, and vice versa we could convert a city name to a range of postal codes.

This could lead to the following query:

```
CONSTRUCT {?user app2:livesInCity ?cname .}
WHERE     {?user app1:hasPostalCode ?pcode .}
TRANSFORM ${
    String Postalquery = "SELECT \?cityname
                          WHERE \?postalcode :stringRep ?pcode ;
                                             :relatedTo \?city .
                                             \?city :hasname \?cityname";
    String[][] cityNames = SparqlQuery ("PostalRepos", Postalquery);
    $cname = cityNames[0][0];
}$
```

Note that 'Postalquery' has two types of variables: Variables that have a local scope, with respect to the query, are escaped with a ' \', where variables that refer to a variable in the WHERE clause are not escaped.

Implementation

STL is implemented and available as either a library or a Webservice. The Webservice is configured with a set of repository names and URLs of the SPARQL endpoints of those helper datasets. An incoming request contains a URL of the source SPARQL endpoint, which can be queried for the data that needs to be mapped. It also contains a URL of the target SPARQL endpoint, which will receive the transformed data. The request also contains the transformation query needed.

An incoming query is parsed and separated in three parts, namely the CONSTRUCT part, the FROM-WHERE part and the TRANSFORM part. First a SPARQL SELECT query is built from the FROM-WHERE part of the query. This query is sent to the SPARQL endpoint of the application for which we want to map its data. This query returns a set of variable bindings as result.

Using this set of variable bindings the TRANSFORM is parsed. All variables in the TRANSFORM section that refer to a variable in the WHERE section are substituted by the variable binding from the result set. Moreover, al variable references in the TRANSFORM section that refer to variables that need to be created for the CONSTRUCT part are explicitly initialized. The resulting expression is dynamically compiled and executed using the Apache Commons JCI[16] component. This is repeated for every result in the initial result set. The result is an extended result set, i.e. the result of the first query to the SPARQL endpoint extended with the newly constructed variable bindings by the TRANSFORM section.

The CONSTRUCT section is interpreted next. Every variable in the construct section is substituted by its value in the extended result set. This is repeated for every row in the extended result set. The result of this process is a long CONSTRUCT query without the FROM-WHERE clause. This CONSTRUCT query is then interpreted by an internal Sesame SPARQL endpoint and converted into an RDF graph which contains the resulting translation that can be sent to the target SPARQL endpoint.

The transformation language based on the SPARQL query language is more powerful than its SWRL variant. It can use arbitrary Java code for value transformation. It is also less verbose than SWRL which improves on it syntax and readability. Moreover, it is portable to different implementations and underlying databases as it can be simply adapted to use every available SPARQL endpoint.

5.5 Automating Relation Discovery

Given that we now defined a query transformation language that we can use to construct mappings between a source and target RDF datastore, the next step is to actually find these mappings. As manually going through source and target ontologies and comparing concepts can be quite laborious for mapping designers, we decided to build tool support that helps the designer by finding possible relations between concepts in a source and target ontology. We call these possible relations matches. We were inspired by various ontology matching and mapping systems [Rahm and Bernstein, 2001; Euzenat and Shvaiko,

[16] refer to `http://commons.apache.org/jci/`

2007], which for instance use machine learning strategies [Doan et al., 2002], but also frameworks that use an interesting combinations of techniques like Coma [Algergawy et al., 2011]. However, during our research no usable working software could be found that we could use: either the software was not available at the time or the results of the software were practically unusable. Therefore, we decided to create our own toolset which uses and combines various effective techniques.

We implemented a component, which we call Relco, which offers a mapping-designer tool support for finding matches and constructing STL mappings. Relco uses techniques to find possible matches between an input concept and concepts in a target ontology. Figure 5.4 is an overview of Relco's behavior.

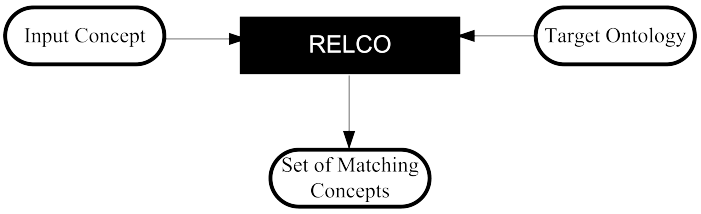

Figure 5.4: Relco information flow

The ambition for Relco is to have a tool that brings together several existing techniques from different fields that allow designers to find matches between concepts (or tags) and target ontologies, which allows the designer to create mappings. Our ambition is not to automatically construct the mappings as the complete background knowledge needed for this is typically not made explicit in RDF ontologies, especially not on the datatype level. For example, the knowledge needed for constructing a query that converts a postal code to a city name is usually not explicitly specified in an application schema describing someone's personal details. Concrete values typically have implicit semantics in applications, most of which is hidden in the application logic. Consider, for example, two learning applications about computer science that keep track of their learners' knowledge of the concept 'concurrency' with a range between 0 and 100. These two applications might assess a knowledge level of 50 of this concept completely different, e.g. for one application it might be interpreted as knowing the basic idea behind concurrency, while the other application might assume that the user has practical knowledge of the use of semaphores, threads and threading pools. In other words, without complete explicit description on the meaning and interpretation of data it is impossible to do automatic (correct) mapping generation. Therefore, we instead chose for a solution in which we try to help the mapping designer by providing matches of concepts between schemas that could be related, but except for the trivial suggestions like value copying, it is up to the designer to make the actual mappings between concept instances.

Relco matches a concept from some source ontology to the concepts in a target ontology. Its result is an ordered list of matching concepts that might be related to the input. The result is ordered based on how good a match is. A concept is a good match if it is semantically related to the input concept. In order to compute matches, Relco compares input and target concepts syntactically, semantically and it uses user input for finetuning and sorting.

5.5.1 String Matching Techniques

The first step Relco takes is finding syntactic matches between concepts in the source and target datasource. This is based on the assumption that concept that are labeled (almost)

equally are more likely to be related than differently labeled concepts. This results in a promising set of target concepts. If we for example start with a concept that is labeled as "lastname", and we have a concept in the target ontology that is also labeled with "lastname", this seems to be a good candidate matching concept because of its syntactical equality.

In order to find syntactical matches between concepts, first the textual representations for our concepts in both the domain and target ontologies have to be determined. As identifying the appropriate textual representation for concepts is ontology-dependent and Relco is designed to be a generic tool, we made it possible for the tool to be configurable for most practical situations.

Usually concepts contain a property like rdfs:label to denote the label for a given concept. Sometimes, there are multiple candidate labels for a concept. For instance, the Simple Knowledge Organization System (SKOS) vocabulary uses the skos:prefLabel and skos:altLabel properties to distinguish preferred labels and alternative labels. Multilabels also occur often in ontological sources that support multilinguality. We also see sources that use a more complex naming schema with intermediate label nodes. Lastly, we also consider labels that are encoded in URIs, usually as part of the fragment identifier. In its configuration Relco supports each of these labeling scenarios via different constructs: for the simple cases only the appropriate properties have to be specified, for the more complex cases a modified SPARQL query can be used, and for URI-encoded labels the delimiter characters have to be specified.

An example configuration could be:

```
Source.0.GraphPattern = {x}rdfs:label{y}
Source.0.ConceptVar = x
Source.0.LabelVar = y
```

The 'Source' keyword indicates we are specifying a source ontology, i.e. which will provide the input concepts that are to be matched to some target ontology. As more than one input source can be specified it is specified that this is the (in this case) 0th source specification. This configuration specifies that every concept 'x' in the ontology is textually represented by label 'y'. This implies that if the string matching process matches the textual representation of concept 'c' with a label 'y', we than consider 'x' as a conceptual match with the concept represented by 'c'.

After having thus obtained the labels for all the concepts in the source and target ontology, we can match the input concepts via their labels. To do that, we first stem both tags and labels[17] to cater for morphological differences between words.

Given real world datasources, one encounters different spellings for the same words, and spelling errors. In English, for example, spelling differences exist between the US and British dictionary. Examples are "aeroplane" versus "airplane", "centre" versus "center", "colour" versus "color" and "moustache" versus "mustache". Also consider composite words, i.e. sometimes words are written together as one word, or separate, for example "family name", versus "family-name" or "familyname". Also, the open Web-like of the Semantic Web implies that misspellings need to be taken into account, i.e. not all ontologies are carefully crafted by groups of domain experts. This is especially important when we use user tags as input instead of a source ontology, see Section 6.1.3.

Therefore, we use an algorithm for computing the similarity between strings. Many similarity metrics are available. In Relco we use the open source Simmetrics library[18] for

[17] using Snowball, cf. http://snowball.tartarus.org/

[18] cf. http://sourceforge.net/projects/simmetrics/

our lexical matching needs, as it contains implementations for many of the well-known string similarity algorithms.

In practice there are three algorithms that we use most often, namely Levenshtein distance [Damerau, 1964], Jaro-Winkler distance [Jaro, 1989] and Soundex [Knuth, 1973]. Levenshtein is a well-known metric for calculating word distances. It calculates the minimum number of edits to transform one word into another (the fewer edits the more alike the words are). Jaro-Winkler is an alternative in which matching characters in two words are compared based on the position of the character in the words. Words that share many similar characters on similar positions in the word are considered more similar than words that do not. The Soundex algorithm is based on the phonetics of words. Words are considered more similar to each other the more they sound alike.

Relco can be configured to specify which matching algorithm to use. The algorithm derives that two concept match if the word distance for their textual representations calculated by the algorithm is equal or greater than a (configurable) threshold. The result of this string matching step is a set of concept candidates that textually relate to the input concept, each candidate with a certainty that reflects the calculated word distance between the labels. The result of this string matching step is then input for the next steps of the matching analysis.

5.5.2 Utilizing Helper Ontologies

The result of Relco's first step is a set of conceptual matches based on syntactic relations between the textual representations of concepts in a source and target ontology. In our next step we want to exploit the semantic structure of the underlying ontologies as well as helper ontologies to extend our set of matches. We base this technique on the observation that an ontology connects semantically related concepts and that such related concepts might be semantically equivalent for the matches we computed after step 1. Semantically related suggestions might for example be convenient if the source and target ontologies define their concepts on different levels of granularity, or use different but semantically equivalent terms.

We can for instance think of exploiting the rdfs:subClassOf property. This property can be used to find more specific concepts, more generic concepts or sibling concepts. Suppose we have a source concept labeled "Java" that syntactically does not match with any concept of the target ontology. The target ontology could still have semantically related terms like 'Object Oriented Programming Language', which is semantically instead of syntactically related. By using a helper ontology about programming language, we could find this match. By following, for example, the rdfs:subClassOf relationship we can now extend this initial result with the semantically related concepts in the target ontology like "programming language" (more general) or "Standard Edition 6" (more specific). We could even consider to follow a longer path in the graph instead of a single property, e.g. by also considering the subclasses of "programming language" we would find "C++" or "Python", i.e. sibling concepts of "Java". If these related concepts can be found in the target ontology they might be good semantic related matches with the concept in the source ontology.

Which property or path of properties will be generally considered in the target or helper ontology must be specified in the configuration for every ontology. In Relco we allow two ways to define which path of properties to consider. One way, for path length 1, is to simply specify the property. Good examples being the often used properties "rdfs:subClassOf" and "skos:related". It is also possible to specify in which direction the property should be inspected, i.e. as they are directed edges on the RDF graph. For example, to find the

more general concept in an ontology that uses the rdfs:subClassOf property we need to inspect it inversely. Sometimes this simple schema is too simple however, i.e. if we want to take into account specific paths of properties (e.g. the sibling example we used earlier). In case of our previous sibling example inspect two subsequent properties (the inverse rdfs:subClassOf and then the regular rdfs:subClassOf). In reality, paths may also be longer and more complex or require additional conditions. In those cases we specify semantic expansion by an explicit query (using SPARQL), for example:

```
PREFIX ws: <http://www.w3.org/2006/03/wn/wn20/schema>
SELECT DISTINCT ?wordForm
WHERE  { ?Synset ws:containsWordSense ?aWordSense .
         ?aWordSense ws:word ?aWord .
         ?aWord ws:lexicalForm {"%inputTerm%"@en-us} .
         ?Synset ws:containsWordSense ?bWordSense .
         ?bWordSense ws:word ?bWord .
         ?bWord ws:lexicalForm ?wordForm .
         FILTER {?wordForm != "%inputTerm%"@en-us}
        }
```

It is a SPARQL query that searches for all the synonyms of a given input term (%inputTerm%) by using the WordNet lexicon as a helper ontology. The query defines the search for the Synset that contains the input term from our source ontology. It then returns all wordforms of that 'Synset' (i.e. the synonyms), except for the original input term. The %inputTerm% variable is not a SPARQL construct, but something we introduced to denote the substitution of the lexical forms of the output of the step we detailed in Section 5.5.1. Relco extracts the results, denoted by the bounded variables from the SELECT clause, as semantic matches for the input terms.

The second, semantic expansion step thus moves from a set of concept matches to an extended set of concept matches (each with a certainty value and a set of associated concepts). This certainty value of the extended matches is lower than the original matches. How much lower is configurable by a decreasing factor based on how relevant a specific property path is judged to be semantically relevant. For our "Java" example the designer could for instance specify a decrease value of 0.2 for the sibling query, meaning that if the concept for "Java" was originally matched in Section 5.5.1 with a certainty of 0.9, the certainty for sibling concept "Python" would be (1-0.2)*0.9=0.72.

5.5.3 Using Context for Disambiguation

In the third step in Relco we go for context disambiguation. There we exploit the structure of both ontologies in Relco. The underlying assumption is that for a certain concept in the ontology, the surrounding concepts in the graph can provide insights in what concepts mean. The idea is, as illustrated by Figure 5.5, that if two or more concepts of a source ontology have a match in the same neighborhood of a target ontology (defined by a maximum distance in the length of the property path between concepts) then the chance is higher that the matches are better. For example, suppose that in our example source ontology there is a property "skos:related" between "java" and "programming". This allows us to consider matches for "java" like "Python" and "SE 6" more relevant than matches like "island" or "coffee", i.e. because we know that "Python" and "Se 6" are also related to the "programming" concept and "island" and "coffee" are not.

We use the path length between the input concept and other input concepts in the source ontology as a neighborhood indication. In our example, we have a path length of

2 between java and c++ (both have a connection with 'programming') and therefor we increase the certainty of "programming" being a good matching concept for "java".

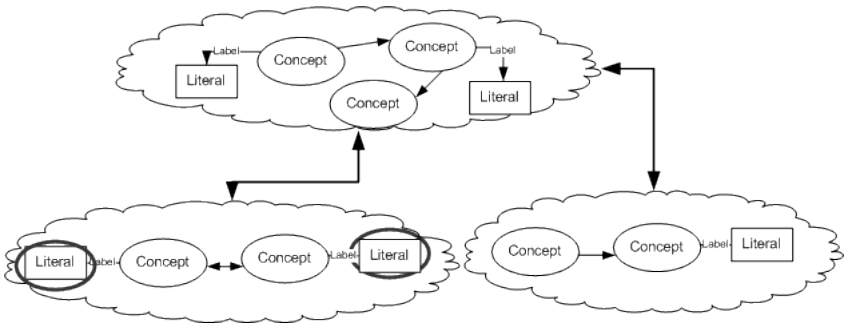

Figure 5.5: Two matches in one region of an ontology

Determining the path length between all concepts that are matched with a set of input tags is very costly if this is determined during runtime only. To do this efficiently we pre-compute the neighborhood of every concept in the target ontology once, i.e. we compute for every concept all the concepts that are within the maximum path distance. The path distance is the minimum number of properties that connect two concepts. The chosen maximum for path distance is typically rather small, especially for well-connected graphs as otherwise almost any concept in an ontology will be in the neighborhood of every other concept, but a good value depends on the connectedness of the ontology.

The certainty-increase for tag suggestions that are found in the same region (in function of the path length) is configurable, as is the maximum path length. We could for instance configure to decrease uncertainty with a maximum factor of 0.8. If the matching concepts have distance 1, as with our "java" and "programming" example, where "C++" has a path distance of 2 from "java" and "programming" has an initial certainty of 0.7, then the new certainty for "programming" would be ((1-0.7)*(0.8/2))+0.7=0.82. Initial matches "island" (e.g. also initial certainty of 0.7) and "coffee" in this example would not get improved certainties, as they are not in the same neighborhood as other input concepts.

How good the chosen value is for influencing the certainty value needs to be evaluated and depends on the structure and language of the target ontology.

5.5.4 Using User Feedback

After the first three steps we have for an input concept obtained a set of matching concept-certainty value pairs. We can now consider what how to use these matches.

One scenario is to now automatically use the set of matches and assume relationships between the input concept of the source ontology and matching concepts in the target ontology, provided that the certainty is above a threshold, and generate mappings on that basis. However, in most cases things are not that simple. Usually the underlying datatypes of the RDF literals differ and need to mapped. For example, different datasets might store distances in different units and granularity, e.g. miles, kilometers, meter, etc. So ontologies that define the size of objects can not be automatically mapped to each other without regarding this possible type mismatch. Another example is that datasets can have relative views, e.g. of user models. If two applications both model the knowledge of a user of the same particular subject, that doesn't mean they can have very different views on it. In one application one is labeled as a Linux-expert if one can use the command prompt without problems, while another application might only label programmers that contribute to the

Linux kernel to be Linux-experts. These granularity issues and viewpoint differences can hardly be solved by automatic mapping mechanisms, as the information needed for this is usually not part of the ontology and depends on the mapping context and designer.

To reduce this problem we involve users in the mapping process as well and we try to make their mental view on the mapping explicit by validating our matches. We present the user with our suggested matches and use their feedback to improve the matching certainties.

To present these concept matches to the user in a user interface, we first need to choose a good textual representation for the concepts, especially in the case a concept has more then one textual representation. The issue is illustrated by Figure 5.6, where we see a concept that has several textual representations.

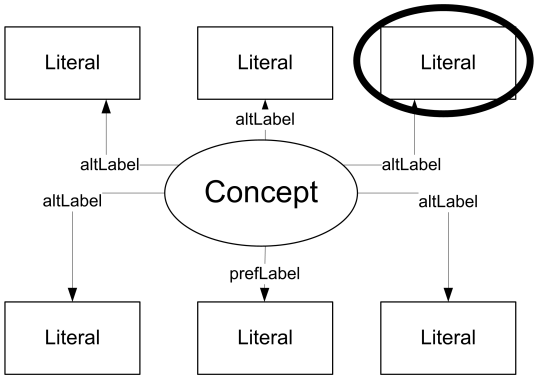

Figure 5.6: Example of concept with multiple labels

Suppose we have a concept with several labels where we can discern between labels, e.g. preferred labels and alternative labels. If the input of the source ontology matches with an alternative label we might want to show the preferred label instead of the alternative one. For instance suppose we have a concept with a preferred label “application” and an alternative label “program”. Now if the textual representation of an input concept matches with “program”, we might want to give the preferred label “application” anyway, instead of the actual label “software” that lead to the match.

Relco provides three configuration options for choosing the best textual representation of a concept for an ontology. One can choose to define a preferred label and present that to the user. Another possibility is to present only the labels that were matched during the string matching process in step 1. The third option is presenting the user all string representations of a concept.

The user can now be presented with the best matching concepts for his input concept. The former processes for finding concept matches for tags were user-independent. By utilizing user feedback we want to improve the process of computing concept matches by storing their choices and use that information to adapt the certainty values.

Relco has a feedback channel for both simple and extensive feedback input. In the simple case, the user is presented with textual representation of matches and is asked if one of them is a good alternative description for the input concept. If the user does select one of the concept matches we consider this as implicit feedback that indicates that the selected concept is a good match to the input concept. We then store this information and use this information to increase the certainty of the chosen match, the next time the same input is encountered. We also provide a more explicit feedback channel by allowing the user to explicitly evaluate concept matches for the given input concepts.

We store the feedback using an RDF data format. Figure 5.7 represents the (simplified)

schema for the feedback instances that we store in our datastore. We store the input concept and the concept match that we computed. For every match we compute for an inputtag we maintain the total number of users that gave a positive or negative rating for that match. This results in a certainty which we store (such that we don't have to recompute). If the user is logged in, we record the individual judgements and with which users they belong.

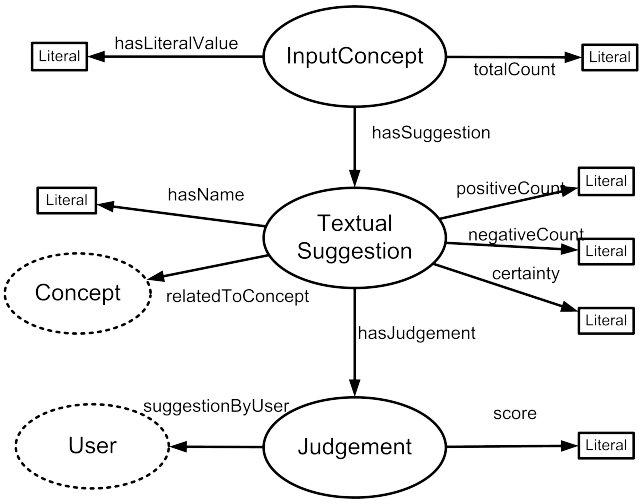

Figure 5.7: Feedback Data Model

The feedback information that we keep in the store is in the first place used to change the certainty value, and make it more fitting with the user's opinion. The first thing Relco checks (before the start of string matching) is to see if the concept is already in the feedback store. If so, we adapt the certainty of matches based on previous input of the user.

Consider for instance the input concept "programming language". Concept matches for programming language might include "java", "c++" and "python". From previous user feedback we might have recorded that "java" is chosen much more often as final concept for the textual representation "programming language" than the other programming languages (e.g. caused by the relative popularity of Java). The certainty of the "java" concept is then promoted based on this user behavior as we find it more likely that "java" will be again be chosen instead of the other alternative concepts. We use both personal information for users and general aggregated information from all users. For example, if Java is most popular it will increase the certainty of Java for programming language for all users. However, if a specific user repeatedly chooses C++ this will lead to a higher certainty for that specific user for C++.

5.5.5 Implementation and Performance

During implementation of Relco we found that a straightforward use of one of these similarity metrics algorithms has one big drawback: it does not scale well. Every concept in the source ontology has to be compared with every label in the ontology, and every comparison operation is relatively expensive. Tests on a reasonably large representative target ontology with about 3800 concepts showed us that it would take about 7 seconds on standard hardware to match one input concept of some source ontology with the string representations of those concepts. To avoid this and improve on this performance, Relco

Input Label	Default Performance		Lucene Performance	
	#results	Performance(s)	#results	Performance(s)
Restaureer	9	7.344	9	0.688
Platteland	8	7.187	8	0.672
Emancipatie	13	7.766	11	0.688
Amsterdam	13	8.150	0	0.516
Zang	25	5.969	24	0.657
Koningshuis	6	7.453	4	0.676

Table 5.1: Relco Peformance Using the GTAA ontology

uses Apache Lucene[19] to build a 'fuzzy' index. The principle behind this index is to compute n-grams of words, meaning that strings are broken up into all subsequences of length n. These subsequences are put in an inverted index. Using the heuristic that two only slightly differing strings share many common subsequences, the n-grams of an input term can be quickly compared with the labels in the ontology. The result of this process is a set of matching labels with a similarity measure that represents the number of matching n-grams between the input tag and the matching label. However, experiments showed that the accuracy of this similarity measure is in general worse than that of the previously discussed algorithms. Therefore, in our case Lucene is used with a relatively low accuracy setting in order to quickly pre-select all candidate strings that might match; then the string similarity algorithms from the Simmetrics library are used for higher-quality string comparisons. With this method using the same test ontology with 3800 concepts we were able to reduce computation time to less than a second. Table 5.1 illustrates these performance differences for the Dutch GTAA ontology (see Section 6.3.1) and some arbitrary input textual labels we produced just for testing.

Another possible performance bottleneck in Relco was during the disambiguation process. The disambiguation process means that Relco has to compute the relation of all neighboring concepts of the input concept and then compute the distance between the target concepts. To speed up this last process we built an initial $|C|x|C|$ distance matrix. We then store that part of the matrix with distances smaller or equal to a constant `n`. This `n` represent the longest path between that may exist between concepts that are still in each other's neighborhood. This `n` is typically rather small, especially for well-connected graphs, but a good value depends on the structure of the ontology. So, when the input concept and its neighboring concepts have target concepts in the same neighborhood this is immediately retrievable from our matrix.

Relco is implemented in the Java programming language. It depends on the Sesame RDF store library, the Simmetrics library (for string matching purposes) and as explained in the previous paragraph: the Apache Lucene library. Relco can be deployed in two ways. It can be deployed as a library. In this mode external applications can easily embed Relco. This has the disadvantage that Relco has to initialize its datastores for every concept matching process, which will give it a slight but perceivable slowdown. Relco can also be deployed as a Web service. It then has to be deployed in a Java servlet container like Apache Tomcat and can then be called by SOAP messages. Relco then only has to be initialized once to be ready for operation. Its setup is slightly more complex, but it is faster (especially when using the in-memory store) and can be used by more Web applications at once.

Figure 5.8 is a simplified class diagram of Relco. The class RelcoProperties is used

[19] `http://lucene.apache.org/`

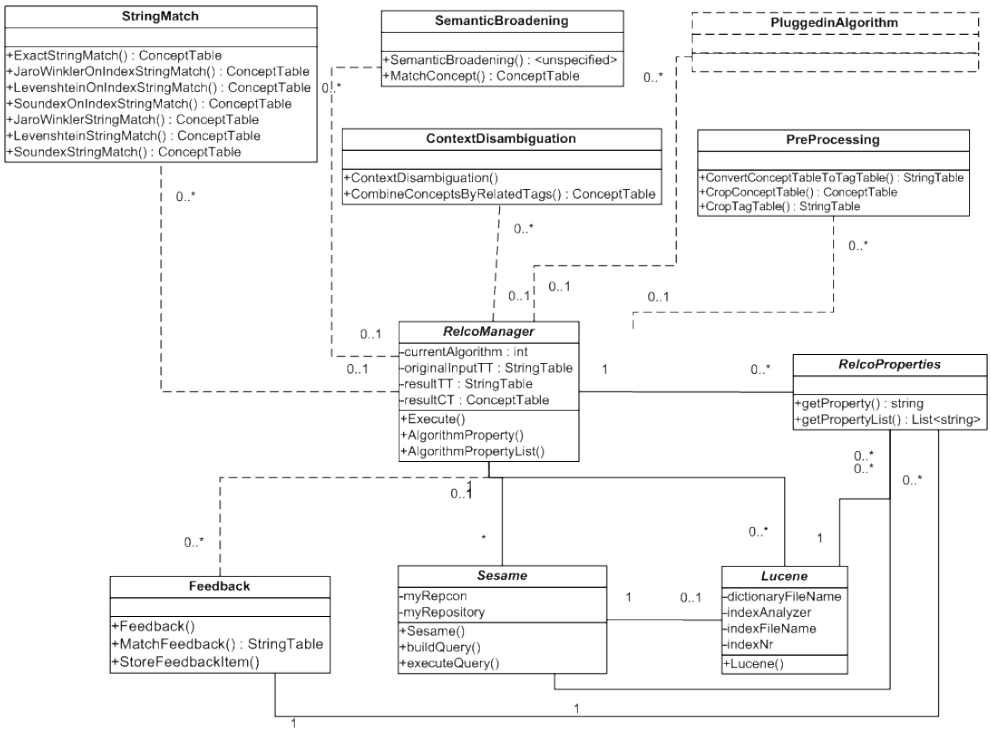

Figure 5.8: Simplified Relco Class diagram

by other classes to retrieve configuration elements. It can return individual properties or a list of properties, for example all source files for a specific repository. The properties are read from the configuration file. The classes Lucene and Sesame support the storage of information. In this case Sesame stores ontologies in repositories and Lucene creates indexes of labels, their related concepts and n-grams of the labels. An index is always based on a repository. The class RelcoManager contains the method Execute(). This method instantiates the models, invokes the algorithms and keeps track of the intermediate tag and concept suggestions and eventually returns the suggestions. A StringTable and ConceptTable object are instantiated to store the intermediate results and are updated after the execution of each step in the process.

5.6 Summary

In this chapter we investigated various aspects of data integration for RDF data sources.

We investigated using different integration approaches. One approach was data coupling, by connecting concepts in different sources with a relationship, e.g. owl:sameAs for identical concepts. This is a simple approach, but not always usable, for example queries over coupled datasources become more complex because they have to incorporate all possible schema structure for their answer.

Another approach is to reach a common ground by defining a standard vocabulary that all applications decide to use, or that is used as an exchange format. This solution has some favorable characteristics as new applications can be easily connected and exchange information with a set of other applications. However, this approach depends on the possibility that a common vocabulary can be agreed on that incorporates all information that is to be exchanged between all parties.

As in many cases this is not feasible we also considered the scenario in which every

pair of applications develop pair-wise mapping for their data. This scenario is necessary for applications that are either very specific and only need to exchange that data with few other applications or are in a domain in which there is no common ground.

Both the case of an intermediate ontology and the application-to-application scenario require mapping data from one schema to another. This requires a formalism for expressing the mapping of instance data between schemas. We investigated two methods for expressing these mappings. One mapping is based on the SWRL rule language. It is based on rule antecedents and consequents, i.e. when some condition holds 'new facts' are derived. These 'new facts' is the actual translated instance data which in our implementation is transported to the target datastore. Because of expressivity limitations in the SWRL rule language as well as readability and verbosity issues, we decided to create another solution based on the SPARQL language. SPARQL was in our implementation extended with a 'TRANSFORM' operator, which provides additional value transformations as well as some additional aggregation operators and operators to use background information. The resulting extension, called STL, is more expressive, versatile and expressive than the SWRL rule language.

We also introduced the Relco framework which can be used to automate relation discovery, which is used as basis for mappings. Relco was designed to find candidate relationships between an input concept and a target ontology. We used an approach based on string matching, the utilization of helper ontologies, context disambiguation and user feedback for finding a set of candidate concepts and a certainty value that expresses the probability that the candidate concept is a good candidate for the input concept. The Relco approach for relating concepts is inspired by work on ontology matching. It follows the frequently used method in ontology matching [Aleksovski et al., 2006; Rahm and Bernstein, 2001] of finding a set of concepts with the highest syntactic and semantic certainty between the input and the target ontologies. However, Relco is targeted on more flexible situations that are slightly different than with those systems, e.g. in Section 6.1.3 we deploy Relco for relating tags to concepts. Therefore, we focused on extending regular techniques with techniques that exploit the data structures that we use and the users of the system.

Chapter 6

Applications of Data Integration

We will now look at a number of Web applications that have data integration requirements. Data integration in these Web application was based on the techniques that were described in the previous chapter. Every application has unique characteristics and challenges that asked for specific attention. Every application also has unique opportunities in terms of capabilities derived from data integration, which we will explore. In this chapter we will look at three applications that (next to data integration issues) share that they disclose multimedia datasets. However, the applications are in very different domains and have very different purposes. First we will look at χ Explorer, a Web application that was built for navigating cultural heritage collections. Its purpose is to disclose cultural heritage collections to the general public, while at the same time providing valuable metadata for professional archivists. Next we look at SenSee, a television recommender system that exploits semantic relations. Important for SenSee was providing efficient access to large RDF datasets and adapting to a multi-device scenario. The third application we discuss is ViTa, a video tagging application for tagging and searching educational videos via user tags.

6.1 χ Explorer

We first look at the Web application called χ Explorer, which we built for Regionaal Historisch Centrum Eindhoven (RHCe). RHCe is appointed by the region of Eindhoven to govern all official city archives and the cultural historic information related to Eindhoven and its surrounding municipalities. The archives contain millions of objects, including official documents, pictures, videos, maps, newspapers, etc. These objects are organized in collections based on their function and type.

One of RHCe's missions is to disclose their collections to the general public. The current practice is to physically retrieve the objects for interested visitors, based on handcrafted metadata indices constructed by RHCe's professional annotators. This is time-consuming for visitors and RHCe staff. Many objects are also fragile and physical contact could be damaging. So, RHCe has started a process of creating high-quality digital copies of their source data as basis for information disclosure. They also envisioned Web disclosure of their archives to reach a larger public and thus get a better idea of the evolving interests and wishes of their visitors/users. An important element in this process is metadata: RHCe employs professionals to correctly annotate objects. However, the amount of new

objects grows much faster than the annotators can keep up with, so the huge current archives are only annotated for a small part at the moment. If one could get the users to help in annotation, this burden could be greatly lessened.

In Section 6.1.1 we describe an overall metadata structure that intelligently defines a semantically linked heterogeneous dataset. Based on this dataset we have been able to offer a navigation structure over the archive objects, and study how to exploit the metadata in the navigation, e.g. through presenting specialized user interfaces for various facets of the data (Section 6.1.2). Since the involvement of the user groups was an important goal in this process of obtaining structured metadata, we describe in Section 6.1.3 how we used Relco (as described in chapter 5.5) to convert unstructured user-generated metadata to well-structured ontology-based metadata ready for the professional annotators. Thus we illustrate how intelligent techniques for metadata integration and user-participation can be applied in the practice of small/medium cultural heritage institutions. These techniques allow to reduce overhead costs of the metadata creation process and allow for the continuous integration and expansion of collections.

6.1.1 Semantic Integration

A first challenge was integration. Previously, all metadata information was stored in separate proprietary database applications. This had legacy reasons, e.g. mergers of historic institutes, but RHCe also consciously chose for having separate applications for different types of objects because of their different metadata. Photos for example typically have properties like camera type and color capabilities, while birth certificates have properties like person name and birth date. Most of these database applications are closed in the sense that the data is only accessible via an application form or in some cases an XML or SQL export.

The desire was to have one web application that allows the user to browse the RHCe datasets as if it was one integrated homogeneous dataset. We therefore first needed to create one integrated data model that describes all metadata fields of all metadata databases. We were especially keen on identifying those metadata fields that the separate collections have in common as RHCe saw great value in connecting objects in different collections. For example consider the scenario that someone is interested in a marriage certificate (e.g. of his grandparents) and then wants to see photos of the church of the marriage from the time of the marriage.

It is important to point out that considerations like these were inspired on the added value that the new interface could provide. That is also why we have presented these aspects as part of the complete study. An advantage of the integrated dataset approach is that it would allow us to create specialized visualization primitives for navigation (see Section 6.1.2). Furthermore, it also allows us to use it as a format for user-generated metadata (see Section 6.1.3).

In Section 6.1.2 we propose specialized faceted browsing visualizations. In order to make this possible we utilize the structure of a specialized ontology for each of them: a location ontology, a domain ontology and a time ontology. These ontologies are also important in the user-generated metadata phase (Section 6.1.3). There we exploit the labels and semantic structure of these ontologies.

When we constructed the overall metadata structure we chose RDF as its formalism (see Section 2.2.3). Basic requirements were that it should be easily extendible and that it should be possible to easily integrate or connect new data sources (more legacy sources from mergers, but also connecting to data sources from other institutions). As we planned

the navigation primitives for χ we also concluded we required a connection with external knowledge sources. RDF fits all these requirements. Before making the translation from the various databases to RDF we first designed a schema for our combined dataset. We needed to combine here the given structure of the available sources (a heterogenous set of relational databases) and the desired properties of the solution (a homogeneous integrated dataset that is easily navigated).

We wanted to base our schema on open standards (i.e. vocabularies) where possible, as this improves extendibility and reuse (as chances are higher that the standards are used in other sources as well). When we created our schema we didn't find a single open metadata standard that allows describing all cultural objects in RHCe's collection. We did find some standards that focused on art objects, however these were not suitable for the larger part of the collection. We did however identify a Dutch standard called GABOS for describing topographic historic information (developed by VTHA, a collective of many local archives and local cultural centers in the Netherlands[1]), which we could use as a basis. GABOS however only provided a flat list of 'descriptors' with proper names to describe objects. We mapped these content descriptor structure to RDF.

Both the location and domain ontology are based on data structures developed internally in RHCe. The location ontology relies on a hierarchy of locations and their coordinates (based on information provided by the city administration). Locations in the location ontology have also been extended with time properties were appropriate, that can be used in the historical perspective to support evolution in locations, e.g. with city restructurings. The domain ontology was based on a classification hierarchy developed for structured annotation in the various RHCe database applications. In both ontologies we reused standardized relationships where possible, e.g. for hierarchical relationships the "broader" and "narrower" relationships from the SKOS vocabulary [Miles and Bechhofer, 2009].

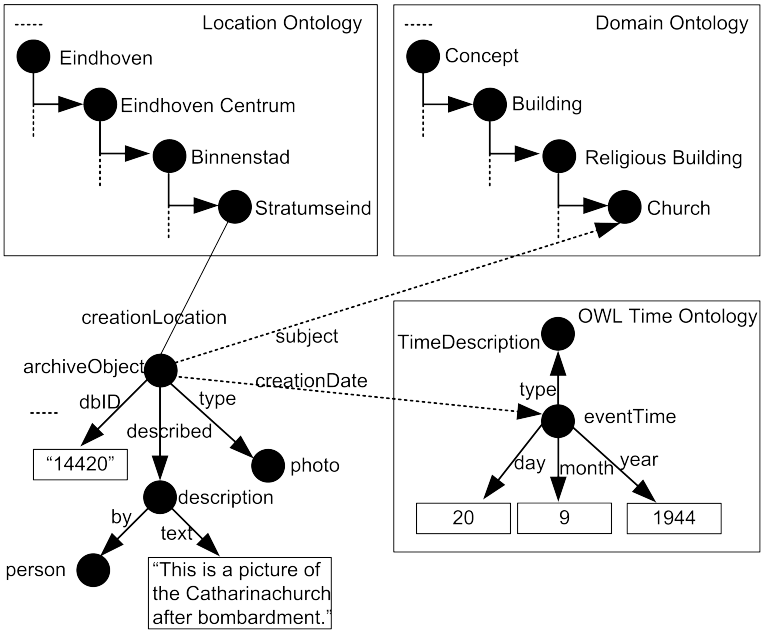

Figure 6.1: χ metadata structure

For the time dimension RHCe didn't have a structured ontology yet, even though different time notions are often used in their metadata databases. As time is an important notion in cultural heritage applications we standardized its notation so that we can access

[1] http://www.vtha.nl/

homogenously time instances later on. We reused the OWL Time ontology and mapped all time notations used in the metadata databases to instances of that ontology.

The next step was to make wrappers that can translate the metadata information from the database applications in RHCe (that remain in use) into instances of our RDF schema. We created unique URIs for objects based on their database-ID and translated all metadata in the databases into terms of our metadata structure with relations to the relevant ontologies. Mapping the actual data items to instances of our schema was a relatively straightforward process, besides some restructuring issues. The time notations could be easily translated to OWL Time notations. The link to the domain ontology was also rather straightforward as long as the annotators correctly used the classification hierarchy. We only considered the classifications that exactly matched the classification hierarchy. For the location hierarchy we had to do some more elaborate matching and mapping, for which we used the Relco framework that was described in Section 5.5.

Figure 6.1 gives an example of an instance (simplified) in our metadata structure. The data describes the picture in Figure 6.2 of the Catharina church after its bombardment on September 19, 1944. Note the links with the ontologies that we created.

Figure 6.2: Example picture from the RHCe dataset

6.1.2 Navigation and Visualization

Based on the integrated dataset we built the user interface of χ Explorer. We focused on navigation and search.

The RHCe user base is typically interested in objects related to specific areas (Section 6.1.4 describes the user base in more detail). Consider our previous example in which a user is interested in the location of his (grand)parents' marriage. Therefore we added a Map view, which should simplify navigating objects over their spatial relationships. We used the Google Maps API (cf. http://code.google.com/apis/maps/) to visualize locations of objects. See Figure 6.3 for a screenshot from the χ Explorer Map view. Here we exploited the transformation of location annotations into links to our location ontology in our integrated dataset. This ontology contains location coordinates which are translated to Google Maps coordinates.

A search or browsing action can lead to the need to visualize a large set of objects on a relatively small part of the map (most are within the city of Eindhoven). Our user

study (see Section 6.1.4) showed that users struggled with large unordered set of objects. Therefore, we incorporated a clustering algorithm which is based on the Hierarchical Agglomerative Clustering Algorithm [Jain et al., 1999]. However, instead of just finding a clustering for a group of points, we reduce the number of visualization points by clustering until we have reached the maximum. The algorithm is as follows:

Figure 6.3: Map view χ Explorer

1. Define the maximum number of clusters that will be visualized on the map.

2. Compute the proximity matrix containing the distance between each pair of search results. Treat each search result as a cluster.

3. Find the closest pair of clusters using the proximity matrix. Merge these two clusters into one cluster. Update the proximity matrix to reflect this merge operation.

4. Stop when the number of clusters equals the maximum.

The clustered sets of objects are afterwards visualized in the Map view. When the user selects a cluster the set of objects associated with that location is visualized. For every object a thumbnail representation is shown together with a description. Within a cluster of objects, the objects can optionally be grouped by creating subsets that have similar properties in one of the other dimensions (i.e. time and domain concepts). In this way, objects in an object set can be grouped either on the time periods or on the concepts for a certain location, which makes browsing an object set more orderly.

Similarly, the user base is typically interested in objects from a specific time period. Consider for example the scenario of a user who is interested in objects from the time his (grand)parents married. Therefore we added a Timeline view in which objects can be related by the time periods in which they have been created. See Figure 6.4 for a

screenshot of the timeline. We used the Simile Timeline component[2] for that. Here we exploited the fact that all date notations in our integrated dataset were translated into OWL time instances. Thus we could generate dates and intervals as elements on the timeline. Dates at different granularity levels can be recognized in the view by the different sizes of the elements.

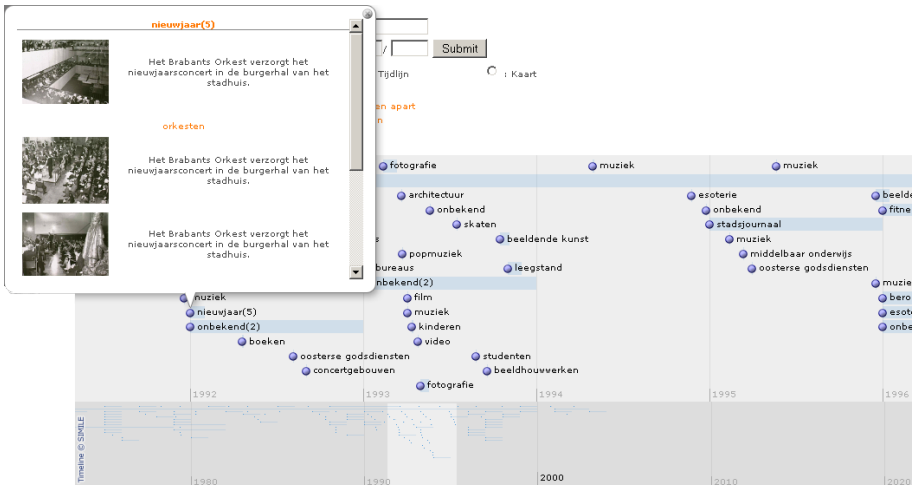

Figure 6.4: Timeline view χ Explorer

For objects on the timeline a similar problem of cluttering the timeline with too many objects had to be faced (as with locations). Objects on the timeline with overlap in the time period are arranged vertically. We had to make sure that the number of vertically placed events fits the screen size. For this we use a slightly adapted version of the clustering algorithm:

1. Determine the candidate timeframes with more results than the maximum.
2. Create a proximity matrix containing the time-distance between each pair of search results in a candidate timeframe. Treat each search result as a cluster.
   a. The distance between timeframes is measured by adding the amount of time necessary to add to both timeframes to come to a union of those timeframes. For instance, given timeframes “1940-1945" and “May 5, 1943", the union of those timeframes is “1940-1945" and 0 time needs to be added to the first timeframe and 5 years minus a day to the second: the total distance between the two is 5 years minus a day.
3. Find the closest pair of clusters using the proximity matrix. Merge these two clusters into one. Update the matrix to reflect this merge.
4. Stop when the number of clusters equals the maximum.
5. Repeat steps 3-5 for each candidate timeframe.

Like in the map facet, a visualization of the objects in an object set can be grouped in the other dimensions (location and domain concepts).

[2] http://code.google.com/p/simile-widgets/

The last visualization we considered was based on the desire of many users to quickly navigate objects that relate to a specific theme or concept. In our example, the user could be interested in objects that relate to the concept of church or marriage. The domain ontology was in fact specifically designed to facilitate this thematic classification of objects, which allows the user to navigate semantically related objects. To visualize the domain ontology we used graph-based visualization, based on the GraphViz graph library[3]. Figure 6.5 is a screenshot of the graph-view in χ Explorer.

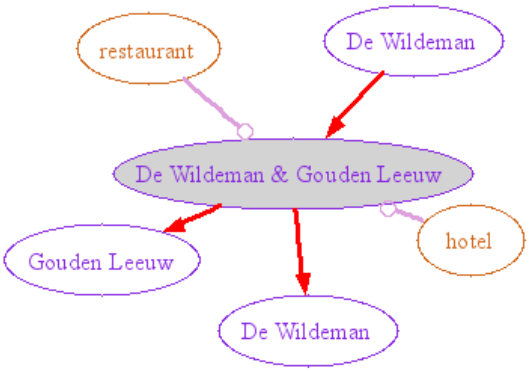

Figure 6.5: Graph view χ Explorer

We visualize the domain ontology, and use that our integrated dataset links data objects to the concept in this ontology. To keep the Graph view simple, the user is shown one central concept and all its related concepts. Different types of relationships have different color codings, as can be seen for the example in Figure 6.5. The central concept there is labeled with "De Wildeman & Gouden Leeuw". This is the name of a business that both contains a "restaurant" and a "hotel", both denoted by the "omvat" relationship ('omvat' is the Dutch word for 'contains'). The other visual relationship is "wasVroeger", which is Dutch for 'was formerly'. This relationship illustrates that this business was formerly called "Gouden Leeuw" and "De Wildeman", i.e. the business merged and retained both names at first. However, eventually the business was again renamed to just "De Wildeman", which is denoted by the top "wasVroeger" relationship.

If a user selects one of the related concepts, that concept becomes the central concept and its related concepts are then shown in the view. The user can also view the set of objects that are annotated with the current concept. Like with the other visualizations, objects can be further grouped in the other dimensions as well.

The χ Framework was built to facilitate the user interface we just described. Figure 6.6 contains an overview of the framework. At the bottom there are the RHCe back ends (the proprietary databases) and the wrapper that transforms that data to instances of our integrated structure. The χ Engine is the application back end that allows storing and querying the data. Furthermore, it maintains a user model where user information can be stored as well as user-generated input (see Section 6.1.3). Querying and storing RDF data in the χ Engine is based on Sesame[4]. The χ Engine also uses Relco (as discussed in 5.5). Relco is used to provides suggestions for mappings between user tags and concepts in the ontologies. The top part of the Figure contains the presentation logic, χ Explorer. χ Explorer provides the specialized user interface (we described earlier) over the data from χ Engine.

[3] `http://www.graphviz.org/`
[4] `http://www.openrdf.org/`

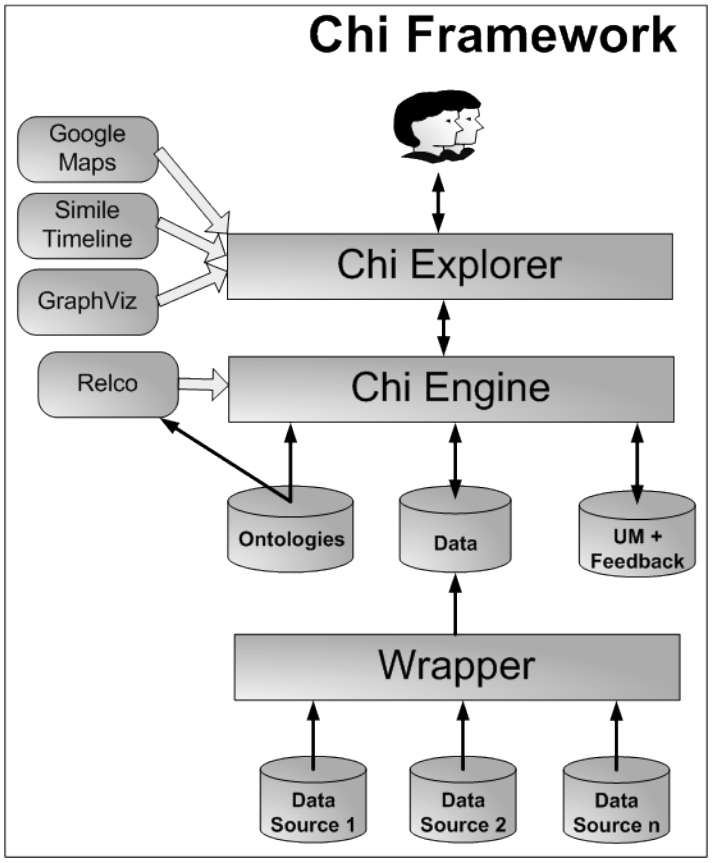

Figure 6.6: χ Framework Architecture

6.1.3 User-generated Metadata

The faceted navigation depends on the relations between our integrated dataset and the ontologies. Given the lack of metadata for many objects we use the Relco component to allow user-generated metadata, including links to the ontologies, which makes the navigation via our visualizations possible.

RHCe has obtained with χ Explorer a web-based application that allows its visitors to efficiently browse the archives based on structured annotations. However, the center has currently only rich annotations available for a relatively small part of its continuously growing archive. Therefore, as part of their new process for creating the metadata, it was decided to attempt using the knowledge and involvement of the visitors to augment the level of well-structured annotations.

Connecting Tags to Concepts

As is typical with this kind of user-participation, RHCe has to cope with the fact that the targeted user group is not used to do professional annotation nor has explicit knowledge of the given annotation data structures. To accommodate lay users, the annotation process is kept simple by extending the system with a tag-based interface [Mika, 2005; Choy and Lui, 2006; Specia and Motta, 2007]. Users can simply enter keywords or short strings of text to give their description of the content, location or date of an object. To warrant the quality of the metadata in the system, it was decided to maintain the current data structures based on RHCe's internal ontologies as the basis for navigating the archives, also because of the known difficulties reported for using a tag-only approach [Choy and

Lui, 2006]. Therefore we use Relco for relating user tags to the concepts in the ontologies.

We had to extend the original Relco with additional input capabilities for using it to relate tags to concepts instead of concepts to concepts. Next to input concepts of an ontology (refer to Figure 5.4), Relco now also accepts input tags. The input for the component consists of an input tag and a set of contextual tags. In the χ Framework these contextual tags are already existing tags for the object that is tagged - which is possibly an empty set for newly tagged objects. For the rest, the Relco process is exactly the same as described in Section 5.5.

χ Explorer uses Relco in two ways. First, when users enter a tag they get a list of tag suggestions, based on string representation of concepts in the used ontologies. If the users select one of the tag suggestions by the system this is not only stored as a tag, but also a relationship with the concerning concept in the ontology. This relationship between concept and object is weighted. If the relationship is added by one user this weight is low. The more people recommend this relationship the higher this weight gets. Maximum weight is assigned by assessment by RHCe professionals. This weight is used for ordering objects during ontology based navigation and search.

Secondly, when the RHCe professionals visit an object page, they will see the user tags and the set of recommended concepts based on those tags. This is used as a tool to assist the professional annotators.

Figure 6.7: χ Explorer Feedback GUI

Figure 6.7 shows the χ Explorer front-end user interface for the Relco feedback mechanism (refer to Section 5.5.4). Both users and professionals can assess the quality of concept recommendations of tags. For every tag suggestion they can indicate if the suggestion is good, neutral or bad. These indications are fed back into Relco for improving future tag suggestions.

Tag quality assessment

Important in a use case like RHCe's is quality control. Besides preserving the objects themselves as good as possible, they have to guard the quality of the metadata describing the objects (and used to enable the wide access). This is the main reason for professional expert annotators. Annotation by end-users (tagging) is a great additional opportunity

to easily gather metadata and especially offers the possibility that specific information is known within the laymen user group that the professional experts do not know (yet). However, obtaining metadata from outsiders is also a vulnerability: despite all good intentions the provided information might be sub-par, not relevant content wise, personal bookmarks or even spam. RHCe considers ways to ensure the quality of user annotations while at the same time not increasing the overhead and costs for the professional annotators.

Figure 6.8: Rating Form

In χ Explorer the users can either reinforce an already existing tag or vote for or against tags of other users (screenshot figure 6.8), to express whether they think that a tag accurately describes the content of an object. The assumption is obviously that if more people favor a tag for an object, it is more likely to be correct. Additionally, χ aggregates tagging and rating activity and presents this data to the RHCe administrators. They get a ranked list of objects, ranked by the number of tagging and rating activity. In this way they can quickly assess which objects are popular and which metadata is debated. This allows professional metadata experts to provide these objects with the best annotations.

As a measure against abuse of the Chi Explorer facilities there is a report button for all user-generated data that allows other users to report abuse quickly: this allows the RHCe administrators to quickly remove abusive metadata entries and ban the respective users if necessary.

User Reputation

Votes for tags are also used to compute the reputation of users. A vote for a tag is interpreted as an indirect vote for the user that suggested that tag. A positive or negative vote will increase or decrease the user reputation weighted by the voter's reputation. So a user's reputation gets higher the more users with high reputation (indirectly) vote for that user.

The users' reputation value is also used during the tag quality assessment, as votes are weighed by the user reputation. In a next version of the system it is planned to assign additional rights to users with high reputation values. These trusted "power users" could then be used to cost-effectively maintain the quality of the metadata content. Showing leader boards of the most valuable users could even provide competition and in this way motivation to voluntarily do this work for RHCe.

6.1.4 Evaluation

We did a small-scale user study for χ Explorer. We will first describe the distinct RHCe user groups.

User Groups

To understand the impact of user tagging and preparing the evaluation of using tags, we consider RHCe's typical user groups more closely. The user base is divided into three main groups, each with their own characteristics:

1. RHCe's largest user group are senior citizens. They are generally very interested in genealogy (usually their own family) and everything they relate to their childhood. They are usually very enthusiastic and are even invited sometimes to supply their stories on certain objects to help the professional annotators with their work. RHCe suspects that this particular user group might form a core of enthusiastic users of the Web application and thus become a great source of metadata (under the condition that the application is kept simple). Browsing through the Map and Time views can help them significantly to find material important to them. They can also be shown new uncharted data.

2. Another substantial user group, and one that RHCe would like to grow, are students. RHCe has contacts with different schools in the neighborhood. These students sometimes get assignments that include finding interesting objects in the archives. RHCe expects less from this group in the sense of new metadata, but they do assume that using the user-generated metadata from the elderly helps the students in finding objects that are interesting for them.

3. The historians form a third group, smaller in size, but important for RHCe. They use the archives to study Eindhoven's history and need metadata to search through the data. They can provide a wealth of metadata, however they are better capable of using a native ontology-based approach instead of a tagging approach as they are already accustomed to ontologies. They probably will not use the χ Explorer directly, but the χ Engine can be invaluable to them for their research.

The current χ Explorer version was made available under experimental conditions (with selected user groups and conditions). Before being released to the general public, the application will go through a number of evaluation steps and user studies to validate the ideas and refine the application based on the specifics of the community. At the same time these explorative steps help RHCe to get a better understanding of the entire metadata creation process that envelops the system.

Focus Group

We did a focus group [Morgan, 1997] to qualitatively review if users appreciate the semantic navigation interfaces in χ Explorer. 15 users participated equally divided in 3 types (i.e. groups of 5 users): external young users (in the range of 15-25 years of age), external senior users (in the range of 55-65 years of age), and RHCe internal users (representing historians). These groups represent the three current main user groups of RHCe.

The participants were given a list with 12 assignments to find and browse objects that were available in the system. An example of such a task was "Try to find photos of the center of Eindhoven that feature buildings damaged in the Second World War". We also advised them to try using all the available facet visualizations. During these tasks we let the users think aloud and filmed their reactions; at the end they also filled out a questionnaire. The users got a list of statements and used scores on a scale between 1 and

	Seniors	Students	Historians
The map view is clear	5	5	5
Searching within locations is easy and clear	5	5	4
The time view is clear	5	5	5
Searching trough the time view is clear	5	5	4
The graph view is clear	5	5	4
Searching trough the graph is easy and clear	5	5	4

Table 6.1: User Study Results for navigation (condensed)

5, where 5 means 'definitely agree' and 1 'definitely disagree'. An 'X' means that there was no agreement within the user group, i.e. the users in those group had opposite reactions.

Table 6.1 is a condensed overview of the most interesting responses from the test group on the navigational facets that were provided to them.

From our observations we found that it took all the users time to get used to the visualization interfaces, especially for the graph visualization. Even for RHCe internal users that recognized the concept descriptor structure. This is also clear from the responses on how easy the search facilities where learnable.

We also saw differences between the groups: the younger users accommodated to the new visualizations more easily than the other groups, e.g. the maps visualization most of them used before, while some senior users overlooked the zoom functionality. All the users however got enthusiastic about the possibilities of the new interfaces after they used those interfaces for some time. The responses in the questionnaire also reflect this attitude towards the χ Framework.

	Seniors	Students	Historians
The graph view should include longer paths	X	2	4
The graph view provides enough relations between concepts	4	5	4
There should be more relationship types	4	X	2
The differences between relationship types are clear	4	5	4
Search capabilities in χ Explorer are sufficient	4	4	4
Using the search facilities was easy to learn	3	4	3
Search history should be stored in the profile	3	4	4

Table 6.2: User Study Results for extensions (condensed)

We also probed the users about possible extensions for χ Explorer, for which the most interesting results are displayed in Table 6.2. Different user groups seem to have contrasting opinions in this matter. For example, the historians preferred to so see a larger part of the concept graph in the Graph view, as they were familiar with the graph already and wanted to navigate faster and combine concepts in searches. The other two groups however strongly preferred the existing graph view as they preferred its simplicity and feared that it might it become too complex if extended. Here we see an opportunity for personalization.

One result of the study was that the users struggled with large result sets within a specific facet. During the study we did not have the clustering feature yet: based on the feedback we introduced the clustering described in Section 6.1.2. Another issue that we are currently looking into is combining facets for searching: users would like to select a region with objects in the map view and then for these objects navigate through the timeline.

Please note that a focus group, in this case with three groups of five persons each does not lead to any hard conclusions. The results of the focus group should only be seen as

indicative, which can be used to make improvements. Future user studies on a larger scale are necessary for definite conclusions.

Evaluation of Relco Suggestions

For an insight into how the Relco component performs in the RHCe case we did some evaluation tests with Relco. Table 6.3 contains two examples of input tags and the suggestions that are provided by Relco that are typical for the RHCe use case (in Dutch). In the first example "christendom" (Christianity) is related to Christian concepts (except maybe the less obvious connection with "koning" (king). In the second example "zang" (singing) is related to music concepts, but also to "zand" (sand) because of the close syntactical resemblance between "zang" and "zand".

Input	Suggestions	Input	Suggestions
christendom	christenen	**zang**	muziek
	religie		liederen
	katholicisme		karaoke
	oosterse kerken		koor
	protestantisme		musical
	aartsbisschop		opera
	bisschop		operette
	christelijk onderwijs		songfestival
	kerk		zand
	priester		geologie
	zonde		grondsoorten
	koning		strand

Table 6.3: Example tag suggestions Relco (Dutch)

Table 6.4 is a summary of our findings of applying Relco in the RHCe use case. The RHCe domain ontology contains 1387 concepts. We had photos from the collection tagged with single word tags to describe the content of the photo (without having knowledge of the ontology). In approximately 58% of the cases Relco gave at least one suggestion. In 8% no result was given, while there was a synonym for that term in the dataset. To improve this recall issue a chaining of two Relco components is possible, by first using it to match on a semantic lexicon (e.g. Wordnet for English) and then passing the result to a second Relco version with the actual (domain) ontology. However, this decreases precision.

The precision of the suggestions (when at least one suggestion was given) averaged on 73% with a standard deviation of 8%. These results can be naturally explained with the underlying structure of the domain ontology. For instance, as a significant part of RHCe's object archives is dedicated to the Second World War, the ontology contains relationships between bombardments and the buildings that were damaged by the bombardments. Therefore a term like "church" is related to "bombardments" in the ontology, while in general users will not consider "bombardment" to be closely related to "church".

Context disambiguation does not influence the precision of the results, only their ordering. When using relevant context, we found in the dataset that in 27% of the cases the ordering of the suggestions improved compared to the case without context disambiguation. Moreover, it never worsened the ordering.

Using user feedback can improve the precision of the suggestion process. The importance of the feedback is configurable, but in the experiment we used a configuration where it

#concepts	#results>0 avg	missed synonyms
1387	58%	8%
precision avg	**context improvement**	**#feedback ordering**
73% (σ=8%)	27%	4.2%

Table 6.4: Data about Relco applied in RHCe domain

took on average 4.2 consistent and correct user feedbacks for a tag to get a 100% precision ordering for that tag. Note that in reality user feedback might not be consistent or correct at all: user tests under those conditions need to show whether we need additional mechanisms to determine which feedback is most probable correct.

Relco Configuration

The quality of the suggestions from Relco depends in general on many factors, such as the richness and accuracy of the concepts and relationships in the underlying ontology. Relco's configuration also has an important impact on its performance. We earlier found that the best way to configure Relco was with help of both users and experts. Given the user's input we asked the domain expert to evaluate (based on his knowledge of the domain ontology) the concepts he found to be relevant for the user and we compared this with suggestions actual provided with the given configuration. In this way, we iteratively fine-tuned the system to an optimal configuration for the current datasets.

	restaureer	platteland	emancipatie	amsterdam	rembrant	christendom	zang	koningin beatrix
Substring matching	0	8	4	160	0	2	31	4
Levenshtein	1	1	3	9	1	3	3	2
JaroWinkler	2	4	3	5	1	4	7	5
Soundex	1506	682	1266	3369	1196	3080	1042	1753
N-gram matching	23	24	24	16	22	20	25	7
N-gram Levenshtein	1	1	3	5	1	3	3	0
N-gram JaroWinkler	2	2	2	2	1	2	2	1
N-gram Soundex	2	4	4	5	1	13	9	6

Table 6.5: The effect of applying different string matching algorithms in Relco

To illustrate the influence of configuration on the results consider the number of results for different Reclo configurations in table 6.5. We used a number of actual user-generated tags in χ Explorer and looked at the influence of different string matching algorithms. The figures in the table represent the number of final results that were computed using the concerning start algorithm. The N-gram technique is used for speeding up computation and can be combined with the other techniques, which combinations are shown in the three bottom rows.

In our application precision is more important than recall. The Soundex algorithm for instance finds most matches of all algorithms, but most matches are not relevant for the

input tag. We think that the problem with Soundex in our case was that it was optimized towards the English language while at RHCe we target Dutch and therefore is not very useful. The substring method found more matches than alternative algorithm because our vocabulary frequently uses some terms as prefix for other terms. For example "Amsterdam" is often used in more specific names, like "Amsterdamse Effectenbeurs" (Amsterdam Stock Exchange). It is debatable if the Amsterdam Stock Exchange is a relevant suggestion for the input tag Amsterdam, but if it is substring matching might be a good candidate in some circumstances.

By looking together with the RHCe domain experts at these results we found that both N-gram Levenshtein and N-gram JaroWinkler are generally good trade-offs in speed, precision and recall for the RHCe domain.

6.1.5 Related Work

Interest and research in using Semantic Web techniques in the cultural heritage domain has been flourishing lately [Hardman et al., 2009]. This research shows that Semantic Web techniques can be used in several additional ways besides the one that we discussed in this section. For example, there is progress in extracting metadata from natural language descriptions [Ruotsalo et al., 2009], using encyclopedias for creating ontological knowledge sources[Witte et al., 2009] and using analysis of materials and techniques in artworks for disseminating the collection to different types of users [Karagiannis et al., 2009]. And these are only some of the subjects which are explored.

The navigation from χ Explorer builds upon related work. For instance, mSpace [Schraefel et al., 2005] and /facet [Hildebrand et al., 2006a] present faceted browsers. These are more general and more flexible than χ Explorer, as they are able to use any RDF-property as basis for a facet. Their approach supports a powerful but easy query paradigm, which is something we could incorporate in χ Explorer as well. Choosing some well-defined facets obviously has the advantage that one can build very specialized visualizations for them, like we did with Google Maps, Simile Timeline and Graphviz. /facet actually has support for specialized facet visualizations as well and also MuseumFinland did something similar in the recent CultureSampo semantic web 2.0 portal for cultural heritage [Hyvönen et al., 2008]. CultureSampo is a much more extensive Web application than χ Explorer. For visualization it actually uses the same visualizations as χ Explorer (however in more versatile ways), even though it does not incorporate a clustering algorithm. The main difference with our work is that we focused a lot on user-based input for not yet well-annotated collections, where they used already well-structured datasets.

While the χ Explorer project is smaller than the ones in literature, the results show, and that was one of the research challenges at the start of this project, that the semantic techniques we use are applicable and useable for small/medium parties like the regional historic centers in the Netherlands. The discussed techniques solve issues that organizations of the type of RHCe have been struggling with for a long time.

6.2 The SenSee TV-Recommender

The second Web application that we examine in this chapter is called SenSee. SenSee is a personalized Web-based recommender for digital television content. It was developed in the context of the Passepartout [Passepartout, 2007] project, which investigates the future of digital television. SenSee collects information about TV programs as well as related data from various sources on the Web. Furthermore, the user can access his personalized

TV guide from various devices at any time, so not only on the TV itself. To provide high-quality personalization SenSee collects data on the user. It provides the user with direct feedback options to provide his information manually, but also observes the user's behavior and with help of sensor information like GPS determines the user context.

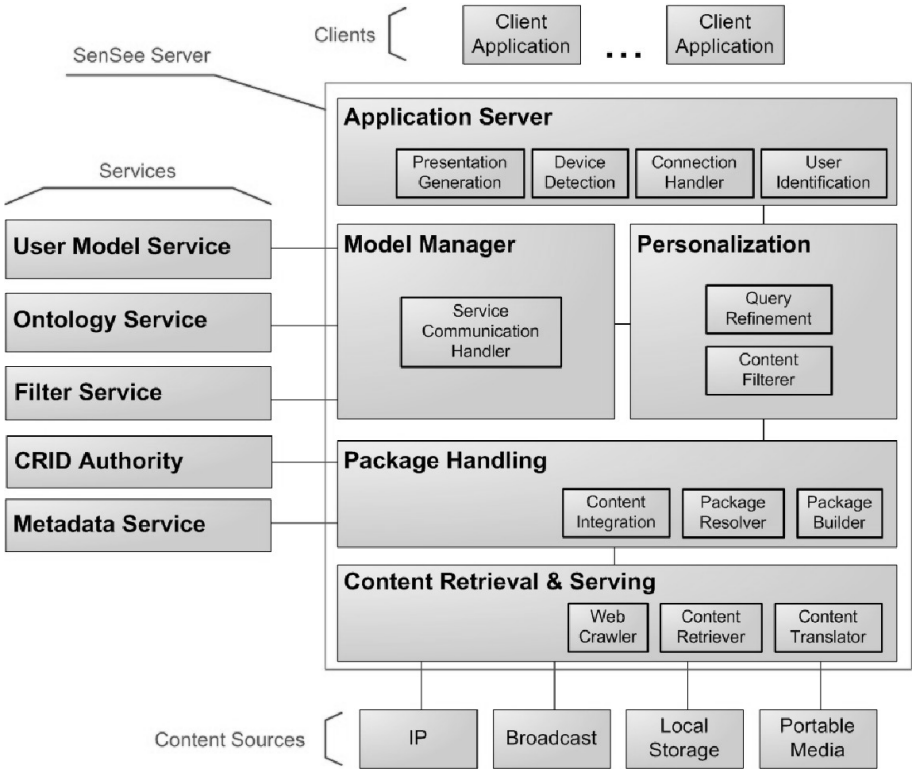

Figure 6.9: SenSee Architecture

6.2.1 SenSee Architecture

The SenSee architecture, as depicted in Figure 6.9, is a layered one. At the bottom part of the figure, we find the various heterogeneous content sources. This can include regular TV broadcasts, but also movie clips from the Web, media stored on the local computer or physical media like DVD and Blu-ray discs. Metadata about the content originates from the bottom, propagates up while at each layer gaining semantics and context, and is finally used by an application at the top. In this process first information about the content is collected, and then converted to our internally used vocabulary and aggregated into packages. In order to do so we use two external services that are specified by the TV-Anytime specification[5], namely the *CRID Authority* for uniquely identifying multimedia objects and the *Metadata Service* that maintains metadata on these objects. Since typically there are numerous content packages available, we use personalization and recommendation based on the context and the user model with the aim to prevent that the user gets lost in the information abundance. Furthermore, we offer the user a user-guided search to assist the user in finding what he or she is looking for by refining a user query (keywords) with help of ontological domain knowledge. Both for personalization and user-guided search we use supporting services, each with their own responsibility and functionality. The User Model Service (UMS) maintains and retrieves data from the

[5] http://www.tv-anytime.org/

user model (including the current user context), the Filter Service (FS) provides filters needed to eliminate content unsuitable for the current user, and the Ontology Service (OS) maintains and manages all vocabularies and ontologies defining the semantics of concepts and properties. Finally, the metadata is shown to the user in a hierarchically structured way: the user can browse through the data and select the desired multimedia object. The object can then be retrieved via its unique CRID identifier [Earnshaw, 2005] for consumption (viewing).

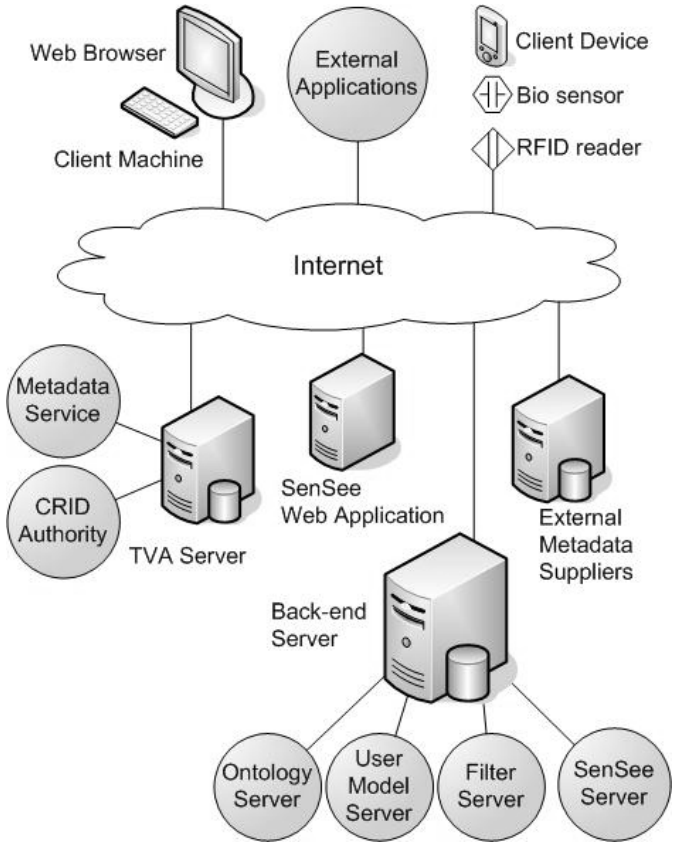

Figure 6.10: SenSee Connection Architecture

Figure 6.10 shows how the different parts of the SenSee framework are connected. The user can access SenSee via a client application. Several client applications exist. In this section we focus on the SenSee Web application, which we built as a proof of concept. The SenSee Web applications allows users to search and browse program and movie information, based on semantic relationships between both interests of the user and between the programs and movies themselves. Next to the academic proof of concept, a commercial front end application called iFanzy[6] has been built by our commercial partner Stoneroos. iFanzy is a set of front end applications including a Web application, a set-top box front end that is controlled by the television remote control and an IPhone app for mobile tv recommendations.

The SenSee Web application connects to the SenSee server, which besides the SenSee Service (which contains the main application logic) also contains the three supporting services: UMS, FS and OS. The TV-Anytime services, CRID Authority Service and Metadata Service, as well as the content metadata that is not stored locally, are accessible via the Internet. Sensors and device information are accessed via external applications

[6] http://www.ifanzy.nl

which are responsible to reading the sensor values and adding or updating them in the user model. Having SenSee running as a service enables these various applications to quickly make a connection and push their readings to the system. SenSee on its turn processes these readings and can immediately take them into account in upcoming actions.

6.2.2 Semantic Integration

SenSee offers users an integrated view over several heterogenous datasets. Achieving data integration was a similar effort as we did for χ Explorer, i.e. we used RDF as our underlying data representation and modeling format and harmonized metadata that is important for navigation in our application. We will mainly focus on the specifics for SenSee.

When a user sends a request, SenSee responds with an integrated set of content containing pieces from various sources. As previously mentioned, SenSee retrieves content and metadata descriptions from these various sources which comply with different data schemes. However, since SenSee is completely built around the TV-Anytime scheme, we chose to transform all incoming content to TV-Anytime. The TV-Anytime specification consists of two parts, "phases" in TV-Anytime speak. The first phase consists of a synchronized set of specification documents for metadata, content referencing, rights management, and content protection. The second phase defines open standards that build on the foundations of the first phase and includes areas such as targeting, redistribution and new content types. A central concept in the second phase is packaging. A package is a structure which contains a set of unambiguous identifiers referring to content-elements which are somehow related. An example would be a 'Lord of The Rings' package containing all the movies, Making-Ofs, the text of the books, images, the soundtracks, etc. To be able to reuse this generated content we add a sense of context to each package from which we can benefit in future requests.

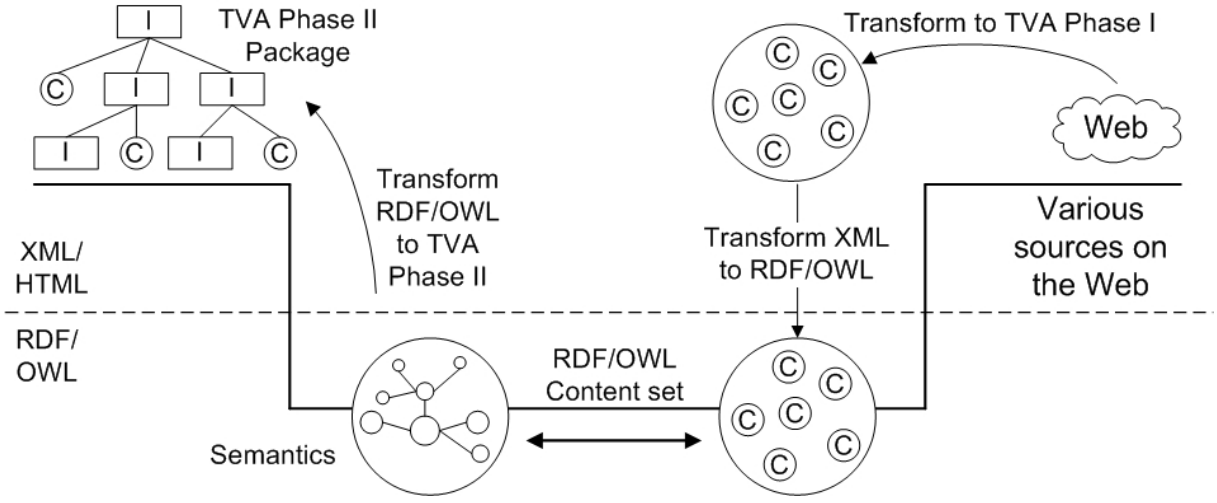

Figure 6.11: SenSee data integration scheme

In Figure 6.11 we see the complete SenSee data integration scheme. In this figure we see that there are two layers visible, namely XML/HTML at the top and RDF/OWL at the bottom of the figure, depicting a clear separation between the two. On the right hand side we see our various sources located on the Web, these sources are typically retrieved in HTML (Wikipedia, IMDB, ...) or XML (XML-TV and BBC Backstage content) formats. This content is then transformed to TV-Anytime Phase I (XML). However, in order to apply our semantics available in the OS which are typically modelled in RDF/OWL, we

need to transform the TV-Anytime XML (Phase I). This second transformation crosses the dashed line bringing the TV-Anytime Phase I content set into the RDF/OWL layer. All content here is stored in a Sesame RDF store which serves as a temporal cache to augment and work with the retrieved data.

In order to integrate the content data in SenSee we in general followed an extendible three step alignment procedure:

1. *Making TV metadata available in RDF.* Adding a new TV metadata source to the system requires it to be in RDF format making sure that it fits and can connect to already available data sources. In SenSee we use three live data sources: online TV guides in XMLTV format[7] (e.g. 1.2M triples for the daily updated programs), online movie databases such as IMDB (83M triples, representing 1M movies and trailers from Videodetective.com), and metadata available from BBCbackstage (e.g. 92K triples, daily updated). All three sources were parsed and converted to our TV-Anytime domain model in RDF.

2. *Making relevant vocabularies available in RDF.* Having the metadata available, it is also necessary to make relevant vocabularies available in RDF. In SenSee we did this in a SKOS-based manner for the genre vocabularies (resulting in 5K triples). All these genres play a role in the classification of the TV content and the user's likings (in order to support the recommendation). We also used the locations hierarchy used in IMDB (60K triples), Word-Net2[8] (2M triples) and the OWL Time Ontology[9] both as published by W3C.

3. *Aligning and enriching vocabularies/metadata.* Here we did (1) alignment of Genre vocabularies (e.g. aligning imdb:sci-fi and tva:science fiction), (2) semantic enrichment of the Genre vocabulary in TV-Anytime (e.g. relating TVAGenres:Sport and TVAGenres:Sport News), and (3) semantic enrichment of TV metadata with IMDB movie metadata (e.g. connecting the broadcasted movie "Il Buono, il brutto, il cattivo" to the IMDB movie "The Good, the Bad and the Ugly").

The advantage of our approach of using RDF is that the resulting integrated datamodel can be easily adapted for plugging in and removing data sources and relating those to supporting ontologies because of the open world nature of RDF. Relating datasource to the supporting ontologies can be done by using the Relco framework as described in chapter 5.5.1. For the online sources this results in STL mapping rules that are applied whenever an incremental update is done.

Let us look at the following typical example. Imagine we retrieve metadata of two separate TV programs broadcasted on different channels retrieved from different sources (with some syntax simplification for brevity reasons):

```
<program channel="NED1">
  <source>http://foo.bar/</source>
  <title>Sportjournaal</title>
  <start>20120309184500</start>
  <end>20120309190000</end>
  <genre>sport nieuws</genre>
</program>
```

[7] xmltv.org
[8] http://www.w3.org/2006/03/wn/wn20/
[9] http://www.w3.org/TR/owl-time/

and

```
<program title="Match of the Day">
  <channel>BBC One</channel>
  <start>2012-03-09T19:45:00Z</start>
  <duration>PT01H15M00S</duration>
  <genre>sport</genre>
</program>
```

Both programs are sport programs broadcasted on the same day. However, since the data is coming from different sources, the syntax and semantics differ. As mentioned we transform all instances from our different sources to the central TV-Anytime domain in RDF using STL mappings. After this transformation the generated RDF looks (for the first program) like:

```
<TVA:ProgramInformation ID="crid://foo.bar/0001">
  <hasTitle>Sportjournaal</hasTitle>
  <hasGenre rdf:resource="TVAGenres:Sport_News"/>
</TVA:ProgramInformation>
<TVA:Schedule ID="TVA:Schedule_0001">
  <serviceIDRef>NED1</serviceIDRef>
  <hasProgram crid="crid://foo.bar/0001"/>
  <startTime rdf:resource="TIME:TimeDescription_0001"/>
</TVA:Schedule>
```

RDF also allows to use inference techniques (e.g. subsumption) to get more useful data for the user and we can utilize mechanisms like the context feature in Sesame 2.0[10] to track the origin of information. Once here, we can apply the semantics stored in the OS to enrich the current content set. For example, every TV-Anytime Phase I content set always has a set of keywords to describe this content in its metadata. The TV-Anytime (TVA) field for this is:

```
<element name="Keyword" type="tva:KeywordType"
                    minOccurs="0" maxOccurs="unbounded"/>
```

We can make a simple enrichment by searching for synonyms of these keywords, in the WordNet dictionary, and expand the existing set. Similarly, we can also add semantics to time specifications. In TVA every time-related field is modeled by an MPEG-7 'timePointType', where a time point is expressed with the regular expression:

```
'-'?yyyy'-'mm'-'dd'T'hh':'mm':'ss('.'s+)?(zzzzzz)?
```

The date '2012-01-01T12:30:05' is an example. By mapping this string to a time description in our Time ontology we can obtain the following time specification (do note the abbreviated syntax):

```
<CalendarClockDescription rdf:ID="example">
    <second>05</second>
    <hour>12</hour>
```

[10] http://www.openrdf.org/doc/sesame2/users/ch08.html#d0e1238

```
    <minute>30</minute>
    <year>2012</year>
    <month>01</month>
    <week>01</week>
    <day>01</day>
</CalendarClockDescription>
```

This richer time description enables us for example to easily cluster all content produced in a particular year or all content being broadcasted before or after a certain point in time.

6.2.3 Inference, Query Expansion and Recommendation

In the previous sections we demonstrated solutions for the data integration issues. However, the described techniques could also be used to improve the core personalization functionality of the application. Besides offering 'passive' recommendations, the user can also actively search through the available content.

Our approach is based on semantic faceted browsing techniques (e.g. similar to Section 6.1.2 and [Yee et al., 2003; Hildebrand et al., 2006b]). The idea is to go beyond a pure keyword-based search. Instead the user is assisted to search through different *facets* of the data. To accomplish this, additional metadata fields as defined in TV-Anytime specification were used. Moreover, as previously mentioned, ontological sources to make a further semantical connection between the terms in TV-Anytime play a important role. Consider for instance the time facet, TV-Anytime uses the XML datetime datatype to express the time when a program is broadcasted. To semantically enrich time information we use the W3C OWL-Time ontology, adding for instance that the concept 'afternoon' ranges from 1400h-1800h. In this way we can easily search for programs in the afternoon by looking for all programs broadcast within this 1400h-1800h interval.

By incorporating all these results, as is shown in the screenshot in Figure 6.12, we try to close the gap between the keywords the user types in and what the user intends to search for. As an example, consider a user making the following keyword request: "sports Scotland 'Friday evening'" when he looks for Scottish sports games to watch during Friday evening when his friends come over. The system reacts by trying to find any ontological matches for these keywords, through the use of the OS. In Figure 6.12 the results are shown. Under the tab 'Genres' we see that the system found some matches for the keyword 'sport' in the genre classification and in the Geo ontology a match was found for 'Scotland'. In the time ontology a match was found for 'Friday evening' and automatically translated to an interval from 18pm until 23pm as shown in the calender under the 'Time' tab. With this system-assistance the user is able to manually refine the original request. With the genre hierarchy it is possible now to specify a more particular sport type the user might be looking for, or just select the broad term 'sports'. In the geographical hierarchy either a narrower term, like a region in Scotland, or a broader term like 'United Kingdom' can be chosen. Via the calender the user can revise his time interval.

After the system-assistance, the newly refined query is taken as input for the retrieval process which tries to find relevant content made available by the various sources. When the sources start responding with multimedia content, the retrieved content needs to be filtered and personalized by leaving out items deemed unsuited, and ranking important item higher up the results list. This process, which forms the heart of the personalization process, uses mainly the UMS to find the user's preferences, the FS to provide the appropriate filter, and the OS to give correct semantics throughout the complete sequence. The filter

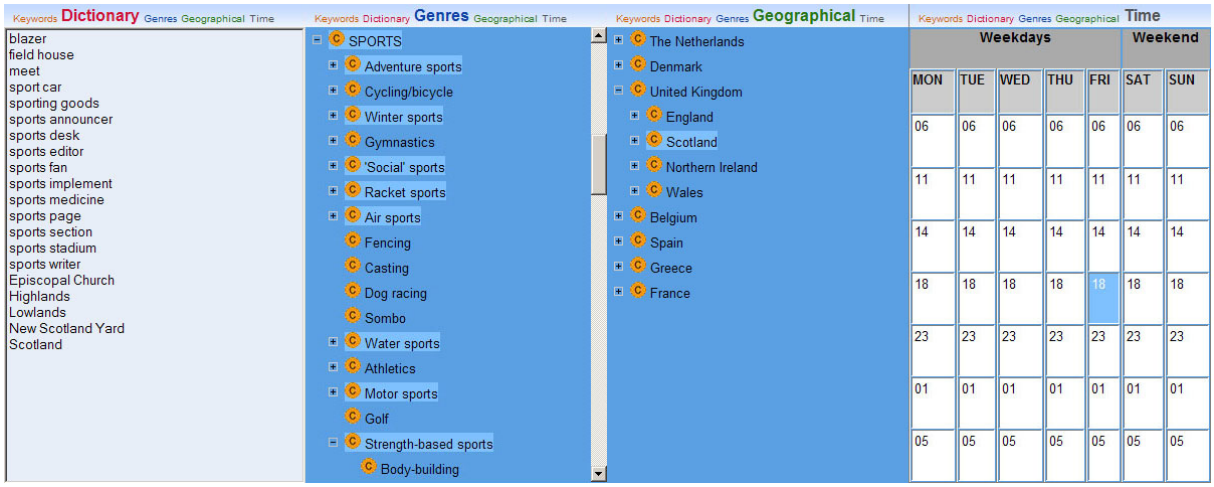

Figure 6.12: User-driven concept refinement

provided by the FS basically acts as the glue between values in the user model and fields in the content's metadata. It decides for every content-element whether or not it should be discarded after comparing its metadata to the interests and preferences of the user. Lastly, the resulting content set needs to be presented to the user, by means of the TV-Anytime packaging concept. Packages allow us to create for example a 'Marlon Brando' package were everything known from this person can be bundled and browsed. Packages can contain all kinds of content-elements ranging from pure text to full HD (High Definition) movie clips. Like this, the results are clustered and hierarchically structured to obtain an easy-to-browse content-container where the user finds what he or she was looking for or at least is interested in.

SenSee also allows recommendation of programs. Recommendations are computed based on the user model. Users can rate programs, which are stored in the user model. Using these ratings SenSee can compute semantically related programs based on the underlying ontology. The recommendation algorithm can for example recommend programs of the same or a related genre, programs with the same or related presenters or actors, filmed on the same or neighboring location, etcetera. For a more extensive treatment of several recommendation strategies refer to [Bellekens, 2010].

The SenSee framework is in general hard to evaluate as most of it are underlying services that allow the system to 'just' work. However, user recommendation is something that can be tested. Table 6.6 contains figures of a user recommendation evaluation study. In this study 60 participants used the SenSee recommender system. In order to overcome the *cold start problem* historic television watching statistics were used from Stichting Kijk-Onderzoek[11]. For every program since 2008 this information includes watching statistics for the Dutch people, including a breakdown in demographics like age, gender and education. As these demographics vary greatly for different programs these seem good first general indicators of program likings. Therefore, during registration to the recommendation system uses were asked to give these three personal characteristics which were then mapped on programs that correspond to those characteristics.

Using this mapping a first set of recommendations could be made and the users were asked to provide feedback about the quality of these recommendations via a questionnaire, which correspond to the rows of table 6.6. Then the users could start rating specific programs and based on those ratings and the underlying ontology more personal user

[11] http://www.kijkonderzoek.nl/

beginning of test \ end of test	almost never	sometimes	regularly	often	almost always
almost never (19%)	0%	40%	30%	30%	0%
sometimes (38%)	0%	35%	30%	30%	5%
regularly (32%)	6%	29%	24%	35%	6%
often (11%)	0%	17%	33%	50%	0%
almost always (0%)	0%	0%	0%	0%	0%

Table 6.6: Recommendation satisfaction evolution

recommendations could be made. After two weeks the users were questioned again with a questionnaire, which figures can be found in the columns of table 6.6. The figures show that semantic recommendation improved the recommendations for 55,8% of the users. However, 26,3% of the users claimed that the recommendation remained roughly of equal quality, while 17,9% of the users indicated that the recommendations decreased in quality. It is hard to specifically determine why recommendations for 44,2% of the users did not improve, but probably the main reason was that some semantic links might be less important for the liking of a program than we thought for these users.

6.2.4 Related Work

Several related TV recommender systems can be found in literature [Ardissono et al., 2004; Blanco Fernández et al., 2006; van Setten, 2005]. However, those systems mainly focus on the recommendation part. In AVATAR [Blanco Fernández et al., 2006], for example, the authors make use of a combination of TV-Anytime [TV-Anytime, 2003] metadata fields and a custom-made genre hierarchy. Their recommendation algorithm is an effective hybrid combination of content-based and collaborative filtering, although there is no inclusion of any kind of user context. To get an overview of the various kinds of recommendation strategies and combinations that exist both for selection, structuring and presentation of content refer to [van Setten, 2005].

In SenSee, however, we focused on the framework which serves as the backbone of the SenSee recommender. This included a data integration issue that we solved using RDF and standards like XMLTV and TV-Anytime to incorporate one extendible graph-structure. We also focused on a multi-device implementation, which recognizes the growing number of devices available to connect to a television, e.g. PC, mobile devices and set-top boxes. In the end, the main objective of our SenSee framework is to provide techniques to improve personalization in the world of multi-device access to interactive Web applications connected via semantic-based integration.

More extensive description of the SenSee framework and related systems and its evaluation can be found in [Bellekens, 2010].

6.3 The ViTa Educational Videos

The ViTa project was meant to research and review techniques that allow to find more relevant information by its users. As part of the project the ViTa framework was built, which is a Video Tagging system for educational videos. The system provides access to a collection of video clips that are supplied with professional metadata such as keywords, title and description. The metadata were provided by Teleblik[12], a Dutch organization that provides access to multimedia sources for students in secondary education. Two other stakeholders in this project were SURFnet[13] and Kennisnet[14]. SURFnet is a Dutch national non-profit organization that provides internet access and internet-related services to most higher education institutes (like universities) and research centers. Kennisnet is a Dutch public organization dedicated to providing IT-services for primary, secondary education and vocational training. Both have a videoportal[15] with many educational videos, but with also very little known metadata about most videos.

6.3.1 Semantic Integration of User Tags

The goal of the ViTa project was to study and review the number of relevant videos that students can find via keyword search by using various types of metadata. The video collection that is used in ViTa has professional metadata. However, as many videos in the collections of Teleblik, Kennisnet and SURFnet do not have this professional metadata (yet) other alternatives were considered as well. The idea was to experiment with different metadata sets on the same video collection and to compare the different types of metadata in a user test with students to see how the different metadata approaches help the user to find the videos they are looking for.

One way to acquire this metadata is user tagging. In this setup we first let a group of students watch the videos and asked them to provide tags. We wanted to evaluate if semantic integration, in this case by relating tags to a semantic structure or other tags, helps students providing those tags so that efficiency in finding the right videos later on improves. We tried different scenarios in extending the tags. The typical scenario in which Relco is used is depicted in Figure 6.13.

The user provides a tag for a video and Relco is used to match the tag with concepts in an ontology. The user then gets a list of textual representations of concepts as suggestions. If such a suggestion is used, the video is effectively annotated with a concept from the ontology and a link is established between the particular tag and the chosen concept. This approach is based on the assumption that using the structured ontology provides an effective way to browse and search the ontology. That using ontologies in browsing and searching can be effective has been demonstrated in various faceted browser solutions, e.g. [Schraefel et al., 2005; Hildebrand et al., 2006b].

Another approach that we tried in this study is depicted in Figure 6.14. Users provide tags and Relco is configured to use the existing set of user tags to come up with suggestions from amongst previous user tags. In order to use Relco effectively in this domain we configured it to use the Dutch Common Thesaurus for Audiovisual Archives (GTAA) ontology [Brugman et al., 2006], to find semantic relations between tags. GTAA is intended to describe TV program metadata. The GTAA contains approximately 160.000 terms:

[12] `http://www.teleblik.nl/`
[13] `http://www.surfnet.nl/`
[14] `http://www.kennisnet.nl/`
[15] `http://www.surfmedia.nl/` and `http://video.kennisnet.nl/`

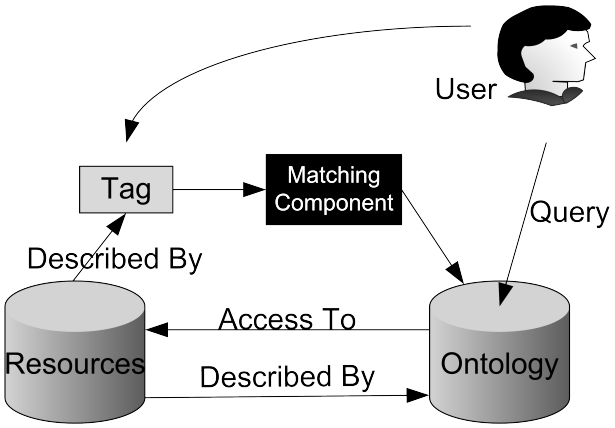

Figure 6.13: "Relating tags to ontology (concepts)" scenario

∼3800 Subjects, ∼97.000 Persons, ∼27.000 Names, ∼14.000 Locations, 113 Genres and ∼18.000 Makers.

This approach helps us in two ways. First, we can find relations between tags and thus build a structured ontology based on the user tags (which also has been tried in [Specia and Motta, 2007]). Second, it allows consolidating the tag set where only the most popular tags are shown and the less frequently related tags will be hidden as alternative for the more popular ones, i.e. we hide unpopular tags and show the more popular related tags as alternative.

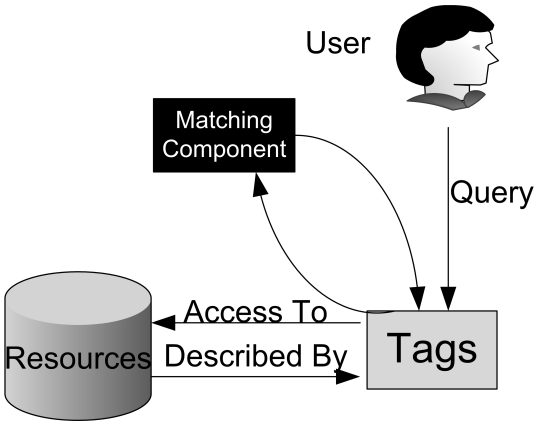

Figure 6.14: "Suggesting previous tags" scenario

A third way to acquire metadata that has been used in ViTa was automatic extraction of relevant tags based on documents that described a video. The videos in the ViTa dataset were described in short documents that contained a natural language description of the video (between half a page and a page long). With a keyword clustering technique the keywords that best describe the topic are detected; the technique is described in [Wartena and Brussee, 2008]. Next, Relco was used to automatically extend this set of keywords with semantically related terms. We used again the GTAA ontology for this. Together these keywords and their semantic extension provide a sufficient description for the video over which the user can search.

Tag Application	Total # tags	Total # unique tags
BasicTagger	3742	2635
SocialTagger	3513	2091
SmartTagger	3880	2401

Table 6.7: Normalized total number of tags per tagging application

6.3.2 Evaluation

These various scenarios have been evaluated in an elaborate user study that studied the effects of user tagging on searching educational videos. The evaluation is described in detail in [Melenhorst et al., 2008]. The concise evaluation overview in this section focusses on the comparison of semantic techniques based on Relco with other approaches.

The study was performed with students of a Dutch high school. The user study consisted of two phases. The tag phase and the search phase. The domain of the study was the art domain, for which a collection of 115 educational videos were available, all between 1 and 2.5 minutes of duration and initially without a five line description provided by professionals.

During the tag phase the students were asked to provide all tags they could think of that would help finding the videos. A total of 194 students participated, of which 97 male and 97 female. Their mean age was 15.5 years old with a 1.1 year standard deviation. Of the 194 students, 135 students were studying on the VWO (higher) level and the other 59 students studied on the HAVO (intermediate) level.

The students were blindly and uniformly divided over three different tagging applications. The *BasicTagger* provided the students no tag suggestion. The *SocialTagger* provided tag suggestions based on the most popular tags used by other students who tagged that video. The *SmartTagger* provided tag suggestions by using Relco, which used the GTAA ontology for providing suggestions on basis of the user input.

The results of the tagging phase are summarized in table 6.7. It is remarkable that the social tagger leads to significant less tags than the other two tag applications. Even though not exactly clear why this is, one can hypothesize that the social taggers allow students to converge and agree on their vocabulary. This might give less incentive to use semantically equivalent terms. Also remarkable is that the SmartTagger resulted in most tags overall but has fewer unique tags than the BasicTagger. This is probably due to the (relatively) limited GTAA ontology that the tag suggestion algorithm can choose from, even though the effect is significantly less than with the social tagger.

During the search phase the students were asked to answer (the same) eight questions each, by searching for the right video clips. Experts determined the right answer beforehand. When students found the right video clip this was made clear in the interface with a 'right-answer-button'. A total of 140 students participated, of which 68 male and 72 female. Their mean age was 15.6 years old with a 0.8 year standard deviation. Of the 140 students, 61 students were studying on the VWO (higher) level and the other 79 students studied on the HAVO (intermediate) level. The interface was restricted to searching only, i.e. no general browsing. The questions and accompanying answers were chosen such that the terms used in formulation of the questions alone were not too obviously related to the keywords related to the answer.

The students were blindly and uniformly divided over five different searching applications. The BasicSearcher could search in professional metadata based on keyword extraction of the approximately five-line video description that was available for every

Search Applications	*Participants*	*# Tags*	*Answers (mean)*	*Correct (mean)*
BasicSearcher	25	1932	6.4 (s.d. 2.3)	4.5 (s.d. 2.2)
BasicSocial	22	6305	6.7 (s.d. 1.6)	5.0 (s.d. 1.6)
DocumentSearcher	31	4022	5.3 (s.d. 2.2)	3.5 (s.d. 2.1)
SmartSearcher	31	1528	6.2 (s.d. 2.2)	4.8 (s.d. 2.0)
SocialSearcher	22	4373	6.7 (s.d. 1.4)	5.5 (s.d. 1.6)

Table 6.8: Evaluation figures on the ViTa Search Applications

video. The SocialSearcher allowed to search in user tags as were captured in the tagging phase, i.e. this were the direct user tags for the video excluding the smart tags that resulted by confirming a tag from the SmartTagger that originated from the GTAA ontology. The BasicSocial searcher combined the tags from the first, to see the delta of user tags over professional tags. The Document searcher used tags based on natural language processing of documents that were related to the videos. And finally, the SmartSearcher used only those tags that were suggested by the SmartTagger based on the GTAA ontology and taken over by the users tagging the videos.

Table 6.8 contains the results of the search phase. The only significant underperforming search application is the document searcher. This seems to indicate that automatic keyword extraction leads to keywords that are not suitable enough to always lead to the right answers. The SocialSearcher outperforms all other search method. Notably, extending the social tags with professional tags does not improve the search quality. On the contrary, it seems to degrade the search results somewhat. But please note that this difference is not statistically significant and no solid conclusions can be drawn from this difference. Also the difference between the SmartSearcher based on the underlying GTAA ontology and the Social and BasicSearcher is not statistically significant. Surprisingly the SmartSearcher leads to similar results with a significant smaller search space than the other search engines. Also, the search space is limited to concepts in the GTAA ontology that is specifically tailored to the tv domain not the art domain.

This shows that using semantically structured metadata is a promising way of tagging and searching multimedia videos. Much depends on the used background ontology, however. The better the underlying ontology contains concept suitable for describing the content of videos, the better it will perform. Not many domain specific ontologies are available in the Dutch language, which makes the application of Relco less obvious in this specific case. However, we have shown that even with an ontology that is not precisely tailored to this domain the results are potentially on par with general collaborative techniques.

6.4 Summary

In this chapter we looked at three applications in three different domains that are only related in that they have some need for data integration. We have looked at the specific data integration need of these applications and how we solved, partly by using the Relco framework which was introduced in chapter 5. For all of these applications we saw how these applications benefit from data integration. Next to that we have also looked at some of the explicit needs and consequences which were the result of these added capabilities.

For χ Explorer we have seen an integration of several legacy databases. Data integration was done by translating these various databases into an RDF representation that was then mapped to a common metadata schema developed by RHCe experts that captured

almost everything available in the various database. Most importantly were three data integration processes, namely harmonizing the location and time data into one common format and syntax and mapping between metadata descriptors and concepts in an ontology (developed by RHCe). These three 'facets' were used as basis for navigation within the Web application via specialized views, i.e. a map view, timeline and a graph view. Furthermore, we introduced a social tagging functionality that allows users to provide tags for existing, but especially for objects without metadata in hope that the users can provide the necessary metadata. Here we chose tagging because of its simplicity, but we extended it with Relco to give suggestions from the expert ontology in the hope we could connect user tags to the concepts from this ontology.

SenSee is a TV-recommendation system. At first glance SenSee is a TV guide application. However, besides the normal program information which can be found on the Web, this data was extended with movie information from IMDb and Wikipedia information for shows, presenters, etc. These datasets were coupled by both transforming the data into RDF and couple the related data items via the datacoupling techniques as provided by Relco. The result of this process is a large program graph. It allows to navigate in several ways, for example by searching programs of the same genre, or that share the lead actor or director, etc. This could also be used for recommendation, e.g. whenever someone rates a program, this rating is propagated and implicitly used to rate semantically related programs. This information can be used to recommend users semantically related programs to the programs they like.

ViTa was a tagging evaluation framework that was built to evaluate several ways of collecting metadata about educational videos for improving finding data within these videos. The results favored using collaborative techniques, but semantic search was still significantly better than using professional metadata while the background ontology was not even specifically tailored for this task. Based on these results, it might be interesting to try and combine these kinds of techniques in this application domain.

Chapter 7

Exchanging User Models

In this chapter we investigate how we can apply data coupling techniques to the specific domain of user modeling. Many applications use user models for personalization and benefit from knowing as much as possible about a user. One way for applications to increase the knowledge about a user is by exchanging user model data between applications. We discuss how data coupling in the user model field presents some unique challenges that need to be dealt with. We illustrate user model data exchange in two project, namely the MobiLife project and the GRAPPLE project. MobiLife is a project that looks at mobile technology, and especially how by using user model exchange the mobile phone can act as a personal assistance that allows context-based personalization. The GRAPPLE project looks at integration of adaptive hypermedia frameworks and learning management systems. By exchanging user model data these systems can achieve tight integration which allows students to combine the strengths of these systems and for example allows students to simultaneously study at several institutions. Based on our work within GRAPPLE, we also found use for defining an engine-independent adaptation language. This language, called GAL, could be used to exchange adaptive applications between different adaptive hypermedia systems. This can improve reusability of these applications, e.g. if different institutions use different adaptive systems. GAL is based on the Hera-S application model we presented earlier, and is for the propose of GRAPPLE extended to deal with all relevant types of adaptation. Also a formalization of GAL is presented as a first possible step towards standardization.

7.1 Introduction

Web applications must because of their nature be able to adapt to a wide and heterogenous audience. In many cases this means that personalization is needed for adapting to the many possible different circumstances or knowledge levels of the users. In order to do so, many Web applications maintain user models. The more they know about the user, the better they are able to adapt to them.

However, most Web applications only have a limited view of the user. Most Web users only spend a limited amount of time per application. This leads to a fragmented perception of the knowledge of the user of several applications. It also gives an additional burden for the users as for a new application they may have to reenter user information that they already have entered in several other systems. One strategy to overcome this

issue is exchanging user information between applications.

However, exchanging user information is not trivial. First there are obvious highly needed privacy and security issues that need to be taken into account. However, because of the depth and broadness of these topics we will leave those issues out of consideration and only focus on the technical challenge of exchanging data. Technically there are plenty of challenges to overcome. For instance, not all applications will be interested in the same user information, so the first challenge is in finding the parts of information that are interesting for the applications to exchange. If these model parts are identified two important issues remain, namely user and context identification and semantic heterogeneity of the underlying data models.

These issues are discussed in more detail below. Afterwards we will illustrate how we solved different combinations of these issues in two different projects, namely MobiLife and GRAPPLE.

7.1.1 User Model Context

User model data inherently belongs to a specific person and these persons might be known differently to different systems. Some applications identify users by an application specific user identifier (e.g. "userAB34"), others use a user's name (e.g. "Kees van der Sluijs"), others us a general identifier like a university wide identifier (e.g. "s441340") and yet others use a national identification number (e.g. "123456782"). Knowing which user the system is dealing with is crucial because of the possibility of pollution of user models with false data, but also because of privacy considerations. Therefore, establishing and mapping the user identity between systems is a precondition before data can be exchanged.

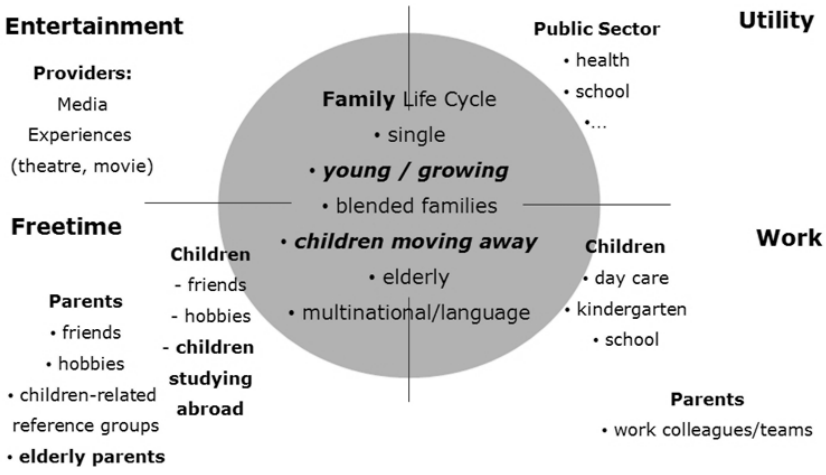

Figure 7.1: Different Social Contexts Users

Next to identification also the user context can be an important factor to take into account. The user context can be defined as any information that can be used to characterize the situation of a user [Dey, 2000]. The context of a user is interesting because users can have different roles, different preferences and different goals in different contexts. Figure 7.1 depicts in which context typical people might interact. Some applications use contextual information to determine the right subset of the user model that is appropriate for the given situation. Consider for example a news application that tries to find relevant news articles for a user. At work this user might be only interested in news that relates to his work, e.g. economic news for an economist. After work the user might be interested in

sports and entertainment news and much less economic news. In this case the relevant context 'being at work' changes the user preferences.

Selecting the proper context and mapping this context between context aware application can be seen as a separate step in the user model exchange process. Also the user context should lead to changes in the user interface, as for example in UsiXML [Stanciulescu et al., 2005]. This process deals more with selecting and comparing similar contexts in different applications than actually mapping data.

7.1.2 Semantic Heterogeneity

The most important step in the user model exchange process is dealing with semantic heterogeneity. This issue is illustrated in Figure 7.2. This figure contains a screenshot of the registration screen of two large Dutch web applications, one is a Web store called Bol.com while the other is a Dutch computer enthusiasts community called Tweakers.net. It is clear that in this case the two information needs have similarities, but also contain unique items for the specific Web applications. Note that this is a simplified example. With an eye on semantic web systems we will not assume user models to be lists of attribute-value pairs, but rather allow for any graph-based structure.

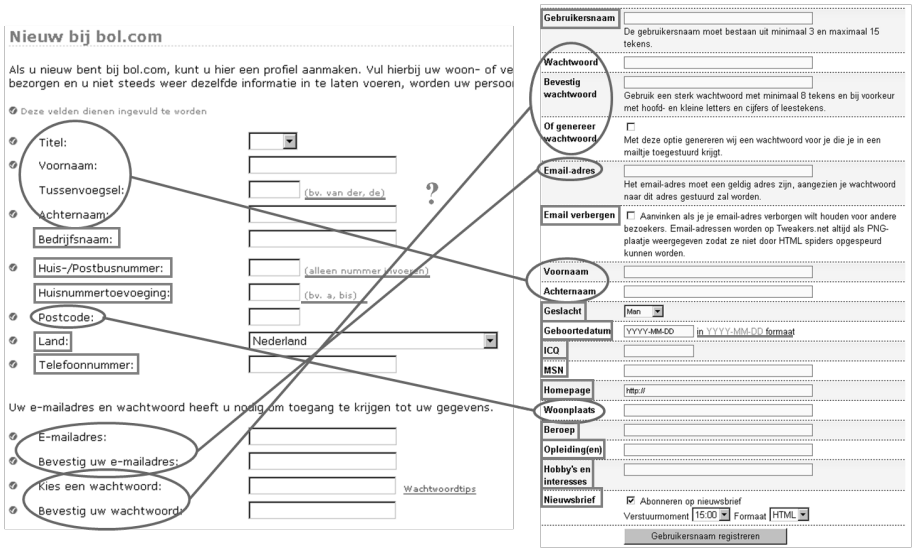

Figure 7.2: Lack of Semantic Interoperability Between Applications

An example of a similar information need is related to a user's name. What we see is that even though the user name might be a simple piece of information, there is still a variety of ways to separate it into different elements. Bol.com separates name elements into 'title', 'first name', 'infix' (common in Dutch names) and 'last name', while Tweakers.net only discerns 'first name' and 'last name'. If we consider the name "ir. Kees van der Sluijs", this will be separated as is depicted in table 7.1. As can be seen, the Tweakers.net application is not interested in titles and doesn't discern infixes from last names.

Mapping the user name between these two applications is still rather straightforward with string based concatenation and split operations. However, other data can be more complex to map. For example, the relationship between postal codes ('postcode' in Dutch) and city ('woonplaats' in Dutch) cannot be mapped by mere string based operations. Figure 7.3 illustrates the relationship between postal code and city. To map between these entities one will first need to have an external source that relates postal codes and cities.

Bol.com		Tweakers.net	
title	ir.	first name	Kees
first name	Kees	last name	van der Sluijs
infix	van der		
last name	Sluijs		

Table 7.1: Example of two different schemas to store a user name

Also, one needs to take into account the difference in granularity. A postal code relates uniquely to a city, but not the other way around. A city can only be related to a range of postal codes. This means that the mapping process between applications might not be equivalent in terms of reusability in the sense that application 'A' might have more relevant information for application 'B' than the other way around.

Figure 7.3: London postal code illustration

Some information might be semantically equivalent in different application schemas but nevertheless cannot be exchanged. An example of that in Figure 7.2 is the password field. Even though both applications might have this password field, this does not mean that the content of this field can be transferred between applications (next to privacy and security issues) as users might use different passwords for different applications. Another example of such an element would be the choice for an 'optional newsletter' for which it might not hold that if a user agrees to that for one application, he is also interested in that option for another application.

Both examples imply that determining mapping schemes between application will generally contain some manual decision making and labor. This is because the schemas generally have implicit meaning assigned to them by applications (and their designers) that is not externalized and therefore cannot be used.

When one is looking at a scenario in which several applications are expected to exchange user model data between them, e.g. in some integrated framework of (possibly existing) applications, one also need to look at which general exchange approach one could take. We

can use several strategies to support this, for example, the strategy depicted in Figure 7.4 by making mappings to and from every application. The advantage of this approach is that the application is not required to use a specific vocabulary and structure to be interoperable with other applications, and specific custom-made mappings between applications can be built. The disadvantage is the complexity involved in the obligation to have $2N^2$ mapping for `N` applications. This means that for adding one application to the system $2N$ mappings need to be created before being able to fully utilize the exchange of data between all the applications.

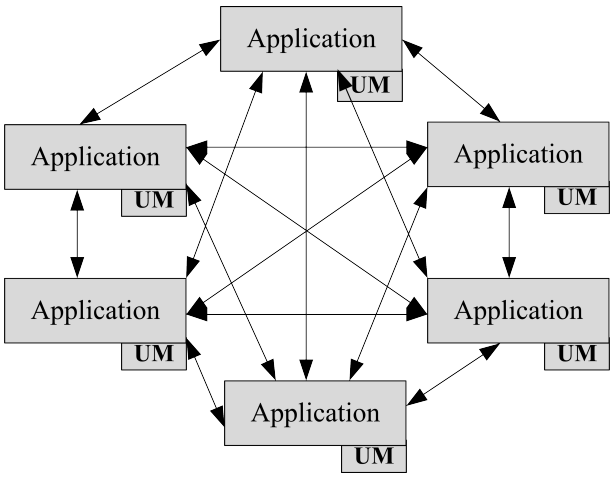

Figure 7.4: Direct mapping strategy

Another strategy is depicted in Figure 7.5. The principle is to create a Shared User Model (S-UM), which contains the most used concepts within the domain. Typically this S-UM could be based on one of the available standard ontologies for UM (like [Heckmann et al., 2005]). By creating a mapping to and from every application and S-UM, S-UM can be used as a mediator for the exchange of user data between the applications. The advantage of this approach is that new applications can be easily added to the system through only two mappings. The complexity is here $2N$ mappings for `N` applications. Disadvantage is that mapping in two steps, via S-UM, might result in loss of information. Moreover, one cannot make specific mappings between applications that take into account subtle details of the two applications that are not reflected in S-UM.

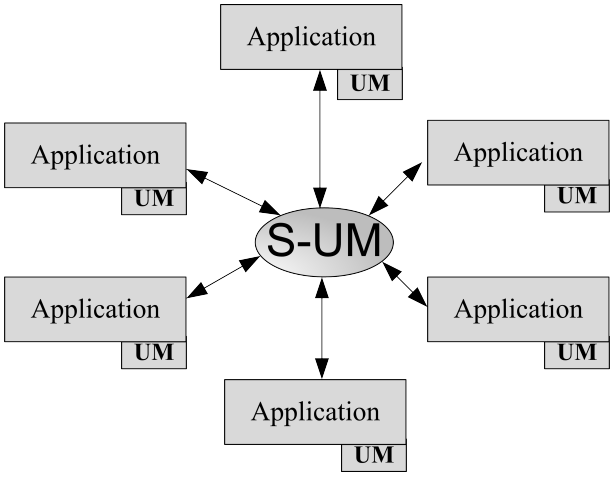

Figure 7.5: Shared UM mapping strategy

The most interesting strategy in most cases in using a hybrid mix between the previous two strategies. Once can use a S-UM for exchanging the most common UM information between the applications, allowing to easily add new applications to the system with a low design complexity. Next to that we allow for direct mappings between applications, such that the need for more custom-made exchange of application-specific information can be fulfilled.

7.2 MobiLife

MobiLife was a European IST-project from 2004 to 2006 targeted on bringing advances in mobile applications and services within the reach of users in their everyday life. The MobiLife consortium consisted of 22 partners throughout Europe and included industrial partners like Nokia, HP, Motorola, Alcatal, Siemens, Suunto and TELIN (Novay[1]). The research area within the project that we focus on was creating a personalization framework that allowed applications running on the phone to do context aware adaptation.

The main idea of this part of the project was that users will use their mobile phones for a growing range of applications. Because of limited ways of user interaction with mobile devices, personalization should help to make mobile application a frustrating experience. As users will have their mobile phones with them all day this will expose them to different situations and different social contexts. This allows the phone to gather and exploit context dependent information. As context and personalization services will be used system wide by mobile phone applications, the MobiLife personalization framework intends to offer a user modeling and context awareness API to all applications running on mobile phones.

Because of the size of the project we will only focus on the part of the project that was done in relation to this thesis. For more information please refer to the deliverables on the Mobilife website[2] or refer to the MobiLife book [Boda et al., 2007].

7.2.1 General architecture

Figure 7.6 shows the overall MobiLife architecture. This architecture consists of several building blocks, namely the User Interface Adaptation Function, Context Awareness Function, Privacy & Trust Function, Personalization Function, Group Awareness Function, Service Usage Function, Service Provisioning Function and Operational Management Function. The names of most of these blocks are rather self-descriptive, but we will give a short explanation for every block below.

The User Interface Adaptation Function Users may use different devices to access mobile applications, even concurrently. Each device supports different input and output modality services, different connectivity, and with different capabilities. To face this problem without creating a specific user interface representation for each device, mobile service developers are supported with user interface adaptation through the User Inferface Adaptation Function. The Gateway Function deals with device discovery and capability management, while the Modality Function specifies the actual service adaptation functionalities.

Context Awareness Function This function takes care of raw, interpreted and aggregated context data. It handles context data related to individual users and to groups

[1] `http://www.novay.nl/`
[2] `www.ist-mobilife.org`

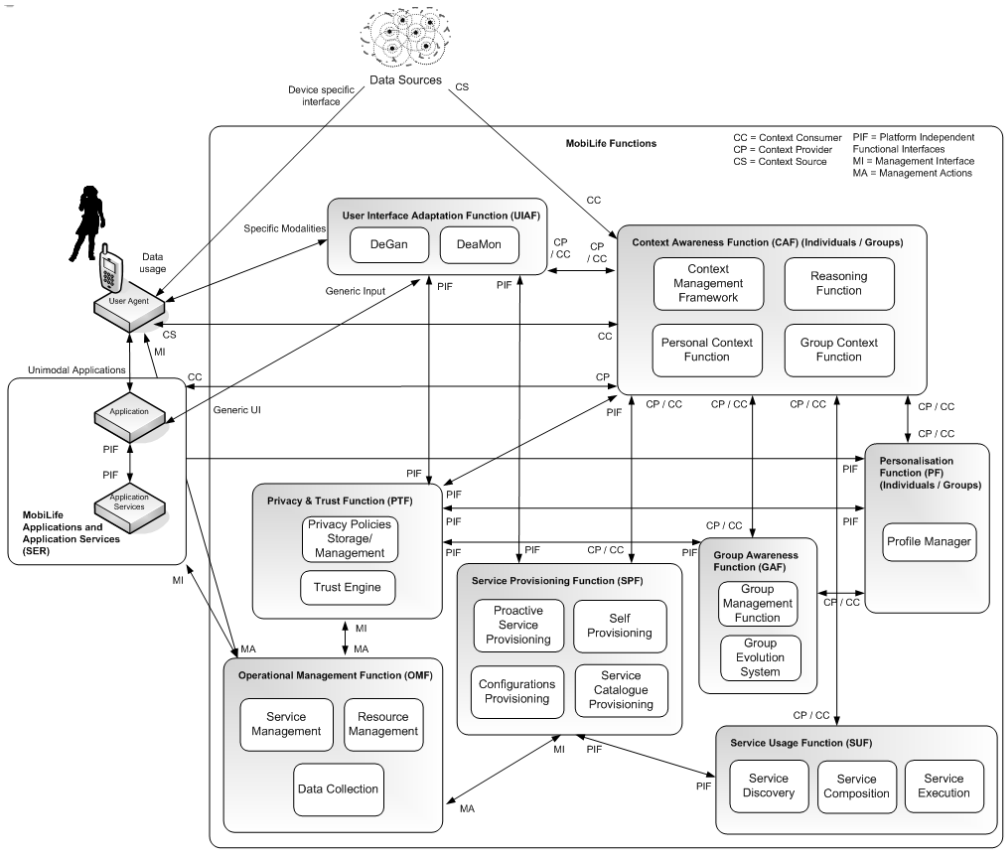

Figure 7.6: MobiLife General Architecture

of users. This module takes care of context descriptors together form a unique context for which separate adaptation actions are needed by using learning techniques. It supports the application developer by providing users' and groups' current context information through well-defined interfaces.

Privacy & Trust Function Mobile services and applications deal with data related to the user, which raises the fundamental issue of trust and privacy of the personal user data. Therefore, the Privacy & Trust Function has been introduced to the Mobile Services Reference Model, ensuring privacy and trust through specifying a Trust Engine and defining privacy policies in Privacy Policies Storage/Management.

Personalization Function This function enables adaptation of mobile services and applications according to personal and group needs and interests. It offers a Profile Manager to manage user and group-related profiles and preferences.

Group Awareness Function The Group Awareness Function is the entrance point for all mobile application services and framework services related to group state information provisioning and to group lifecycle aspects. It provides means for the management of group lifecycle aspects through the Group Management Function as well as the automatic creation and deletion of groups through the Group Evolution System.

Service Usage Function This function covers all aspects related to the service usage. In particular it covers every step in the 'timeframe' between service discovery and

service offering. The Service Provisioning Function covers the components Service Discovery, Service Composition and Service Execution.

Service Provisioning Function This function deals with how services can be pro-actively offered to a user. It provides information about available mobile applications and application services in the Service Catalogue Provisioning. It enables the advertisement of applications automatically or pro-actively to users/groups, without any user/group request or interaction, defined in the Proactive Service Provisioning components. Furthermore, it offers functionality to keep track of a user's service usage behavior and to automatically recommend services to users based on this history data through Self Provisioning functionality. Finally, service configurations can be pushed through Configurations Provisioning.

Operational Management Function This function performs operational management of mobile applications, application services, and related configuration of resources. The Service Management focuses on the knowledge of services (access, connectivity, content, etc.). The Resource Management maintains knowledge of resources (application, computing and network infrastructures) and is responsible for managing all the resources (e.g., networks, IT systems, servers, routers, etc.) utilized to deliver and support services required by or proposed to customers. Data Collection processes interact with the resources to collect administrative, network and information technology events and performance information for distribution to other processes within the enterprise.

MobiLife was envisioned to run on a client-server architecture, i.e. all processor heavy processes would be maintained on dedicated servers while the mobile phone because of limited processing power would only show the user interface. However since enormous advances in mobile processing power have been made, the whole MobiLife infrastructures could nowadays be easily placed on the mobile phone itself. This would reduce security issues involved in transferring personal data and do away with the requirement to maintain internet connection with the mobile phone at all times in order to maintain the full service function.

7.2.2 User Model Context

User identification was done by identifying the user by his phone. It was assumed that mobile phones are only used by one user and that users generally use one phone, so that all user model gathered on a mobile phone could be assumed to be a single user. As every mobile phone has a unique International Mobile Equipment Identifier (IMEI) associated with it, this number can be used as identification of the user as well - or so was assumed for simplification reasons.

Figure 7.7 depicts the general architecture of the MobiLife personalization framework. This framework consists of two parts, the contextualization part called Personal Context Function (PCF) and the personalization part called Personal Profiling Function (PPF). The user exchange strategy in MobiLife was the direct mapping strategy as depicted in Figure 7.4. The reason for choosing this strategy was because it is beforehand unclear what the domain is of the applications developed for the mobile phones. Because these applications can be diverse it will be very hard to come to a common ontology. Also, we did not expect tight integration of most mobile applications, so the number of direct mappings might be reasonably low.

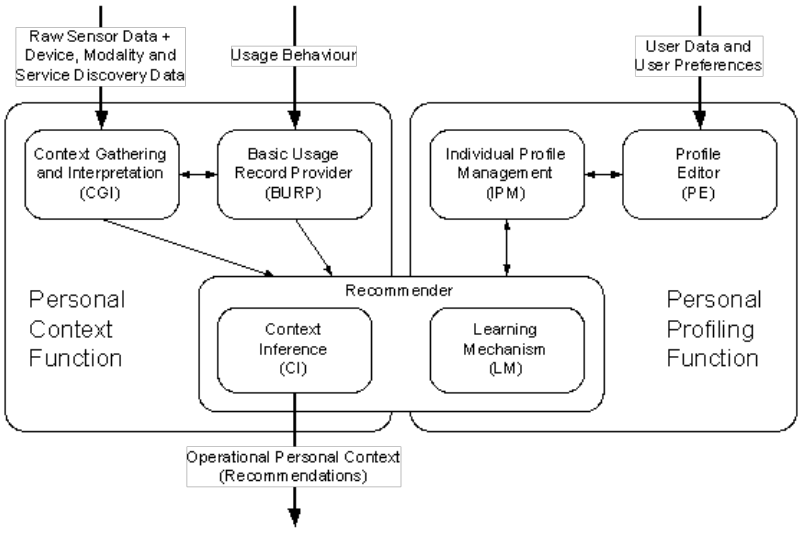

Figure 7.7: Architecture of the PCF and PPF

Context awareness was an important factor in the project. The context awareness module was to process the various phone sensors to determine the current context. These sensors typically are time and location, but could also include heart rate, altitude and depth if the software was for example running on one of Suunto's wrist computers for diving or outdoor training. PCF receives the raw sensor data, data about available services, devices and modalities, and application requests from the Device and Modality Function and other services and applications. The Context Gathering and Interpretation block transforms and grounds all the sensory data streams in context representations and transforms it into instances of the MobiLife context ontology. This is implemented by a rule engine which allows to produce low-level and high-level interpretations of the user's context and situation. One can define rules and labels for high-layer interpretations. For example, one could define 'being in the car' as being in close enough proximity of the bluetooth network of the carkit and 'driving' could be defined as 'being in the car' plus having a speed higher than 20 kilometers per hour calculated from the GPS input stream.

Another function of the PCF is monitoring user behavior in applications. Applications can send user actions to the MobiLife framework using an API. This data can be interpreted by using various machine learning approaches. The structure of this learning mechanism is highly modular and eases the reuse and evaluation of different machine learning methods. Furthermore, the modularity makes it easier to evaluate how well different algorithms perform in different domains. Currently, the system supports rule based reasoning (the same one as for interpreting context data) and tree augmented naive Bayesian classifiers (TAN), by calculating the maximum weight spanning tree using the Chow and Liu algorithm [Chow and Liu, 1968]. The result of this process is an inferred set of user preferences which is stored in the user model.

Another part of the user model is inserted via the PPF. These are the explicit user data and user preferences as provided by the user in the different mobile applications. The application can use the PPF to store this information so that every application doesn't need to build a separate facility for that.

User can see and edit all their data in the different applications using the Mobilife Profile Editor. This includes both the explicit user data and user preferences and the inferred user data, except for data for which this is explicitly prohibited in the schema via a special property. This might be useful for too fine grained data that is useful for the

user, but unintelligible for the user. Figure 7.8 is a screenshot of the pc-version of this profile editor. Given the limitations in screensize and screen capabilities of mobile phones during the MobiLife project we opted not to build a mobile version of the profile editor as the total amount of information stored by the applications and the possible complex structure of the data would be too complex to easily browse on the mobile phone. Instead, the users can still access their data directly in the applications on their phone and us the Mobilife Profile Editor on their pc to review and modify their information if needed. Providing the users the means for exerting the control on the user model data is intended for transparency reasons and might help to improve the understanding of the user about which data is collected and how this can improve the applications.

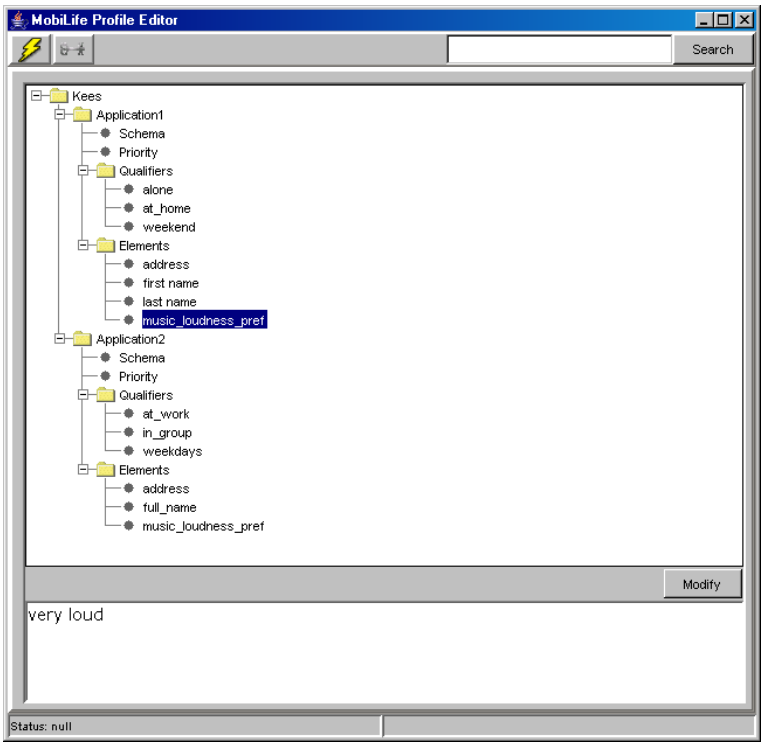

Figure 7.8: Profile Editor

The Mobilife Profile Editor visualizes the user models instances as trees. The RDF format is however intrinsically graph based. Trees are obviously no problem, but the Mobilife Profile Editor is restricted in the sense that cyclic nodes in graphs will not be visualized. Instead every node in the tree that is already visualized elsewhere in the tree is shown in italics to indicate that it is a double and its subnodes will not be visualized. For every phone application we encountered during the Mobilife project this sufficed as these applications typically only stored user data in attribute-value pairs, so typically this is not an issue.

The MobiLife user model is structured as depicted in Figure 7.9. Every user has an associated profile with possibly some personal details. Next to that every profile is associated with zero or more so called profile subsets. Every profile subset is related to a specific application and has a context and schema associated with it. The context is a set of descriptors which describe for which context the profile subset is valid. The empty context is the default profile subset for the user for that application. The schema is an RDF schema of which the user profile data is an instance. The schema information can be used both for validating the user model instance data, but can also be used as basis for schema matching so that applications can exchange user model information e.g. using

the Relco framework (refer to Section 5.5). The actual data is stored in RDF statements under 'UserData'. The 'UserData' can hold any RDF document which the applications might want to store, i.e. it is not necessary to adhere to the schema stored in the Qualifiers. However, if the applications want to check the validity of the 'UserData' as instance of the schema this can be checked. It is upon the application to deal with any errors resulting from this check.

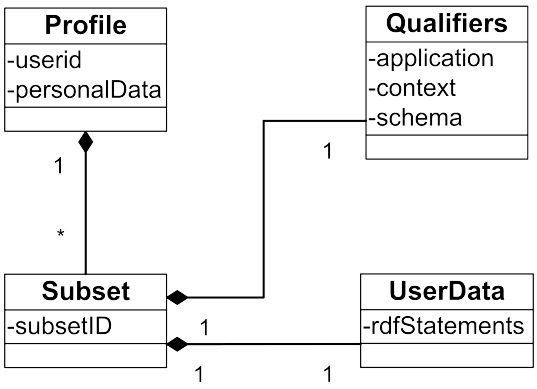

Figure 7.9: MobiLife User Model

Enabling contextual personalization of future services includes a variety of aspects that has been covered by related work. As an example, [Tafazolli, 2006] describes high-level requirements for personalization and profiling enablers. There are also several specifications, which deal with more detailed requirements for user profiles and user profile management such as the 3GPP Generic User Profile (GUP) [Koza, 2004] and the ETSI User Profile Management specification [Petersen et al., 2005]. However, these specifications basically provide high-level requirements but no concrete solutions or implementations. Furthermore, there is also existing work on profile describing schemas like CC/PP [Klyne et al., 2004b] and user model ontologies like GUMO [Heckmann et al., 2005]. However, these approaches lack considering context-dependent user preferences.

7.2.3 Applications

The whole purpose of the MobiLife framework is to support the creation of rich mobile applications. To prove that the context and profile framework is able to support such applications eight of these applications have been developed within the project. As these applications themselves are not themselves part of the work for these thesis, but only give an idea of the possibilities, we will only give a short description of every application:

- **ContextWatcher** is an application that makes it easy and unobtrusive for people to share information about their whereabouts, or any other piece of context information they might want to share, ranging from body data to pictures, local weather information, and even subjective data such as moods and experiences. The goal of the application is to offer people new ways to keep (and stay) in touch without the need for direct interaction. The ContextWatcher can be used to generate and publish personal blogs (screenshot Figure 7.10).

- **Personal Context Monitor** has been designed as a solution to monitor and process in real time essential physiological parameters such as electrocardiogram (ECG) and heartbeat using mobile body measuring devices. This application supports wireless

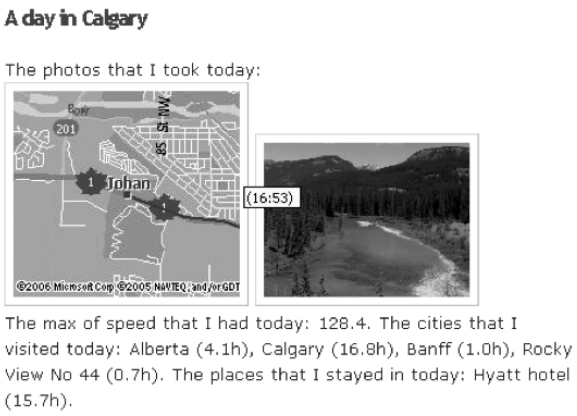

Figure 7.10: A sample blog entry for a day in Canada

real-time transmission of the data stream via Bluetooth all types of listening devices e.g. for reviewing, storage, or e.g. interpretation by a doctor.

- **Proactive Service Portal** provides a simple, but personalized and dynamic service portal. The basic idea of the portal is to relieve the user from browsing through endless index lists to finally find a desired service; or to avoid having the user to type in lengthy URLs with the restricted keyboard capabilities of a mobile device. This functionality is realized by situation-based service recommendation and identification, and proximity-based self-promotion of services.

- **Multimodal Infotainer** is a multi-functional client-server system for use on mobile devices or personal computers. It is able to discover surrounding environment parameters and enables users to watch information from configurable resources through manageable content feeds (news services, blogs and other sources of data). It can also play multimedia content, through multiple devices or modalities, from remote streams or from local files utilizing gathered environment configuration data with the built-in discovery engine.

- **Wellness-Aware Multimodal Gaming System** is a personalized wellness monitoring system that allows monitoring a user's fitness situation and that creates personal training plans according to the current wellness situation of the user (screenshot Figure 7.11). The recommended training plans take into account the user's training habits derived form past trainings, the user's preferences (e.g. the preferred training devices), general user data that might influence the training (such as user's age and weight), as well as expert knowledge from professional gym coaches.

- **FamilyMap** is a location-sensitive messaging system implemented into a map user interface. The main focus of the FamilyMap application is to assist families with small children in their everyday life, especially when on the move. Dealing with location-based, logistical and baby-care related issues came from the realization of difficulties present when one tries to find necessary information from sporadically available information sources while in the city.

- **TimeGems** is a group-centric application that highlights groups as social interaction and communication spaces. The application is designed to provide users with extended interaction possibilities within the various groups they belong to

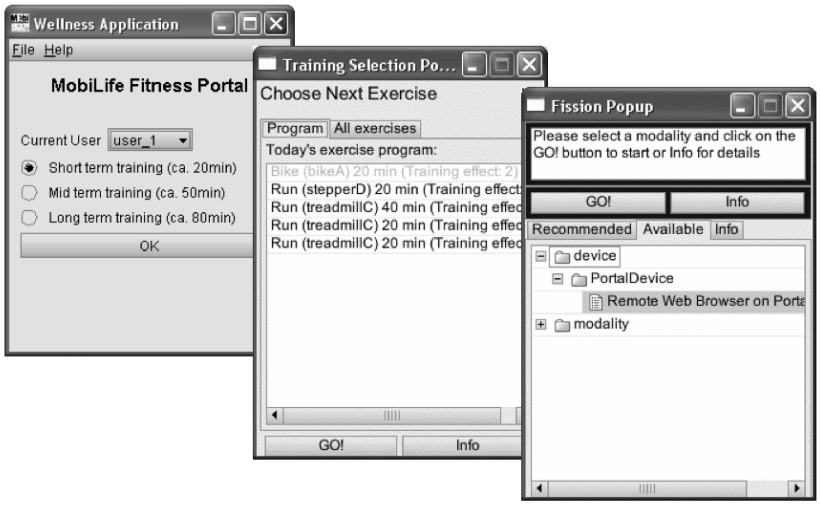

Figure 7.11: Screenshot of the Wellness-Aware Multimodal Gaming System

(e.g., work, family and friends). The application primarily aims to facilitate group activity planning within the users' unused occasional and free-time slots for which a spontaneous organization of group activities is usually very difficult.

- **MobiCar** allows users to plan and set up a group with the objective of sharing a ride in a car that belongs to one of the members, i.e. carpooling.

Every application uses the MobiLife framework and uses and exchanges information with sensory applications and exchange information with other applications. Next to that, some of these applications are social and communicate with nearby machines or people, i.e. the MobiLife context and personalization functions are heavily used.

7.3 Grapple

GRAPPLE[3] was a European FP7[4] STREP[5] Project between 2008 and 2011 [De Bra et al., 2010b]. GRAPPLE involved 15 academic and industrial partners from all over Europe.

GRAPPLE stands for "Generic Responsive Adaptive Personalized Learning Environment". The project's goal was to extend technology-enhanced learning environments like Learning Management System (LMS)s with a beyond the state of the art Adaptive Learning Environment (ALE), in order to provide students with a technology-enhanced learning environment for life-long learning. The following steps were taken within GRAPPLE to reach this goal:

- Extend the state of the art of ALE and build a renewed adaptation framework
- Integrate this new ALE in the main existing LMSs
- Provide interoperability between ALE applications, LMSs and possible interesting external knowledge via an integration framework, including an engine for exchanging user model information

[3] http://grapple-project.org/
[4] http://ec.europa.eu/research/participants/portal/page/fp7_calls
[5] Specific Targeted Research Projects

- Provide additional tooling for laymen course designers (e.g. teachers) like an authoring framework, additional visualization components and prototype extensions of the ALE with a Simulation and Virtual Reality framework
- Evaluate the ALE environment and authoring environment with user studies to study user acceptance and ideas for improvement for future work

The whole project is way too large too completely discuss here, so we will not describe all of these aforementioned points. Instead, we will focus on the work done in relation with this thesis and provide summarizations were necessary. For more detailed information please refer to the GRAPPLE deliverables on the GRAPPLE website[6].

7.3.1 GALE

First goal was to extend the state of the art of ALE. The seminal papers on adaptive hypermedia by Brusilovsky [Brusilovsky, 1996, 2001] have been an inspiration for a lot of research on personalized information access and resulted many adaptive systems as summarized in [Knutov et al., 2009]. The project included authors of many such systems, but the main inspiration for the GRAPPLE ALE was AHA! [De Bra et al., 2006]. For GRAPPLE this resulted in the GRAPPLE adaptive learning environment (GALE). GALE is the spiritual successor of the AHA! system, but it is not just an evolution as the system has been completely redesigned and reprogrammed.

Creating adaptive applications in GALE is at first glance rather similar to creating adaptive applications in Hera-S (chapter 3). GALE needs several input models, including a domain model, a user model and an adaptation model.

The domain model describes concepts that are modeled using an URI that can have properties and relations with other concepts. In essence there are numerous similarities between the GALE domain model and RDF.

A user model in GALE contains domain dependent and domain independent variables. Domain dependent variables are typically extensions of the concept URIs in the domain model (e.g. the user knowledge attribute of a domain concept), while independent variables are modeled via different URIs that do not refer to concepts in the domain model.

The adaptation model describes user model updates based on events, so called event-condition-action (ECA) rules. ECAs are coupled to concepts in the domain. Whenever some concept is requested or viewed or some custom event is specified, this triggers an user model action based on some condition. These conditions and actions may contain arbitrary Java code, making them rather expressive. When some event code is executed this will typically result in user model updates. User models updates can trigger other user model updates, e.g. when an event is set on a user model update, which is called *forward reasoning*.

The actual content are called resources. Resources are related to concepts in the domain model. A concept can have several resources. Within GALE users navigate to concepts and the GALE engine will then select the appropriate resource to show to the user, typically based on the user model. Resources are referred to using URLs and can be either stored on the server running GALE but can also refer to external pages. Users can embed adaptive behavior inside a resource by embedding GALE code that expresses conditional inclusion of objects and fragments like images but also other DM resources. The conditions typically refer to the user model. An example of such conditional inclusion object is:

[6] `http://grapple-project.org/deliverables`

```
<img><gale:attr-variable name="src"
  expr="${gale://gale.tue.nl/personal#isVerbaliser}?
    ${?smallimage}:${?largeimage}"></img>
<object><gale:attr-variable name="data"
  expr="${gale://gale.tue.nl/personal#isVerbaliser}?
    ${?longinfo}:${?shortinfo}"></object>
```

If the "isVerbaliser" attribute is true (the user has the Verbaliser learning style), a small image will be included and a long explanation, whereas users without the Visualiser learning style ("isVerbaliser" is false) will see a large image and only a short description.

Presentation specifics are typically specified in the resources themselves. Resources are (x)html documents that contain markup information or associated CSS files like any other webresource. GALE does provide some additional presentation features through variable processors. These processors can be custom made and can contain directives to create specific views, e.g. defining a layout schema that uses frames. Several views are predefined in GALE. An often used example is the static-tree-view. The static-tree-view creates a hierarchical view of the domain model based on 'parent' relationships. Optionally an integer 'order' attribute can be used to order elements in the tree and a boolean 'hierarchy' attribute can be used to omit elements in the tree. Other examples of views include the next-view for presenting a link to the best suitable next concept for the user and a file-view that displays content e.g. in the domain model.

GALE versus Hera-S

GALE and Hera-S might seem rather similar at first glance. However, there are some large and some more subtler differences. The main difference between the two are its different starting points. Hera-S, as an MDWE design approach, is a data driven approach which means that the content is assumed to be in some RDF or relational datasource. It then provides the means to build the pages of the web applications by refering to the elements in that datasource. GALE, like many adaptive hypermedia approaches, typically allows to remodel an existing non-adaptive Web application into an adaptive one. In other words, given an existing Web application GALE allows to easily extend it and/or transform it with minimal effort.

An example of such a transformed Web application is the business English course shown in the screenshot in Figure 7.12. The business English course is an existing non-adaptive course offered by the BBC[7] which was made adaptive using GALE. It uses the static-tree-view to indicate the available topics and does not only allows adaptivity, but also more freely browsing through the course (i.e. not only linear) based on learning conditions. For more information on this application refer to [van der Sluijs and Höver, 2009a].

There are many other differences between GALE and Hera-S, but more interesting are the striking similarities. Both approaches allow defining full-fledged adaptive and personalized applications for the user. For example, note that both approaches allow designers to design their application from scratch and in that respect it is a matter of taste which approach one could take best. Besides the different approaches, one could say that GALE might be more powerful (e.g in using custom Java code in expressions) while Hera-S might be more flexible in using external and existing domain models and externalizing all data including the application and user models.

[7] http://www.bbc.co.uk/worldservice/learningenglish/business/tendays/index.shtml

Figure 7.12: The adaptive business English course

One of the consequences of the similarity between GALE and Hera-S is that as part of the interoperability and standards focus within the project it was decided to look into interoperability between the many existing adaptive frameworks. Part of the project was to enable exchange of information between LMSs, e.g. for the scenario of cooperating universities or universities that use different LMSs. Exchanging user model data for example allows students to seamlessly follow courses at several institutions, while keeping track of their global progress. A similar case could be serving the same course at different institutions that might use different adaptive frameworks. By defining a standard format for all models necessary to define an adaptive course we envisioned we could enable this interoperability between adaptive systems.

We conceived the information flow for GRAPPLE as visualized in Figure 7.13. The output of the authoring process is the so-called Conceptual Adaptation Model (CAM). The CAM contains all models necessary to define an application, but also includes information like canvas shapes, position, size, and other tool specific information, and it does not contain the actual resources. The CAM is sent to the Engine Specific Compiler via the communication channel called GRAPPLE Event Bus (GEB) which compiles this into the GALE models which allows the GALE engine to run the course. The GALE engine then can communicate via the GEB with a Engine Independent Compiler and send the entire course (i.e. all models and adapted content). This compiler then should be capable to transform this data into an engine-independent language that can be communicated between engines. The engine-independent language was called Generic Adaptation Language (GAL). This Engine Independent Compiler should also be able to retranslate from the GAL language to GALE specific models.

Our foremost goal therefore was to come up with an engine-independent language. We chose to base GAL on the Hera-S models. This effort is further detailed in Section 7.3.1.

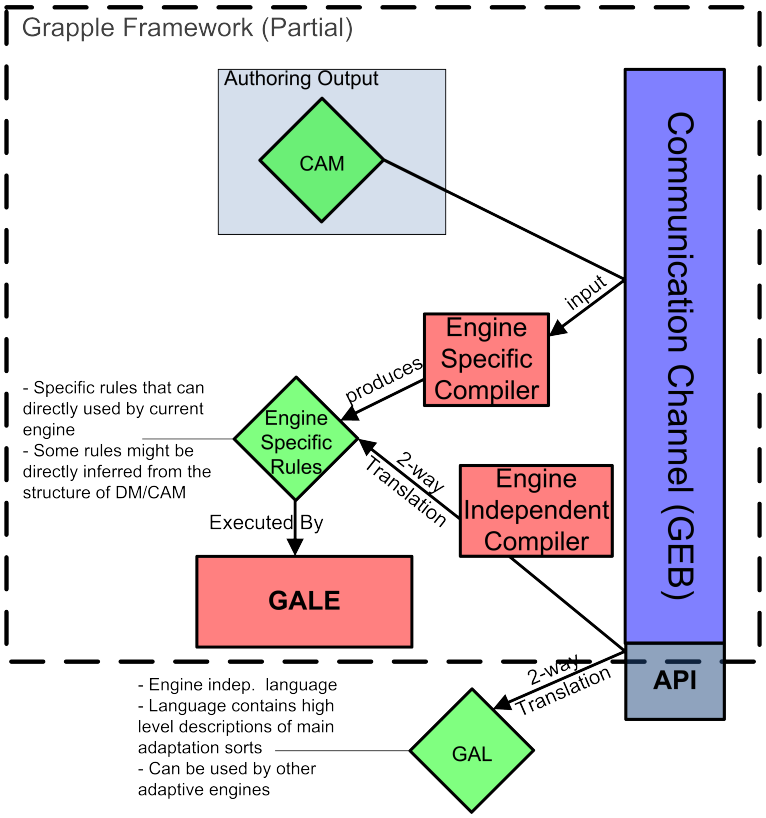

Figure 7.13: GRAPPLE Framework Information Flow

7.3.2 Overall Architecture

The GRAPPLE architecture (shown in Figure 7.14) consists of the following functional blocks:

- Integration with existing LMSs that manage the learning process, progress and support. The actual LMSs with which GRAPPLE integrated were well-known open source systems like Moodle[8], Claroline[9] and Sakai[10], but also proprietary systems aimed at corporate learning applications that were realized by GRAPPLE's industry partners.
- GEB, a generic Event Bus that facilitates an asynchronous information exchange between all GRAPPLE components, but can also be used for external communication.
- GALE for the adaptive delivery of learning material, as described in Section 7.3.1.
- Generic User Model Framework (GUMF) that maintains information about each learner as he uses different LMSs, GALE instances or other software.
- A single sign-on facility (based on Shibboleth[11]) for user authentication that ensures user identification within all components connected to the GRAPPLE framework.
- The GRAPPLE authoring tools (GAT), which provides authors and course designers (e.g. teachers) to visually create adaptive courses that can be deployed in GALE.

[8] http://moodle.org/
[9] http://www.claroline.net/
[10] http://sakaiproject.org/
[11] shibboleth.internet2.edu/

- GVIS, a visualization framework that can aggregate and visualize information from GUMF. GVIS is also integrated with every LMS.

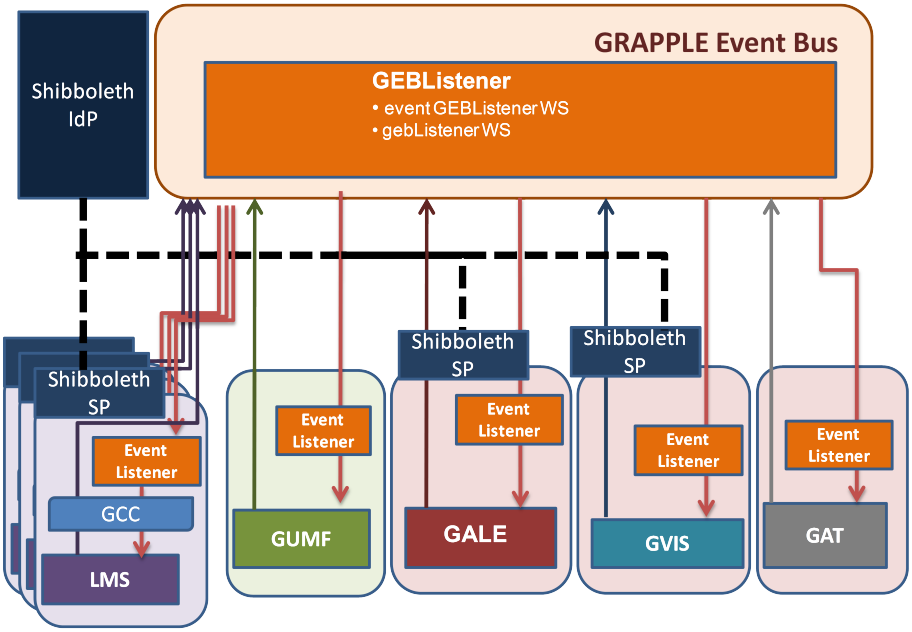

Figure 7.14: GRAPPLE Architecture

Most interesting in terms of interoperability are the GEB and GUMF.

GEB facilitates communication within the GRAPPLE framework. Applications (every internal GRAPPLE component or GRAPPLE external applications) that connect with the Grapple Framework subscribe via the Event Bus webservice. Every application can set a list of event types in which it is interested and every application can post events of a specified type to the Event Bus. If an application posts an event to the Event Bus it will send this to all listening applications that specified interest in that type of event. Every event contains the following components:

- *eventId*, a unique identifier of events of type int. The eventId is generated by GEB. The functionality of this ID, is to provide the possibility to identify the messages exchanged between the components through the GRAPPLE Event Bus.

- *previousIdEvent*, a unique identifier of type int. It is an optional attribute. In this case, this ID is provided by the component which made the request. This ID is the identification, if the request has any relation with other event.

- *methodName*, the type of event. It is used by the Event Bus to decide to which listening components an event should be sent.

- *body*, contains the actual event message that is interpreted by the components that receive the event. The content of the body is not processed by GEB but only passed on to the listeners.

For example, if the LMSs Moodle and Sakai both have an interest in course updates from their students they can both subscribe to the "umChanged" event. When a user model update then takes place in GUMF, GUMF can then send a "umChanged" event to GEB, reporting for which students UM updates have taken place. GEB takes care of

sending around the "umChanged" events to all listening applications. In this way, GUMF does not need to keep track nor hardcode which applications might be listening and might need notifications. The listening application can then process the event and if needed send an event-reply.

The GRAPPLE event bus abstracts from the need that every GRAPPLE component needs to be exactly aware about which LMS or other application accesses it. The Event Bus also simplifies the process of adding new components to the framework and facilitates distribution of components as the components might have instances running at several locations. The connecting applications only need to provide an event listener interface to GUMF that GEB can send the messages to.

GEB is an asynchronous service, meaning that components that send messages do not have the facility to wait for an answer or know for certain if any more answers might come or not. For some applications this might be an issue, especially for components that run in the client browser as these cannot run web service listeners. Therefore, for these applications (e.g. GAT) a proxy webservice was created that can be used to access the GEB through and which can temporary store replies, which allows the (browser) client applications to periodically check for replies.

GEB is an open platform that allows any component to subscribe and listen to events. This can of course be a privacy concern. And even though security measures have not been implemented it would not be to hard to extend with authentication measures and encryption to increase security.

7.3.3 GUMF

The GUMF facilitates exchange of user data between GALE, LMS and any other application that wants to exchange user data. Examples of what might be exchanged between GALE and LMS are for instance a measure of progress with the adaptive course from GALE to the LMS and the results for a specific test in the LMS to GALE, which allows blending LMS capabilities like taking tests with GALE's adaptive course. Exchange between GALE applications allows reuse of data between similar domains, e.g. we could estimate from a regular English course what a users level is when starting the business English course.

To be an effective framework for exchanging user data between applications, GUMF has to somehow deal with user identification, context and semantic heterogeneity as detailed in sections 7.1.1 and 7.1.2.

In GUMF users are identified by a unique URI, i.e. a URI uniquely identifies a user. Users can have different URIs in different applications, though. GUMF explicitly does not solve this issue, but leaves this to the applications that use GUMF. This means that applications that wants to use the information of one of their users in another application somehow need to know the URI used by that other applications to identify the user in question. Within the GRAPPLE framework this is covered by using the Shibboleth authentication mechanism. In this way, GALE and every LMS connected within the GRAPPLE framework will use the same user identifier. In other scenarios applications will need to find other ways to establish the user identity, e.g. by once requesting users to provide the login details of the target application. In this scenario user involvement is typically also desired because of privacy and scrutiny reasons. Leaving this issue to the applications on one hand burdens the application with solving this issue, on the other hand it also allows applications to use specialized user authentication mechanisms that typically are already used in most institutions and companies anyway. The choice in this

case is for flexibility instead of additional (possibly redundant) service.

Contexts in GUMF can be covered by so-called data spaces. A data space is specified as a logical bundle of data (instances) as well as ontologies and rules (schema). Data spaces thus go beyond the notion of namespaces as they explicitly allow us to specify a set of things that span a certain data space, on which an operation (e.g. query operation or reasoning operation) should be executed. In more detail, such data spaces represent the part of GUMF that a certain client application is managing and responsible for, i.e., its own workspace. However, in typical use an application has only one dataspace in GRAPPLE. This is because in the educational domain is a bit different from typical consumer applications, as it is not the user preferences in relation to the context that is the most important, but the actual knowledge and progress of the user as a student is. In this case context seem to be less important. This does not mean that contexts cannot separate user model data context, technically data spaces could perfectly cover this, but in practice this is typically less important in the educational domain.

Semantic heterogeneity is dealt with in several ways within GRAPPLE and in GUMF in particular. First of all as the existing LMSs are rather similar in nature it was decided to come to standardize input and output communication via the existing IMS-LIP standard [IMS Global Learning Consortium, 2001]. The user information currently interesting for the LMS could be encoded using this standards. GALE supports arbitrary complex user models and also can embed ECAs which cannot be captured by IMS-LIP. So GALE user model data is not translated into IMS-LIP.

GUMF internally models UM data by means of Grapple statements. A Grapple statement is basically a packaged RDF statement with some fixed additional metadata. Some of this data should be provided by the applications that use GUMF, e.g. the target data space, the user whom the information is about, the time of observation of the information and the application that is the source of the statement. Some other elements are added by the GUMF framework itself like id, storage date, originating URL, etc. This user model metadata allows richer queries over the data.

As IMS-LIP has a different syntax and structure than Grapple statments, the IMS-LIP data needs to be converted from and to Grapple statements when stored in GUMF. This process is detailed in Figure 7.15. The IMS-LIP is sent to GUMF via the GEB and arrives at the GUMF listener. This listener recognizes the IMS-LIP datastructure and it calls the converter which maps the IMS-LIP data to Grapple statements. Then it can be stored in GUMF. In order to exchange user data between LMSs, these LMSs only have communicate the data space address that stores the user information from the other LMS to retrieve their user model data (i.e. by configuration).

GALE transforms its data directly into Grapple statements, i.e. by one of its processors. As domain and user model data resemble RDF in structure this is a straightforward processor.

The most interesting transformation within the GUMF engine is between the differently modeled data, like between GALE and the LMSs, or other possible applications. To deal with this, GUMF has so-called Grapple Derivation Rules (GDR), which is a rule-based mapping language that is inspired by both the SWRL based mapping system which was described in Section 5.4.3 and the SPARQL based mapping system which was described in Section 5.4.4. GDR allow to map and aggregate values from the data stored in one data space into the structure of another data space.

GDR have premises and consequents. For example, the following GDR rule instructs the multiply operator to increase the knowledge level of a student for the concept "gale://..HotelCheckin" if the student completes a quiz (with ID 118316) that tests the

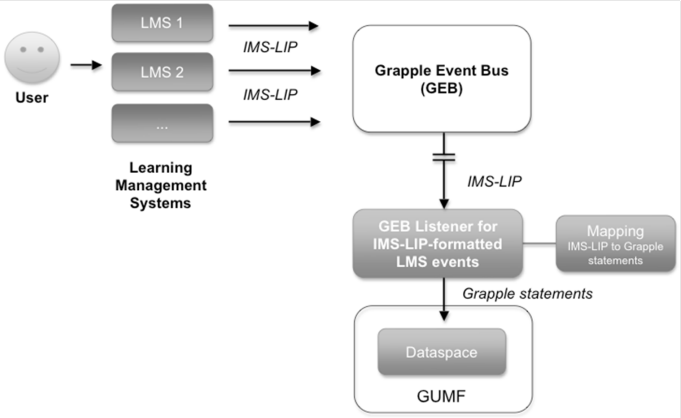

Figure 7.15: GUMF Information Flow

student's knowledge in conversing in English when checking into hotels. The rule simply multiplies the current knowledge level for these concepts by 1.5, i.e. new knowledge level of 'HotelCheckin' is equal to the previous knowledge level multiplied by 1.5:

```
<gdr:rule name="Quiz Level to External Knowledge (CLIX)"
        description="Maps quiz result of a CLIX quiz to gale attribute"
        creator="http://pcwin530:8080/gumf/client/1">
   <gdr:premise dataspace="1">
      <gc:subject>?user</gc:subject>
      <gc:predicate rdf:resource="http://grapple/ims-lip/completedTest"/>
      <gc:object>118316</gc:object>
      <gc:level>?level</gc:level>
   </gdr:premise>
   <gdr:consequent dataspace="1">
      <gc:subject>?user</gc:subject>
      <gc:predicate rdf:resource="http://gale/predicate/knowledge"/>
      <gc:object>gale://gale/BusinessEnglishCAM/HotelCheckin</gc:object>
      <gc:level>op:multiply(?level,1.5)</gc:level>
   </gdr:consequent>
</gdr:rule>
```

This rule shows how properties between an LMS and GALE can be exchanged and how actual values can be manipulated using data operators. For this purpose several operators are available (e.g. multiply, divide, add, substract). These can be easily extended with more. The GDR are executed via a custom built rule engine within GUMF, which internally uses sesame and SPARQL queries and custom transformation code.

Obviously there is a learning curve involved in creating these mapping rules. However, these rules need to be created only once by a specialist between two applications. Currently, there is some basic tool support for creating these rules in GAT. However, this tool support can be easily extended using Relco to find matching elements between user models and a basic facility which should be able to point and click these relations between user models to generate a GDR copy rule. This would only necessitate more specialistic knowledge of these rules for more complex situations.

7.4 GAL

As noted in Section 7.3.1, we are interested in an engine-independent adaptation language. We named this language GAL and based it on the Hera-S AM (refer to Section 3.4). This is because the Hera-S models seems to be a fine grained way to model both the content and adaptation. Our hypothesis is that if we are able to show that GAL is able to perform all types of adaptation as defined by Brusilovsky [Brusilovsky, 1996], it should be able to define adaptation as can be expressed by most existing adaptive hypermedia systems. Furthermore, we reasoned that a formal definition of the GAL language is an important requirement if it is to be used an engine-independent standard language for adaptation that can be used to exchange applications between e.g. AHA!, GALE and others.

7.4.1 Extending GAL

GAL is basically defined as the Hera-S AM model. However, first we will extend the AM with two language constructs that in Hera-S are more typically considered to be part of presentation considerations, but are deemed important in terms of adaptation - as some important types of adaptation include presentation adaptation.

These two constructs are 'Emphasis' and 'Order'.

GAL Components can be emphasized by giving them the property

```
:hasEmphasis "high"
```

We discern four types of emphasis:

- *low emphasis* means under-emphasis. A component with low emphasis should get less emphasis than normal objects.
- *default emphasis* is as if no hasEmphasis property has been defined, i.e. the default for all GAL Components.
- *emphasis* means more emphasis than normal (no emphasis). This level allows the adaptive engine to differentiate between more important objects ("normal" emphasis) and most important objects ("high" emphasis).
- *high emphasis* is the most emphasis a GAL Component can have.

Emphasis is a construct that directly influences the presentation. Even though we mainly target adaptive navigation, emphasizing components (especially links) on a page can strongly influence the navigation behavior of the user and it is a regular adaptive construction. Therefore, it was chosen to include this construct in GAL. It is up to the engine to decide how to present an emphasized GAL Component, e.g. by highlighting. Emphasis for a NavigationUnit means emphasis for every element in that unit.

Order is also an important aspect of adaptive navigation. Moreover, it might be important for the internal logical structure of a page. GAL Components can have an order element that specifies an ordering. The value of an ordering element is either an integer, or a (conditional) query resulting in an integer. For example we can give an attribute "Conversation" an order element with value 1 as follows:

```
:CheckingIn :hasAttribute [
    :name Conversation;
    :order 1
]
```

The order element dictates a partial order in GAL Components within the scope of a particular NavigationUnit. This means that if two or more GAL Components have an order attribute with an integer value, the subcomponent with the lowest value is guaranteed to be displayed before the subcomponent with a higher ordering value. Subcomponents with the same ordering value are considered relatively unordered, meaning that there is no guarantee which of these subcomponent is displayed before the other. The ordering of GAL components that have non-integral values or missing ordering properties are undefined. In general we do not assume any particular order based on the order of GAL code. However, we do allow specifying to strictly follow the GAL code order within a GAL Component by defining a default-ordering property for that unit by simple declaring:

```
:default-ordering true
```

Note that using an explicit order attribute takes precedence over the default ordering.

For set results (e.g. for the SetUnit construct) we automatically assume the default-ordering to be true, unless specified otherwise.

7.4.2 Types of Adaptation

Our basic GAL constructs thus far expressed adaptation on a very basic level. This allows for very generic types of adaptation. However, we also consider more specific adaptation technologies as proposed in Brusilovsky's adaptation taxonomy [Brusilovsky, 1996], which gives us the following adaptation techniques:

- Adaptive multimedia presentation
- Adaptive text presentation
- Direct guidance
- Link sorting
- Link hiding
- Link annotation
- Map adaptation

We review these techniques and present how GAL can be used to apply these techniques and we also provide some constructs that help to apply some of the techniques and we explain how these additional constructs can be translated into basic GAL constructs. In this way we show that GAL supports the basic types of adaptation as recognized in the field of adaptive systems. Along the way, we will introduce shorthand GAL constructs that can directly express the different adaptation types. We will also provide how these shorthand constructs can be modeled with basic GAL constructs.

Adaptive Multimedia Presentation

Usually in adaptive systems Adaptive Multimedia Presentations means personalized media selection, i.e. given that there is a choice of a number of multimedia clips the system chooses the one that fits best for the user.

A selection would look like this:

```
:select  [
:case [ :casenumber "1";
            :condition [Query1];
            :hasAttribute [
              ..details of multimedia attribute 1..]
    ] ;
...
:case [ :casenumber "N";
            :condition [QueryN];
            :hasAttribute [
              ..details of multimedia attribute N..]
    ] ;
:case [ :casedefault "true";
            :hasAttribute [
              ..details of multimedia attribute in the default case..]
    ] .
].
```

The semantics of this statement is that the first case that matches its corresponding condition will be shown and otherwise the default case will be shown. One way of translating this in terms of GAL if-statements of one attribute with different values is as follows:

```
:hasAttribute [
  ..some general details of the attribute..
  :if [Query1] ;
  :then [..query of multimedia attribute 1..];
  :else[ ... :if [QueryN];
       :then [..query of multimedia attribute N..];
           :else [..query of default multimedia attribute..] .
   ] .
] .
```

Direct adaptation of multimedia objects cannot be done directly in GAL (except if they can be expressed in RDF). However, if a Web Service exists that allows adaptation of a multimedia object this can be supported in GAL via the Web Service construct. Personalization can than be done. GAL can send variable parameters to the Web Service call. However, the Web Service should send a result that can be handled within the GAL language, e.g. text or an URL.

Adaptive Text Presentation

In GAL an attribute can be a piece of text as well as any other type of media (identified by a URL). So the choice of construction in the previous paragraph also applies to text. This means that alternative page fragments can be selected. Moreover, general text adaptation can be done using the (conditional) query mechanism in GAL that can refer to both domain concepts and the UM. We offer a specific construct to weave different versions of a document together. This allows to quickly select a group of items to be shown based on a condition.

At the start of a Unit for which we want to define different versions together we declare something like

```
:alternative [
:case [ :casenumber "1";
            :condition [Query1];
            :selectid "alternative1"
    ] ;
    ...
:case [ :casenumber "N";
            :condition [QueryN];
            :selectid "alternativeN"
    ] ;
:case [ :casedefault "true";
            :selectid "default"
    ] .
] .
```

Based on this selection criterion we now define in the rest of the unit which GAL sub-Components belong to which selection id by adding the :selectid property with the appropriate name, for example:

```
:has Attribute [
   :label "Conversation Option:";
   :value [query];
   :selectid "alternativeN"
]
```

In this way all GAL Components with selected "alternative" will be initialized, while all attributes that belong to one of the other alternatives are not initialized. This can be done for all GAL Components that accept queries. This alternative construction can be used to prevent repeating selection queries and to get a better overview for which page-alternative a construct is used. The semantics of this construction is that all GAL Components with a specific selectid will be shown only if it's the first case in the alternative statement for which the condition is satisfied.

The translation into GAL constructs (internally) is done by repeating the conditions for every GAL component to check if the condition for the specific GAL component is the first that satisfies the condition of the case statement.

Direct Guidance

Direct Guidance means that the system decides what the best next link for a user is. In order to support this we use a slightly adapted version of the 'alternative' construction above, namely

```
:emphasize  [
:case [ :casenumber "1";
            :condition [Query1];
            :emphasisid "emphasis1";
            :hasEmphasis "high"
    ] ;
...
    :case [ :casenumber "N";
```

```
            :condition [QueryN];
            : emphasisid "emphasisN"
    ] ;
:case [ :casedefault "true";
            : emphasisid "default"
    ] .
] .
```

So instead of a selectid property we use an emphasisid, where the id points to GAL Components. Typically for Direct Guidance we would only point to links, but this construct can also be used to emphasize other GAL Components other than links.

Translation in GAL terms is slightly different here, because conditions cannot be attached to hasEmphasis properties, but only to query elements. Instead we generate two versions of the same GAL Component that is referred in the emphasisid clause, an emphasized and a non-emphasized one, and based on conditions we choose one of them.

Link Sorting

Adaptive link sorting is already directly supported in GAL via the SetUnit in combination with ordering. This assumes that the query language supports ordering. By referring in the query to the user model (as well as the DM) allows to retrieve a set of links that depend on the user model.

Link Hiding

Link hiding (as well as hiding of other GAL Components) is also directly supported in GAL via conditions. By not specifying a then or else part in the condition means that a link (or other component) is not initiated and thus effectively hidden.

If the GAL Component should not be really hidden but just displayed differently (e.g. grayed out) the emphasis level “low” can be used.

Link Annotation

Link annotation means that there is some form of adaptive comment or presentational makeup, to give visual cues to the user. We do not specify direct presentational aspects of link annotation in GAL, as this is an engine specific presentation detail. However, what is necessary from GAL perspective is that user information can be updated and used to adapt different aspects of a link. This is possible in GAL because the value of the properties in a link can be determined via queries in combination with adaptive conditions. For example, given GALE, we could adaptively determine the class of an attribute or a link. GALE then is responsible for the concrete different forms of annotation of links.

Map Adaptation

Map adaptation is used to for instance generate menu structures. One typical type of map adaptation is a hierarchical structure of links. This can be defined in a hierarchical construction as follows:

```
:hasHierarchy [
    :item [ :itemid "ch1";
            :subitem :root;
```

```
            :hasLink [Link specification ch1] .
    ] ;
    :item [ :itemid "ch1.1";
            :subitem "ch1";
            :hasLink [Link specification ch1.1] .
    ] ;
    :item [ :itemid "ch1.1.1";
            :subitem "ch1.1";
            :hasLink [Link specification ch1.1.1] .
    ] ;
....
    :item [ :itemid "chN";
            :subitem :root;
            :hasLink [Link specification chN] .
    ] .
] .
```

This can be translated in GAL terms by using the hierarchical structures of Units in combination with the Order attribute. For example consider the following (partial) translation:

```
:hasUnit [
   :hasNavigationRelationship
   [  :order 1;
       {..Further Link specification ch1..}
   ] ;
   :hasUnit[ :order 2;
   {..Ref SubUnit that specifies ch1.1 and ch1.1.1..}
   ] ;
....
   :hasNavigationRelationship
   [ :order N;
      {..Further Link specification chN..}
   ] .
] .
```

7.4.3 Formal Specification

Here we provide a first formal specification of the GAL language. This is not a complete and strict formalization, just a first step in that direction. Also note that some of the following constructs have been named and specified slightly differently from the AM definition in Section 3.4, for example we changed setUnits into listUnits as these last are simpler to define. All these changes are introduced to simplify the formalization. It depends on the resulting implementation of the language which constructs are easier and preferable to use. Based on this experience we can either choose to use the changed construct, or change the formalization (and complicate that a bit in the process).

Basic constructs

The purpose of GAL is to describe the navigational structure of a web application and how this adapts itself to the actions of the users. The central concept is that of NavigationUnit

which is an abstract representation of a page as it is shown to a certain user after a certain request. Basically it is a hierarchical structure that contains the content and links that are to be shown, and also updates that are to be executed when the user performs certain actions such as requesting the page or leaving the page. All data from which these units are generated is assumed to be accessible via a single RDF query endpoint, and the updates are also applied via this endpoint.

Preliminary definitions and notation

For the purpose of formalizing the syntax and semantics of GAL we postulate the following sets and constants: `ET` (event types), `DB` (database instances, which are queried and updated through the RDF query endpoint), `VQ` (value queries over the RDF query endpoint that retrieve a single value), `LQ` (list queries that retrieve lists of bindings from the RDF query endpoint), `UQ` (update queries that update the RDF query endpoint), `A` (attribute names), `V` (values, which includes RDF literals and URIs), `EL` (element labels), `UT` (unit types, which we here assume to be a URI consisting of a namespace and a local identifier), `EN` (element names), `X` (variable names containing a distinguished variable **user**), `DC` (domain concepts). We use the special constant $\perp$ which is not in any of the postulated sets, and the special constants **"gal:_top"**, **"gal:_self"** and **"gal:_parent"** which are not in `UL`. We define the set `B` to be the set of bindings, i.e., partial functions $b : X \rightharpoonup V$ that map variables to value. For partial and total functions b_1 and b_2 we define their combination as $b_1 \# b_2$ such that $(b_1 \# b_2)(x) = b_1(x)$ if $b_1(x)$ is defined and $b_2(x)$ is undefined, $(b_1 \# b_2)(x) = b_2(x)$ if $b_2(x)$ is defined and $(b_1 \# b_2)(x)$ is undefined otherwise. We generalize the # operator such that $(b_1 \# \ldots \# b_n) = (b_1 \# \ldots \# b_{n-1}) \# b_n$ for $n > 0$ and $(b_1 \# \ldots \# b_n) = \emptyset$ for $n = 0$.

Additionally we introduce the following notation: $\mathcal{L}(S)$ the set of finite lists with elements from `S`. Lists are denoted as $[1, 1, 2]$, and list concatenation as $L_1 \cdot L_2$. We write $x \in L$ to denote that an element `x` occurs in list `L`. We use list-comprehension syntax, such as $[f(x, y) \mid x \in L_1, y \in L_2]$ to construct lists and bags with the usual semantics.

For queries we assume that they contain variables, which can be replaced with a value. Concretely we assume the queries are expressed in SPARQL where variables are used in the graph patterns and are preceded with "?" or "$". For such queries `q` we let $q[b]$ denote the query that is obtained when the variables in `q` for which `b` is defined are replaced with their value in `b`. Strictly speaking for SPARQL queries this can result in an invalid query if variables in the SELECT clause are replaced, but we ignore this for the moment. We assume that for all queries `q` a semantics is given, viz., $[\![q]\!] : DB \to V$ for value queries $q \in VQ$, $[\![q]\!] : DB \to \mathcal{L}(B)$ for list queries $q \in LQ$ and $[\![q]\!] : DB \to DB$ for update queries. We also assume that it is stored in the database which values denote an instance of a domain concept. For a database instance $db \in DB$ and domain concept $dc \in DC$the set of instances of `dc` is given by a set $[\![dc]\!]_{db} \subseteq V$. Concretely we will assume this set is retrieved by the SPARQL query `SELECT ?v WHERE { ?v a dc } .`

We now define the syntax of value expressions. We assume that the following non-terminals already are defined: <value> describes `V`, <val-query> describes `VQ`, <list-query> describes `LQ`.

<val-expr> ::= <value> | **"$"**<var-name> | **"("**<val-expr>**"."**<val-expr>**")"** |
"[" **"gal:query"**<val-query>**"]"** |
"[" **"gal:if"**<list-query> **";"** (**"gal:then"**<val-expr>**";"**)? (**"gal:else"**<val-expr>**";"**)?**"]"**.

The set of all value expressions, i.e., those belonging to <val-expr>, is denoted as `VE`. A value expression binding is a partial function $be : X \rightharpoonup VE$.

The definition of units and content elements

We first define the set `NR` of navigation relationships as they are represented in a navigational unit. It contains exactly all tuples of the form $\mathbf{ln}(ut, b, q, be, t)$ with a unit type $ut \in UT$, a value binding $b \in X \rightharpoonup V$, a query $q \in LQ \cup \{\bot\}$, a value expression binding $be : X \rightharpoonup VE$ and a link target $t \in N \cup \{\textbf{“gal:_top”}, \textbf{“gal:_self”}, \textbf{“gal:_parent”}\}$.

The intuition of the definition of navigation links is as follows. The unit type `ut` indicates to what type of unit the link will lead, the value binding `b` specifies the fixed link parameters that are passed with the page request, the query `q` computes a binding that defines a set of additional parameters which are computed at the time the link is followed, the value expression binding `be` specifies even more additional parameters that are computed at navigation time by given value expressions for individual variables and finally the target `t` indicates which part of the current page will be replaced by the result of the request that is generated by navigating the link.

We define the sets `U` of *proper units* and `E` of *content elements* as the smallest sets such that (a) `U` contains

1. all *units*, i.e., tuples $\mathbf{un}(C, h, nl)$ where $C \in \mathcal{L}(E)$ is a finite list of content elements, a relation $h \subseteq ET \times UQ$ and the navigation relationship $nr \in NL \cup \{\bot\}$,

and (b) that `E` contains

1. all *attributes*, i.e, pairs $\mathbf{at}(n, l, v)$ with name $n \in EN \cup \{\bot\}$, label $l \in EL \cup \{\bot\}$ and value $v \in V$,
2. all *unit containers*, i.e., tuples $\mathbf{uc}(n, l, u)$ with name $n \in EN \cup \{\bot\}$, label $l \in EL \cup \{\bot\}$, and unit $u \in U$,
3. all *unit lists*, i.e., tuples $\mathbf{ul}(l, UL)$ with label $l \in EL \cup \{\bot\}$, and unit list $UL \in \mathcal{L}(U)$.

Note that since `U` and `E` are defined as the smallest sets that satisfy the properties (a) and (b), the mutual recursion is finite, i.e., the trees that are defined by mutually nested units and content elements are of finite height.

The intuition behind the concepts of units and content elements is as follows. A unit consists of (1) its content `C`, (2) the relation `h` that indicates which update queries are associate with which event type, i.e., are executed when these events occur, and (3) the navigation relationship `nr` that will be associated with the representation of the unit.

The intuition behind the definition of content elements is as follows. Attributes represent basic content and consist of (1) the name `n`, which is mostly there to indicate the meaning of the attributes, (2) the label `l`, which will be used in the presentation on the page and (3) the value `v`, which represents the content of the attribute. Unit containers represent nested units and consist of (1) the name `n`, which is used to identify the particular position of this nested unit, (2) the label `l`, which is used in the representation of the nested unit and (3) the nested unit `u`. The unit sets and unit lists represent a nested collection of units and consist of (1) a label `l`, which is used the representation of these collections, and (2) the bag or list of units that is the collection.

Syntax of the Language

We assume that the following non-terminals already are defined: <unit-type-name> describes `UT`, <var-name> describes `X`, <domain-concept> describes `DC`, <literal> describes

RDF literals, <val-query> describes `VQ`, <list-query> describes `LQ`, <update-query> describes `UQ`, <unit-label> describes `UL`, <unit-name> describes `UN` and <event-type> describes `ET`.

The set of GAL expressions is then defined by the following syntax:

<gal-expr> ::= (<unit-decl> **“.”**)* .
<unit-decl> ::= <unit-type-name> <unit-type-def> .
<unit-type-def> ::= **“a” “gal:unit” “;”** <input-var>* <content> <link-expr>? <on-event-expr>*
<content> ::= **“gal:contains” “(”** (**“[”** (<attr-def> | <subunit-expr> | <list-unit-expr>) **“]”**)* **“)”** .
<input-var> ::= **“gal:hasInputVariable”“[” “gal:varName”** <var-name> **“;” “gal:varType”** <domain-concept> **“]” “;”** .
<on-event-expr> ::= **“gal:onEvent” “[” “gal:eventType”** (**“gal:onAccess”** | **“gal:onExit”**) **“;” “gal:update”** <update-query> **“]” “;”**.
<attr-def> ::= **“a” “gal:attribute” “;”** <name-spec>? <label-spec>? **“gal:value”** <val-expr> .
<name-spec> ::= **“gal:name”** <elem-name> **“;”** .
<label-spec> ::= **“gal:label”** <val-expr> **“;”** .
<subunit-expr> ::= **“a” “gal:subUnit”** <name-spec>? <label-spec>? <unit-constr> **“;”** .
<list-unit-expr> ::= **“a” “gal:listUnit”** <label-spec>? <unit-constr> **“;”** .
<link-expr> ::= **“gal:hasLink” “[”** <link-constr> <target-spec>? **“]” “;”** .
<link-constr> ::= **“gal:refersTo”** <unit-type-name> **“;”** <bind-list> .
<target-spec> ::= **“gal:targetUnit”** (<unit-name> | **“gal:_top”**| **“gal:_self”**| **“gal:_parent”**) **“;”** .
<bind-list> ::= <query-bind>? <var-bind>* .
<query-bind> ::= **“gal:hasQuery” “[”** <list-query> **“]” “;”** .
<var-bind> ::= **“gal:assignVariable” “[” “gal:varName”** <var-name> **“;” “gal:value”** <val-expr> **“]” “;”** .
<unit-constr> ::= **“gal:refersTo” “[”** <unit-type-def> **“]” “;”** <bind-list> .

We let each non-terminal also denote the set of expressions associated with it by the syntax, so <gal-expr> is the set of all GAL expressions.

We extend the syntax with some syntactic sugar: inside a <unit-constr> the fragment **“[”** <unit-type-def> **“]”** can be replaced by a <unit-type-name> if in the total expression this is associated with this <unit-type-def>. In other words, if a <unit-type-name> occurs in the <unit-constr> after the **“gal:refersTo”** then it is interpreted as the associated **“[”** <unit-type-def> **“]”**.

Semantics of the Language

The semantics of a GAL expression is described in two parts. In the first part the semantics of a GAL expression describes how a page request results in a unit. In the second part it is defined how a total navigation step is performed, i.e., what are the updates to the database and the resulting current unit in the browser if in a certain unit a certain navigation link is followed.

The resulting unit of a page request

The semantics of the syntax is defined by giving functions for each non-terminal, for example, for expressions `ge` in <gal-expr> we define a partial function that maps a user identifier (the requesting user), a unit type, a binding and a database to a unit, denoted as $[\![ge]\!] : V \times UT \times B \times DB \rightharpoonup U$. We overload this notation for all non-terminals, but not in all cases with the same signature.

Before we proceed with the semantics of GAL expressions, we first define the semantics of <val-expr> defined earlier. We recall the syntax rule:

<val-expr> ::= <value> | **“$”** <var-name> | **“(”** <val-expr> **“.”** <val-expr> **“)”** |
“[” “gal:query” <val-query> **“]”** |
“[” “gal:if” <list-query> **“;”**
(**“gal:then”** <val-expr> **“;”**)? (**“gal:else”** <val-expr> **“;”**)? **“]”** .

For $ve \in$ <val-expr> we define $[\![ve]\!] : B \times DB \rightharpoonup V$ as follows. If $ve = v \in V$ then $[\![ve]\!](b, db) = v$. If $ve =$ **“$”**$x$ with $x \in X$ then $[\![ve]\!](b, db) = b(x)$ if $b(x)$ is defined and undefined otherwise. If $ve =$ **“(”**ve_1**“.”**ve_2**“)”** then, if $[\![ve_1]\!](b, db)$ is a string s_1 and $[\![ve_1]\!](b, db)$ is a string s_2, then $[\![ve]\!](b, db)$ is the string concatenation of s_1 and s_2, otherwise the result is undefined. If $ve =$ **“[”“gal:query”**vq**“]”** then, if $q[b] \in VQ$ then $[\![ve]\!](b, db) = [\![q[b]]\!](db)$, and undefined otherwise. If $ve =$ **“[”“gal:if”**q**“;”“gal:then”**ve_1**“;”“gal:else”**ve_2**“;”“]”** then $[\![ve]\!](b, db) = [\![ve_1]\!](b, db)$ if $[\![q[b]]\!](db) \neq []$, and $[\![ve_2]\!](b, db)$ otherwise. If ve_1 or ve_2 are not specified in `ie` then $[\![ve_1]\!](b, db)$ or $[\![ve_2]\!](b, db)$ are presumed to be undefined.

We now proceed with defining the semantics for GAL expressions. We first recall the relevant syntax rules, and then give the corresponding semantics.

<gal-expr> ::= (<unit-decl> **“.”**)* .
<unit-decl> ::= <unit-type-name> <unit-type-def> .

For $ge \in$ <gal-expr> we define $[\![ge]\!] : V \times UT \times B \times DB \rightharpoonup U$ as follows. Assume that $ge = ud_1$**“.”**$\ldots ud_n$**“.”**, with each $ud_i = ut_i\ udf_i$. Let `j` be the smallest number such that $ut_j = ut$ then $[\![ge]\!](v, ut, b, db) = [\![udf_j]\!](b\#\{(\mathbf{user}, v)\}, db)$, and $[\![ge]\!](v, ut, b, db)$ is undefined if such a `j` does not exist.

<unit-type-def> ::= **“a” “gal:unit” “;”** <input-var>* <content> <link-expr>? <on-event-expr>*
<content> ::= **“gal:contains” “(”** (**“[”** (<attr-def> | <subunit-expr> |
<list-unit-expr>) **“]”**)* **“)”** .
<input-var> ::= **“gal:hasInputVariable”“[” “gal:varName”** <var-name> **“;”**
“gal:varType” <domain-concept> **“]” “;”** .
<on-event-expr> ::= **“gal:onEvent” “[” “gal:eventType”** (**“gal:onAccess”** | **“gal:onExit”**) **“;”**
“gal:update” <update-query> **“]” “;”**.

For $ud \in$ <unit-type-def> we define $[\![ud]\!] : B \times DB \rightharpoonup U$ as follows. Let us assume that $ud =$ **“a”“gal:unit”“;”**$iv_1 \ldots iv_n$ *cont le* $eve_1 \ldots eve_p$, with each
$iv_i =$ **“gal:hasInputVariable”“[”“gal:varName”**x_i**“;”“gal:varType”**dc_i**“]”“;”**,
$cont =$ **“gal:contains”“(”“[”**ce_1**“]”**$\ldots$**“[”**ce_m**“]”“)”**,
each $ce_j \in$ <attr-def> $\cup$ <subunit-type-def> $\cup$ <list-unit-expr>, $le \in$ <link-expr>, and each $eve_k =$ **“gal:onEvent”“[”“gal:eventType”**et_k**“;”“gal:update”**uq_k**“]”“;”**. We say that the binding $b \in B$ is correct under database $db \in DB$ if for all $1 \leq i \leq n$ it holds that $b(x_i)$ is defined and $b(x_i) \in [\![dc_i]\!]_{db}$. If `b` is not correct under `db` then $[\![ud]\!](b, db)$ is undefined, and otherwise $[\![ud]\!](b, db) = \mathbf{un}(C, h, nl)$ where $C = [\![ce_1]\!](b', db) \cdot \ldots \cdot [\![ce_m]\!](b', db)$, $h = \{(et_1, uq_1), \ldots, (et_p, uq_p)\}$, $b' = \{(x, v) \in b \mid x \in \{\mathbf{user}, x_1, \ldots, x_n\}\}$ and $nr = [\![le]\!](b', db)$. If `le` is missing in `ud` then the result is the same except that $nr = \perp$.

Note that the specified input variables are used to restrict the bindings under which the nested content expressions evaluation. Also note that these expressions are assumed to result in a list, which will either be a singleton list containing a content element or an empty list if there no well-defined resulting content element.

<attr-def> ::= **“gal:hasAttribute”“[”** <name-spec>? <label-spec>? **“gal:value”** <val-expr> **“]”**.
<name-spec> ::= **“gal:name”** <elem-name> **“;”** .
<label-spec> ::= **“gal:label”** <val-expr> **“;”** .

For $ad \in$ <attr-def> we define $[\![ad]\!] : B \times DB \to \mathcal{L}(E)$ as follows. If $ad =$ **"gal:hasAttribute"**"["**"gal:name"**an**";"**"gal:value"**"["ve"]"**"]"**, then $[\![ad]\!](b, db) = [\mathbf{at}(an, [\![ve]\!](b, db),])$ if $[\![ve]\!](b, db)$ is defined and [] otherwise. If **an** is not present then we assume that $an = \bot$.

<subunit-expr> ::= **"gal:hasSubUnit" "["**<name-spec>? <label-spec>? <unit-constr> **"]" ";"** .

For $su \in$ <subunit-expr> we define $[\![su|]\!] : B \times DB \to \mathcal{L}(E)$ as follows. If $su =$ **"gal:hasSubUnit"**"["$ls\ ns\ uc$"]"**";"** with $ls =$ **"gal:label"**ul**";"**, $ns =$ **"gal:name"** un**";"** and $ud \in$ <unit-descr> then $[\![su]\!](b, db) = [\mathbf{uc}(ul, un, u_1)]$ if $[\![uc]\!](b, db) = [u_1, \ldots, u_n]$ and [] otherwise. If **ls** or **ns** is not present in **su** then the result is the same except that $ul = \bot$ or $un = \bot$.

<list-unit-expr> ::= **"gal:hasListUnit" "["** <label-spec>? <unit-constr> **"]" ";"** .

For $lue \in$ <list-unit-expr> we define $[\![lue]\!] : B \times DB \to \mathcal{L}(E)$ as follows. If $sue =$ **"gal:hasListUnit"**"["$ls\ uc$"]"**";"** with $ls =$ **"gal:label"**ul**";"** and $uc \in$ <unit-constr> then $[\![lue]\!](b, db) = [\mathbf{ul}(ul, \mathcal{L}[\![uc]\!](b, db))]$. If ls is not present then we assume that $ul = \bot$.

<link-expr> ::=**"gal:hasLink" "["** <link-constr> <target-spec>? **"]" ";"** .
<link-constr> ::=**"gal:refersTo"** <unit-type-name> **";"** <bind-list> .
<target-spec> := **"gal:targetUnit"** (<unit-name> | **"gal:_top"**| **"gal:_self"**| **"gal:_parent"**) **";"** .

For $le \in$ <link-expr> we define $[\![le]\!] : B \times DB \to NL$ as follows. Assume that $le =$ **"gal:hasLink"**"["$lc\ ts$"]"**";"** with $lc =$ **"gal:refersTo"**ut**";"**bl and $bl = q\ vb_1 \ldots vb_n$ with $q \in$ <query> and each $vb_i =$ **"gal:assignVariable"**"["**"gal:varName"**x_i**";"**"gal:value"**ve_i-**"]"**";"** with $x_i \in$ <var-name> and $ve_i \in$ <val-expr>. Also assume that $ts =$ **"gal:targetUnit"**tu**";"** then $[\![le]\!](b, db) = \mathbf{ln}(ut, b, q, be, tu)$ with $be = \{(x_1, ve_1), \ldots, (x_n, ve_n)\}$. If **ts** is missing in **le** then the result is the same except that $tu =$ **"gal:_top"**.

Note that in a link expression the interpretation of the binding list **bl** is different form that in subunit, list-unit and set-unit expressions. Rather than computing one or more bindings it is more or less stored as is such that it can be computed when the link is followed.

<bind-list> ::= <query-bind>? <var-bind>* .

For $bl \in$ <bind-list> we define $[\![bl]\!] : B \times DB \to \mathcal{L}(B)$ as follows. Let $bl = be_1 \ldots be_n$ with $be_i \in$ <query-bind> $\cup$ <var-bind>. Then $[\![bl]\!](b, db) = [b_1 \# \ldots \# b_m \mid b_1 \in [\![be_1]\!], \ldots, b_n \in [\![be_n]\!]]$.

<query-bind> ::= **"gal:hasQuery" "["** <list-query> **"]" ";"**

For $qb \in$ <query-bind> we define $[\![qb]\!] : B \times DB \to \mathcal{L}(B)$ as follows. If $qb =$ **"gal:hasQuery"**"["q"]"**";"** then $[\![qb]\!](b, db) = [\![q[b]]\!](db)$ if $q[b] \in LQ$ and [] otherwise.

<var-bind> ::= **"gal:assignVariable" "["** **"gal:varName"** <var-name> **";"**
"gal:value" <val-expr> **"]" ";"** .

For $vb \in$ <var-bind> we define $[\![be]\!] : B \times DB \to \mathcal{L}(B)$ as follows. If $vb =$ **"gal:assignVariable"**"["**"gal:varName"**x**";"**"gal:value"**ve"]"**";"** then $[\![vb]\!](b, db) = [(x, [\![ve]\!](b, db))]$ if $[\![ve]\!](b, db)$ is defined and [] otherwise.

<unit-constr> ::= **"gal:refersTo" "["** <unit-type-def> **"]" ";"** <bind-list> .

For $uc \in$ <unit-constr> we define $[\![uc]\!] : B \times DB \to \mathcal{L}(U)$. Assume that $uc =$ **"gal:refersTo"**"["ud"]"**";"**be and that $[\![be]\!](b, db) = [b_1, \ldots, b_n]$ then $[\![uc]\!](b, db) = [u \mid b_i \in [\![be]\!](b, db),\ u = [\![ud]\!](b'_i, db)]$ with $b'_i = b \# b_i \# \{(\mathbf{user}, b(\mathbf{user}))\}$.

The side effects and result of following a navigation link

Here we (informally) describe the navigation process that is defined by a certain GAL program. At the core of the navigation semantics is the previously described computation of the unit that is the result of the request of a user. The resulting unit then is presented to the user and becomes the current unit, i.e., the unit that is currently presented to the user. In addition all the update queries associated with the gal:onAccess events in it at any nesting depth are applied to the RDF store in some arbitrary order.

If subsequently the user navigates to an external link then all the update queries associated in it with the gal:onExit event at any nesting depth are applied to the RDF store in some arbitrary order. If the user follows an internal link $\mathbf{ln}(ut, b, q, be, t)$ in the current unit, then the following happens. First the target unit in the current unit is determined as follows. If the target is a unit name then this is the unit in the first unit container with that name. If it is **"gal:_self"** then it is the containing unit, i.e., the unit in which the link is directly nested. If it is **"gal:_parent"** then it is the parent unit, i.e., the unit which has as a content element a unit container, unit set or unit list that refers to the containing unit. If it is **"gal:_top"** then it is the root unit of the current unit. If in any of the preceding cases the result is not well defined then the target unit is the root unit of the current unit. When the target unit is determined then all **"gal:onExit"** events that are directly or indirectly nested in it are applied to the RDF store in some arbitrary order.

Then, the binding that describes the parameters is computed as follows: the binding `b` is combined with the first binding in the result of $q[b]$ and this in turn is combined with the binding that is obtained when evaluating be for binding `b` and the current database instance. The resulting binding is sent together with the specified unit type as a request. Then, if this request results in a unit, the **"gal:onAccess"** update queries in this unit are executed as described before. In the final step the new current unit is the old current unit but with the target unit replaced with the resulting unit of the request.

7.5 Summary

In this chapter we investigated applying data coupling techniques to the specific domain of user modeling. We introduced what factors need to specifically dealt with in exchange of user model data, namely identification, context and interoperability.

We explored two projects in which these issues were dealt with, namely MobiLife and GRAPPLE. MobiLife was a project that centered around mobile and context aware applications, e.g. applications that use time and location to learn something about the user and use this information to improve the user experience. We built a server-side user profile and contextualization framework that can be used by mobile applications to store and exchange user model data and which allows richer personalization in applications.

GRAPPLE was focused on providing a new state of the art adaptation framework, that would also be integrated with learning management systems. Its goal is to improve interoperability between all those systems and improve on the existing infrastructures by the use of an adaptive hypermedia framework. It was envisioned that this would provide an technology-enhanced learning environment which fits in the EU's vision of lifelong learning. We described the several components in the GRAPPLE framework, including the GALE adaptive engine, the GEB communication framework and the GUMF user model framework. We demonstrated that this integration of components and the exchange of user model information provides a tight integration between this system, which improves the student experience.

Finally, we looked at an engine-independent adaptation language, called GAL. GAL is based on the Hera-S models. We showed how it provides a generic way of defining an adaptive application which is able to express the most used types of adaptation. We also provided a first formalization of the GAL language, which could be a start for standardization if this would seem useful by the community.

Chapter 8

Conclusions and Future Work

This last chapter is about our conclusions and future work. We reexamine the research questions that were formulated in chapter 1 and formulate the answers that follow from the work throughout the chapters. We also give a short outlook into the future and propose some suggestions for future research.

8.1 Conclusions

In this thesis we presented a completely remodeled and reimplemented version of Hera [Vdovjak et al., 2003], which is a Model-Driven Web Engineering method with a strong focus on the Semantic Web. The approach has now an even stronger focus on Semantic Web, allowing any RDF data source as input, to make use of the many of these sources that are nowadays available on the Web. The architecture of this new version of Hera, called Hera-S, offers capabilities and possibilities for a greatly enhanced flexibility in designing Web-based applications. We have investigated and discussed some of these capabilities like mixing content and user model or mixing possible content types, as well as capabilities for for the use of scripting and web services. We also showed some of the new possibilities for using data integration techniques at the Hera-S back end, we investigated the use of aspect-oriented programming techniques to inject functionality in running applications, and the use of specialized components for presentation generation together with Hera-S.

The largest focal point was data integration. As this is an issue larger than within Hera-S alone, we have studied this issue in isolation. This resulted in a tool that can be used to integrate heterogenous datasourses via a mapping language and engine that can actually do the integration of a datasource with another one, together with a set of techniques that allows to discover possible relationships between elements in different RDF (or other) datasets. We showed the universal applicability of this approach in several case studies in diverse domains. It showed us that solving the data integration issue opens up a wealth of interesting new rich Web applications in these domains.

We now look at the questions from chapter 1. For every question we describe what has been done in this thesis to solve it.

Question 1: How do MDWE architectures have to be adapted to enable flexible and dynamic Semantic Web data integration?

We presented a totally rebuilt version of the Hera-S MDWE method and implementation in chapter 3. The goal was, as stated in the question, to enable flexible and dynamic

Semantic Web data integration. In general we were interested in simplifying existing models used in Hera-S, i.e. the Domain Model (DM), User Model (UM) and Application Model (AM). By leaving full freedom in the DM and the UM, i.e. by allowing to reuse any RDF related input, and even allow to mix these models provides a larger freedom and reusability of existing data sources and enables support of the many Semantic Web tools available on the Web.

Furthermore, the DM and UM are now completely decoupled from the Hera-S implementation. This means that Hera-S does not keep these models under its own closed control, but allows other processes access to the data - also at runtime. This allows data integraton processes to get the final content in the background that is used inside the Web application. Thus, this open approach proves to be crucial to enable flexibility and dynamic behavior.

These decisions did have their effect on the rest of Hera-S' design. For example, the AM had to be completely revised as it could no longer depend on a certain structure of the DM. Instead the AM uses SPARQL query expressions that help us to refer to the data in both UM and DM. This comes at some cost in terms of simplicity, as every reference to data has to be a query, even though the queries themselves are often simple. Some of this added complexity can be alleviated by the model builders which are able to generate the queries for simple RDF elements by just clicking on them. A large benefit compared to past AM structures is the added expressive power by using queries, where now every element of both the content and the user model can be used everywhere in the Web application and for personalization.

This of course also had consequences for the implementation of Hera-S. This implementation, called Hydragen, is basically stateless along the architectural guidelines of the REST software architecture [Fielding, 2000]. This means that Hydragen does not maintain state information for its clients, but that the clients need to communicate all information needed to generate the next AM (instance) page. Whenever such a call from the client comes in, Hydragen will query the AM at the Sesame database to retrieve the relevant pieces of the AM to generate the AMP. This means that external changes to the AM will be immediately available to the clients as Hydragen will query the current state of the AM and will not keep this in memory (except for possible caching mechanisms for performance reasons).

Furthermore, as part of a separation of concerns effort we decoupled different functionalities in Hera-S. Hera-S is now focussed on application modeling, which has always been its core. This is where our desing efforts were mainly aimed at. Actual presentation generation was implemented as a separate functional block, which can easily be replaced by another functional component as we showed when we simply replaced our own simple presentation module by the specialized AMACONT presentation generation framework. Similarly, access to the datasources and models was implemented using the Sesame RDF database. However, it is rather simple to replace Sesame by another RDF store if this would be for some reason beneficial. As long as this other store has SPARQL support, it can simply replace Sesame by implementing a single abstract connection class that describes how to connect with the replacing datasource. In this way also specialized RDF stores could be used that offer some specialized functionality we might want to use for specific applications.

The resulting focus of Hera-S on the application model actually inspired us to use it as a basis for an engine-independent adaptation language called GAL, as we described in Section 7.4.3.

Question 2: How can MDWE benefit from data coupling techniques?

Data coupling, i.e. coupling data sets by adding links between concepts instead of data integration efforts in which data is actually transformed to some target structure, allows MDWE methods like Hera-S to still use it like one big integrated dataset. This is because RDF and its query language SPARQL allow to traverse over these links, where the actual origin of this link is not necessarily important. Using data coupling is very interesting because it allows the use of an external dataset that the MDWE engine has no control over. This is important for several reasons. Not everyone will completely open up their dataset, e.g. for legal reasons or to just ensure the correctness of the dataset they govern. Another reason may be a highly volatile dataset with realtime data for which continues exports may be unwanted. By using data coupling Hera-S is still able to use these potentially interesting datasets directly in a Web application.

Consequently, data coupling allows combining data sources and mashing up interesting Web applications. For example, the scenario that we gave in Section 4.2 where might want to combine datasets about movies, with a dataset about cinemas and a dataset about restaurants, for making a Web application that allows users to plan a complete night out to the movies.

Our decoupling of models and engine provides more benefits as we have shown in our work on aspect-orientation in Section 4.4. This technique allows to couple the existing AM with new funcitonality using data coupling techniques, i.e. by refering to which navigational unit, or attribute, some injected piece of model belongs. In this way, the added functionality does not actually has to be injected in the actual AM datasource, but can be in a separate context. When Hydragen queries the AM via configuration it will then also query the relevant parts of new code. This is good for maintainability, because if some aspect-oriented functional block is no longer needed it can be easily deleted from this context instead of a need to modify the original AM context. In a similar way it could also be possible to couple applications, and mix local applications with remote ones.

Question 3: How to couple semantic heterogenous RDF data?

By creating links between concepts in those data sources. However, this is certainly not simple in many cases. If you have large datasets and changing datasets it's laborious and sometimes just impossible to do this by hand. Also, sometimes just data coupling is not enough and data integration is a better solution. For example, when we have ten different heterogenous sources which we want to navigate in a Web application, this could mean the designer has to create a complex query that refers to the specifics of every data source to get the whole dataset.

In order to help designers solve these we problems we introduced the Relco toolset in chapter 5. This toolset offers two main functionalities. A transformation language which allows to specify how content (or parts of it) can actually be transformed into a target format. Two such languages were created, one based on the SWRL rule language, and one based on an extended version of the SPARQL query language. The transformation always consists of two parts, one selection mechanism that selects the data to be transformed that includes variable bindings. The second part is the transformation that transforms the variables and applies a structure transformation and value manipulation in order to translate the data.

The second part of the toolset contains techniques for finding candidate concepts in an ontology for a textual representation of some input concept. These techniques

included syntax-based techniques (i.e. string comparisons), semantic-based techniques (i.e. by using external knowledge sources like WordNet and exploit context information), and collaborative techniques (i.e. based on user feedback).

Relco provides designers with tools that help in constructing mappings. Accuracy of Relco depends on many variables, like quality of source and target ontologies, the external knowledge sources, the natural language that is used, and other domain specific factors. In some domains Relco can provide only rough guesses while in others it can be right in most of the cases and can completely automate data integration.

Relco can be used as a separate toolset, but also as a library within other projects that need data integration functionality.

Question 4: How can we apply data integration in Web applications in various domains?

In chapters 6 and 7 we consider five diverse applications is various domains that all have to deal with data integration in one form or another.

- χ *Explorer* is a Web application to disclose a very large collection of historic data from Regionaal Historisch Centrum Eindhoven (RHCe). RHCe has a large number of different datasets that were acquired in numerous ways, and of which the digital data was stored in several databases and other data sources. In order to offer these fragmented datasource as one homogeneous dataset to users, these datasets needed to be somehow integrated. This was done by first writing RDF wrappers for the various datasources that were able to transform the original data into RDF (and vice versa for some instances). Next these sets were mapped onto a single extendible data structure. As part of the integration effort time and location data were standardized, i.e. mapped to one specific format, so that later on we could provide homogenous browsing over these facets. Also, the available metadata was used to link the objects to concepts in a standard RHCe ontology (i.e. using Relco), which later on provided subject-based graph navigation, using the relations in the standard ontology as edges and the concepts as nodes. For objects that did not contain (enough) metadata user-based tagging was used, were Relco was used again to provide suggestions for linking the user tags to concepts in the ontology.

- *The SenSee TV-Recommender* is a Web application (and underlying framework) for providing television recommendations. In order to provide these recommendations datasets are combined, e.g. program guide data with data from the IMDb for movies and from Wikipedia information about presenters, shows, etc. Because of the size and volatility of the datasets this integration had to be done fully automatic, which could be realised with Relco. Like in χ Explorer, dates and locations were standardized.

- *ViTa* was a project that evaluated several ways of using user tagging and automatically generated metadata for finding educational videos, via a user study with intermediate level students. We did an integration effort by integrating user tags with concepts of an existing ontology using Relco to provide concept suggestions. Another approach, also with Relco, was to relate user tags with previous user tags and present those as alternatives in order to convergate the metadata tagset for particular videos.

- *MobiLife* was a project for advancing mobile applications with personalization and contextualization. Part of this effort was to integrate user models of existing and

new mobile phone applications. In order to achieve that, designers could use Relco-based techniques to map and exchange user model information between applications. Combining this richer user models should help in providing better personalization.

- *GRAPPLE* was a project which focussed on integrating existing leading Learning Management Systems with a beyond the state of the art Adaptive Learning Environment in order to provide students with a technology-enhanced learning environment. This integration effort lead to the GRAPPLE framework, which among others consisted of a User Modeling Framwork called GUMF. GUMF was a crucial factor for the integration efforts, as true integration includes exchanging mutual beneficial information. GUMF was realized with a transformation language based on the Relco transformation language and can benefit from the Relco for finding the relations between datasets. In this way, for example, results from LMS quizzes could be used to estimate the knowledge of the user in GALE, while the LMSs could get information about the progress of the user with their adaptive courses.

Question 5: How can domain specific Web applications benefit from data integration?

The applications, as mentioned under the answer of Question 4, all have some benefit from data integration. We consider some benefits that these application have that is made possible by data integration:

- χ *Explorer* provided users a homogenous browsing experience over an integrated data set originating from very heterogenous sources. Browsing was based on metadata items that were common among most historical objects in the dataset, namely time, location and contentwise descriptions using an ontology. Users can either search the dataset, but also browse the different facets. For time, objects are put on a timeline. In this way users can easily browse objects that are related to a very specific time frame, like the day Eindhoven was bombed during WW2. For location, objects were grouped and put on a map. In this way users could access every object that was related to a specific location. For metadata descriptions we provided a simple graph view, in which for every object that relates to a specific concept in the RHCe ontology we can see other objects which relate to the same concept or objects that relate to related concepts.

- *The SenSee TV-Recommender* provides several browsing paradigms. Besides browsing using the typical tv-guide view, one can also view 'related' programs, which are defined via semantic relations, e.g. movies by the same director or actor, programs with the same presenter, programs of the same genre, etc. Recommendations work similarly. People can recommend a program, and using predefined weights of relation types this propagates through the underlying graph and implicitly also rates semantically related programs. In this way people can get recommendations for programs based on their ratings.

- *ViTa* evaluated several searching strategies for educational videos that were provided with metadata, also using several strategies. In the evaluation, students had to find specific information in educational videos searching over the acquired metadata sets. The results showed that using collaborative techniques performed best. Semantic search also performed well and provided results that were similar to the professionally

tagged datasources. Both performed significantly better than searching based on automatic extraction of metadata based on documents describing the resource, i.e. the way most multimedia Web search engine work nowadays.

- *MobiLife* used the user modeling framework for exchanging contextualized user data between applications to facilitate a number of example mobile phone applications. This includes an semi-automatic blogging application that collects information about user activity and helps the users to create rich blogs about their activities. A personal portal application that uses contextual data and other user information to collect local and relevant content feeds, like news, blogs, etc. It also includes a workout application that collects personal health information from sensors and other context to provide personalized exercise programs. Generally, many other applications exploiting the possibilities of this framework can be thought of and created.

- *GRAPPLE* used data integration for exchanging user information. In this way details of the course, including aggregated class information, could be visualized inside the various LMS. It also allows LMS to collaborate, which is for example important if students study courses on several universities that typically use different LMSs. Another application of GUMF is exchanging user data between different GALE applications. For example, if someone starts the advanced business English course, the information from the beginning course can be used to estimate the current level of knowledge of the student.

8.2 Contributions

Here we summarize the contributions made by this thesis.

The first contribution is a redesign and reimplementation of the Hera method, which is more powerful and flexible in several ways. Hera-S is capable to use any RDF data source as input. Moreover by using a query mechanism it is more flexible and powerful in combining and reusing data, including user and context data, in every page. Units can be reused and included as subunits inside other units. Hera-S also allows the use of scripts and web services in its models. Furthermore, Hera-S provides a basic presentation model which allows to model presentation concerns to presentation units. Hera-S is implemented in a stateless engine called Hydragen. Hydragen allows updates to the different models (domain, user, application and presentation) at runtime which allows several extension points to be implemented. The results of this contribution have been published in the following papers: [Houben et al., 2008; Schwinger et al., 2008]. The Hera-S application model was finally used as the basis for the GAL-language as published in [van der Sluijs et al., 2009].

The second contribution is in a set of tools and extensions that were created on top of Hera-S that explore its power and flexibility. This includes data integration capabilities using Sesame. This has been published in [van der Sluijs et al., 2006]. Another tool is Hera-Studio which offers model builders specific for the Hera-S method. These model builders allow to create Hera-S models graphically, directly deploy the models in Hydragen and provide basic model checking capabilities. Another extension point on top of Hera-S is aspect-oriented adaptation. Aspect-orientation in Hera-S allows to include application wide functionality to be included on runtime. This resulted in a query-based method which does not require code changes in the original application models. This has been

published in the following papers: [Casteleyn et al., 2009, 2006a,b]. The presentation layer of Hera-S was extended with a specialized presentation adaptation component called AMACONT. This includes integration of AMACONT in the Hydragen engine and the conceptual connection between the two frameworks, as well as the extension of AMACONT with semantic-based adaptation. This was published in the following papers: [van der Sluijs et al., 2010; Niederhausen et al., 2009; Houben et al., 2005].

The third contribution is a data integration toolset called Relco. Relco offers a data transformation language called STL which extends SPARQL with an additional TRANSFORM construct. Relco also combines a set of techniques that can be used to relate concepts from one ontology to concepts a target ontology, e.g./ techniques like string matching, utilization of helper ontology, context disambiguation, etc. The results for this contribution have been published in the following papers: [van der Sluijs and Houben, 2009, 2006a,b, 2005].

The fourth contribution is a set of applications that are created using the Relco toolset and its techniques that exploit the advantages of data integration. In χ Explorer, an cultural heritage application, several heterogenous databases have been integrated. This allowed us to build a faceted browsing application which allows users to browse objects using a map, a timeline and a graph. It also provides users ways to actively contribute new metadata. Results on χ Explorer have been published in the following papers: [van der Sluijs and Houben, 2010, 2008c,a]. In SenSee, a TV guide application, several TV-guide sources have been integrated with information from IMDb and Wikipedia. The underlying graph can be used to semantically extend user queries and to compute semantical recommendations. Results on SenSee have been published in the following papers: [Bellekens et al., 2008, 2007]. ViTa is a student video portal that allows users to tag educational videos and uses this metadata for improving searching specific videos. Techniques from Relco were used to improve user tagging quality and use semantic relation in an helper ontology to improve finding relevant videos for user queries. Evaluations showed that applying semantic tagging and searching was significantly better for students than using metadata that was created by professionals. Results of our contribution to ViTa were published in [van der Sluijs and Houben, 2008b]. MobiLife was a project on mobile phone infrastructure. Our main focus was a user modeling framework called Personal Profiling Function, which allows mobile applications to store context-based user model data and which can use Relco techniques for exchange of user model data between applications. Within the project it was shown that this approach works as applications that use this framework were made. Results of our contribution to MobiLife were published in [Sutterer et al., 2007]. GRAPPLE was a project for providing LMSs adaptive learning facilities. This thesis focuses on the general architecture and communication infrastructure for the project which facilitated the integration of LMSs and GALE, and the general infrastructure of the GUMF framework that can use techniques similar to Relco's STL for exchanging user model data between applications. Results of our contribution to GRAPPLE were published in [De Bra et al., 2012; van der Sluijs and Höver, 2009a; De Bra et al., 2010a; van der Sluijs and Höver, 2009b; Leonardi et al., 2009; Abel et al., 2009].

8.3 Future Work

This thesis explored several topics which can be the basis for further research. Following is a number of possible interesting research topics that might deserver attention.

8.3.1 Hera

In our research we assumed that the datastructure of content sources is more or less stable. Even though we allow for flexibility and change in the application model, a dynamic content structure can still have a negative impact on an application. In many cases this dependency on stability is fine since many data sources show stability with respect to their schema. However, some scenarios might show frequent change. Dealing with these changes and automatically (or with minimal effort) updating the application model would be an interesting line of research. This means that first changes that impact the application have to be detected using some advanced model checking capabilities. After detection steps have to be taken to either update the application model or at least inform the designer for possible issues with their application.

Along the same lines, we could envision automatic model generation. Many scenarios exist in which designers will follow very similar ways in building a Web application for a given dataset. It might be possible to define generalized strategies and patterns that can be applied to certain datasets. Automatic generation of application models, or part of application models, might greatly improve the efficiency in design. Similarly, recording and analyzing designer's behavior might present ways to deduce new strategies and patterns that could be reused.

Another line of research may be automatic adaptation based on user behavior. Especially for educational courses with a focus on knowledge gain it might be interesting if the system can introduce small changes to courses, e.g. with respect to adaptativity, order, presentation, etc. By monitoring these changes and measuring learning efficiency in terms of results of quizzes, etc, allows assessing the quality of these changes. Furthermore, they might present the designer with clues about the course itself, e.g. about which parts of a course are too difficult or too simple in certain contexts.

In Hera-S we considered data source heterogeneity in the light of data integration purposes. Another interesting topic would be data distribution for performance reasons. Some introductory work was done on this work [Bellekens et al., 2008; Verheijen, 2008], but this is an interesting research topic in its own rights. This includes related topics like caching, load distribution, sharding, consistency, etcetera. This is partly RDF database research, but some interesting strategies might exist that exploits knowledge of the underlying MDWE framework. Next to back-end distribution, front-end distribution might prove interesting. For example, aspect-orientation provides interesting way to implement functional blocks. Combining different functional blocks into one Web application with one homogenous presentation might be a challenging goal. Similarly, mixing (parts of) existing Web applications into new functionality could be a possibility.

Note that much of the future work we suggested for Hera is not Hera-specific, i.e. they might be researched either in general Semantic Web applications or another MDWE context.

8.3.2 Relco

Relco allows the exploitation of external knowledge sources. This provides good results and demonstrates the possibilities of reuse on the Semantic Web. This is an interesting path to do further research on. Human semantic processing power is based on a (relative) enormous set of background knowledge. By exploiting large sets of connected datasets (e.g. the Linked Open Data dataset) and efficiently access to these datasets, e.g. by applying reasoning, knowledge chains, etc, the results of semantic interpretation and disambiguation could improve. The potential of dealing with many large datasets versus the effective

and efficient use of this data is an interesting consideration. It also introduces many diverse issues like the need for context dependent assessments of quality, correctness and applicability of (parts of) datasets.

Another way to improve on Relco is by learning from the designers. By using more quantitative considerations (e.g. through machine learning), we could try to assess the process of internal and implicit knowledge modeling by the designer. It would be interesting to use designer's behavior for deducing if some (e.g. unknown) underlying concept or relationships must assist. This type of research could be interesting in two ways. First, it provides a platform like Relco with new information that could be exploited to get better matches between concepts. It could however also be used for deducing new knowledge sources and derive new useful ontologies that could makes implicit knowledge from designers more explicit and useful.

8.3.3 Cultural Heritage

In χ Explorer we have explored the data integration issue within one institution which wants to share its data with the public. A logical next step would be to connect different institutions together. This would mean dealing with diversification of solutions and limitations that were chosen on a grander scale.

Besides an organizational challenge, a nation wide integration of datasets would open up new possibilities for new applications of these datasets. For example, a historic framework could be assembled with multiple innovative analysis and visualizing tools for historic research. Combining datasources could for example provide insights in the dynamics of wealthy families over the century, e.g. if they spread over the country over the century, if they could hold on to their wealth, and the relationships with other wealthy families. Analysis and visualization techniques could greatly improve this line of research. But most importantly the necessary sources need to be integrated and connected. Solutions should be found to effectively and efficiently integrate many diverse sources.

Further research could be the integration of feature recognition techniques for identifying semantic relationships between objects. This could be a possible enrichment for dealing with the 'cold start' problem. Especially cultural heritage institutions with an already enormous growing set of uncataloged objects need ways to gather new metadata. Even though feature recognition could help in this respect, current feature recognition techniques deal with low accuracy rates. Dealing with uncertainty of these processes in combination with users might be challenging, e.g. you might not want to confront the users with too many faulty data which might give the users the impression that the system just doesn't work. In other words, research should be done on how a system which lacks the metadata to perform optimally can use 'crowd-sourcing' without scaring away the user.

8.3.4 SenSee

SenSee is a recommender system that uses a semantic network for computing recommendations. Most popular recommenders online use collaborative filtering. It would be interesting to explore a recommendation system that combines these techniques. For example, detecting subnets of semantic networks of which the nodes are often explored by similar users can be used to compute which semantic relationships are considered important in a social context. This is only an example of how combining background knowledge together with social engineering could improve the knowledge about users which in its turn might allow better recommendations.

There are several other research directions for SenSee. One is including Video On-Demand sources, Web videos and Web streams in the TV-guide recommendations. As more and more new TV-sets are connected to the internet this is a perfectly viable option. This would greatly increase the possible sources too choose from and it would be a great source to compute a complete 'gapless' TV-schedule for the night. This could be achieved by for instance a collaborating web of agent-based systems that continuously monitors the Web for content. Building in synchronization between agents of friends and colleagues could for example ensure that people have something to discuss at work or in some social setting. It is an interesting challenge to build agents with these kinds of capabilities - especially effective and non-pervasive communication with users could be challenging.

8.3.5 GRAPPLE

A bottleneck for the acceptance of adaptive systems is the quality of the authoring tools. The authoring tools must provide an intuitive interface which guide authors throughout the entire authoring process. It should provide an environment that lets authors think naturally in the adaptive exploratory paradigm instead of the usual linear course design. A similar issue holds for user modeling and user model exchange. These are complicated issues for which innovative authoring interfaces are needed to guide the authors through the process. If done right this could lead to a breakthrough for ALEs.

Another issue that should get attention is error detection. GALE is a powerful adaptive engine, but this power also makes it error prone in which authors might not realize the consequence of some seemingly logical choices. Static analysis and simulation analysis could be tools to find all kinds of logical errors like dead ends, unreachable material, recursion loops, etc.

User model information over several courses could also be the topic of further studies. This information could provide insight in student's interests and competencies. This could be the basis of course advice (e.g. which course are interesting for the user) and maybe even career advice. This could be achieved with some kind of global user model that extracts key characteristics from courses. If a part of this process could be done (semi-)automatically remains to be seen, but it would be at least be an interesting topic of study.

Bibliography

Abel, F., Heckmann, D., Herder, E., Hidders, J., Houben, G.-J., Krause, D., Leonardi, E., and van der Sluijs, K. (2009). A framework for flexible user profile mashups. In *Proceedings of the International Workshop on Adaptation and Personalization for Web 2.0 (APWEB 2.0 2009)*.

Aleksovski, Z., ten Kate, W., and van Harmelen, F. (2006). Ontology matching using comprehensive ontology as background knowledge. In et al., P. S., editor, *Proceedings of the International Workshop on Ontology Matching at ISWC 2006*, pages 13–24. CEUR.

Algergawy, A., Massmann, S., and Rahm, E. (2011). A clustering-based approach for large-scale ontology matching. In *Advances in Databases and Information Systems*, volume 6909 of *LNCS*, pages 415–428. Springer Berlin / Heidelberg.

Ames, M. and Naaman, N. (2007). Why we tag: motivations for annotation in mobile and online media. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 971–980, New York, NY, USA. ACM.

Ardissono, L., Kobsa, A., and Maybury, M. T., editors (2004). *Personalized Digital Television*, volume 6 of *Human-Computer Interaction Series*. Springer.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Baumeister, H., Knapp, A., Koch, N., and Zhang, G. (2005). Modelling adaptivity with aspects. In *Proceedings of the 5th International Conference on Web Engineering (ICWE'05)*, pages 406–416, Sydney, Australia. Springer.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2004). Owl web ontology language reference. W3C Recommendation 10 February 2004. `http://www.w3.org/TR/owl-ref/`.

Beckett, D. and Berners-Lee, T. (2011). Turtle - terse rdf triple language. W3C Team Submission. `http://www.w3.org/TeamSubmission/turtle/`.

Beckett, D. and Broekstra, J. (2008). Sparql query results xml format. W3C Recommendation 15 January 2008. `http://www.w3.org/TR/rdf-sparql-XMLres/`.

Bellas, F. (2004). Standards for second-generation portals. *IEEE Internet Computing*, 8(2):54–60.

Bellekens, P. (2010). *An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain*. PhD thesis, Technische Universiteit Eindhoven, Eindhoven, the Netherlands.

Bellekens, P., van der Sluijs, K., Aroyo, L., and Houben, G.-J. (2007). Engineering semantic-based interactive multi-device web applications. In *Proceedings of the 7th International Conference on Web Engineering (ICWE'07)*, volume 4607 of *Lecture Notes in Computer Science*, pages 328–343, Como, Italy. Springer.

Bellekens, P., van der Sluijs, K., van Woensel, W., Casteleyn, S., and Houben, G.-J. (2008). Achieving efficient access to large integrated sets of semantic data in web applications. In *Proceedings of the 2008 Eighth International Conference on Web Engineering*, ICWE '08, pages 52–64, Washington, DC, USA. IEEE Computer Society.

Berners-Lee, T. (2003). WWW past & future. `http://www.w3.org/2003/Talks/0922-rsoc-tbl/`.

Berners-Lee, T. (2006). Design issues: Linked data. `http://www.w3.org/DesignIssues/LinkedData.html`.

Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., and Secret, A. (1994). The world-wide web. *Commun. ACM*, 37(8):76–82.

Berners-Lee, T. and Fischetti, M. (1999). *Weaving the Web; The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor (2 Cassettes)*. Harper Audio.

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

Biron, P. V., Permanente, K., and Malhotra, A. (2004). Xml schema part 2: Datatypes. W3C Recommendation 28 October 2004. `http://www.w3.org/TR/xmlschema-2/`.

Blanco Fernández, Y., Pazos Arias, J. J., Gil Solla, A., Ramos Cabrer, M., and López Nores, M. (2006). Bringing together content-based methods, collaborative filtering and semantic inference to improve personalized tv. *4th European Conference on Interactive Television (EuroITV 2006)*.

Boda, P., Brgulja, N., Gessler, S., Giuliani, G., Koolwaaij, J., Martin, M., Melpignano, D., Millerat, J., Nani, R., Nurmi, P., Ollikainen, P. J., Polasek, P., Radziszewski, M., Salacinski, M., Schultz, G., Sutterer, M., Trendafilov, D., and Ukropec, L. (2007). In Klemettinen, M., editor, *Enabling Technologies for Mobile Services: The MobiLife Book*. John Wiley & Sons.

Bos, B., Çelik, T., Hickson, I., and Lie, H. W. (2011). Cascading style sheets level 2 revision 1 (css 2.1) specification. W3C Recommendation 07 June 2011. `http://www.w3.org/TR/CSS2/`.

Bozzon, A., Comai, S., Fraternali, P., and Carughi, G. T. (2006). Conceptual modeling and code generation for rich internet applications. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 353–360, New York, NY, USA. ACM.

Brambilla, M., Butti, S., and Fraternali, P. (2010). Webratio bpm: A tool for designing and deploying business processes on the web. In Benatallah, B., Casati, F., Kappel, G., and Rossi, G., editors, *Web Engineering*, volume 6189 of *Lecture Notes in Computer Science*, pages 415–429. Springer Berlin / Heidelberg.

Brambilla, M., Celino, I., Ceri, S., Cerizza, D., Valle, E. D., and Facca, F. M. (2006). A software engineering approach to design and development of semantic web service applications. In *Proceedings of the 5th International Semantic Web Conference (ISWC'06)*, pages 172–186, Athens, GA, USA. Springer.

Brambilla, M., Comai, S., fraternali, P., and Matera, M. (2008). Designing web applications with webml and webratio. In Rossi, G., Pastor, O., Schwabe, D., and Olsina, L., editors, *Web Engineering: Modelling and Implementing Web Applications*, volume 12 of *Human-Computer Interaction Series*, pages 221–261. Springer.

Brickley, D. and Guha, R. (2004). Rdf vocabulary description language 1.0: Rdf schema. W3C Recommendation 10 February 2004. `http://www.w3.org/TR/rdf-schema/`.

Broekstra, J. (2005). Serql: A second-generation rdf query language. In *Storage, Querying and Inferencing for Semantic Web Languages, PhD Thesis*, pages 67–87. PhD Thesis, Vrije Universiteit Amsterdam.

Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying rdf and rdf schema. In Horrocks, I. and Hendler, J., editors, *Proceedings of the First Internation Semantic Web Conference*, number 2342 in Lecture Notes in Computer Science, pages 54–68. Springer Verlag.

Brugman, H., Malaisé, V., and Gazendam, L. (2006). A web based general thesaurus browser to support indexing of television and radio programs. In *Proceedings of the 5th international conference on Language Resources and Evaluation (LREC 2006)*, pages 1488–1491.

Brusilovsky, P. (1996). Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6:87–129.

Brusilovsky, P. (2001). Adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 11(1 - 2):87–110.

Brusilovsky, P. (2007). Adaptive navigation support. In Brusilovsky, P., Kobsa, A., and Nejdl, W., editors, *The Adaptive Web*, pages 263–290. Springer-Verlag, Berlin, Heidelberg.

Bulterman, D., Jansen, J., Cesar, P., Mullender, S., Hyche, E., DeMeglio, M., Quint, J., Kawamura, H., Weck, D., Pañeda, X. G., Melendi, D., Cruz-Lara, S., Hanclik, M., Zucker, D. F., and Michel, T. (2008). Synchronized multimedia integration language (smil 3.0). W3C Recommendation 01 December 2008. `http://www.w3.org/TR/SMIL/`.

Busch, M., Knapp, A., and Koch, N. (2011). Modeling secure navigation in web information systems. In *BIR*, LNBIP, pages 239–253. Springer Verlag.

Casteleyn, S., Fiala, Z., Houben, G.-J., and van der Sluijs, K. (2006a). Considering additional adaptation concerns in the design of web applications. In *Proceedings of the 4th International Conference on Adaptive Hypermedia (AH'2006)*, volume 4018 of *Lecture Notes in Computer Science*, pages 254–258, Dublin, Ireland. Springer.

Casteleyn, S., Fiala, Z., Houben, G.-J., and van der Sluijs, K. (2006b). From adaptation engineering to aspect-oriented context-dependency. In *Proceedings of the 15th International Conference on World Wide Web (WWW'06)*, pages 897–8998, Edinburgh, Scotland. ACM.

Casteleyn, S., van Woensel, W., van der Sluijs, K., and Houben, G.-J. (2009). Aspect-oriented adaptation specification in web information systems: a semantics-based approach. *New Review of Hypermedia and Multimedia*, 15(1):39–71.

Ceri, S., Daniel, F., Demaldé, V., and Facca, F. M. (2005). An approach to user-behavior-aware web applications. In *Proceedings of the 5th International Conference on Web Engineering (ICWE'05)*, pages 417–428, Sydney, Australia. Springer.

Ceri, S., Daniel, F., Matera, M., and Facca, F. M. (2007). Model-driven development of context-aware web applications. *ACM Trans. Internet Technol.*, 7(1).

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., and Matera, M. (2002). *Designing Data-Intensive Web Applications*. Morgan Kaufmann.

Chow, C. and Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467.

Choy, S.-O. and Lui, A. K. (2006). Web information retrieval in collaborative tagging systems. In *Proceedings of the International Conference on Web Intelligence*, pages 352–355, New York. IEEE Press.

Clark, J. (1999). Xsl transformations (xslt). W3C Recommendation 16 November 1999. `http://www.w3.org/TR/xslt/`.

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176.

De Bra, P., D. Smits and, K. v. d. S., Cristea, A., Hendrix, M., and other members of the GRAPPLE Research Team (2010a). Grapple: Personalization and adaptation in learning management systems. In *ED-MEDIA 2010-World Conference on Educational Multimedia, Hypermedia and Telecommunications*, Proc. of ED-MEDIA 2010. Association for the Advancement of Computing in Education (AACE).

De Bra, P., Smits, D., and Stash, N. (2006). The design of aha! In *Proceedings of the seventeenth conference on Hypertext and hypermedia*, HYPERTEXT '06, pages 133–134, New York, NY, USA. ACM.

De Bra, P., Smits, D., van der Sluijs, K., Cristea, A. I., Foss, J., Glahn, C., and Steiner, C. M. (2012). Grapple: Learning management systems meet adaptive learning environments (forthcoming).

De Bra, P., Smits, D., van der Sluijs, K., Cristea, A. I., and Hendrix, M. (2010b). Grapple: Personalization and adaptation in learning management systems. In *ED-MEDIA 2010-World Conference on Educational Multimedia, Hypermedia & Telecommunications*, Proc. of ED-MEDIA 2010. Association for the Advancement of Computing in Education (AACE).

de Moura, S. S. and Schwabe, D. (2004). Interface development for hypermedia applications in the semantic web. In *LA-WEBMEDIA '04: Proceedings of the WebMedia & LA-Web 2004 Joint Conference 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress*, pages 106–113, Washington, DC, USA. IEEE Computer Society.

De Troyer, O., Casteleyn, S., and Plessers, P. (2008). Wsdm: Web semantics design method. In Rossi, G., Pastor, O., Schwabe, D., and Olsina, L., editors, *Web Engineering: Modelling and Implementing Web Applications*, volume 12 of *Human-Computer Interaction Series*, pages 303–352. Springer.

Dey, A. K. (2000). *Providing architectural support for building context-aware applications*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA. AAI9994400.

Doan, A., Madhavan, J., Domingos, P., and Halevy, A. (2002). Learning to map between ontologies on the semantic web. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 662–673, New York, NY, USA. ACM.

Earnshaw, N. (2005). The tv-anytime content reference identifier (crid).

Euzenat, J. and Shvaiko, P. (2007). *Ontology Matching*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Facca, F. M. and Brambilla, M. (2007). Extending webml towards semantic web. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1235–1236, New York, NY, USA. ACM.

Falkovych, K., Sabou, M., and Stuckenschmidt, H. (2003). Uml for the semantic web: Transformation-based approaches. In *Knowledge Transformation for the Semantic Web*, pages 92–106. IOS Press,US.

Fiala, Z., Frasincar, F., Hinz, M., Houben, G.-J., Barna, P., and Meissner, K. (2004). Engineering the presentation layer of adaptable web information systems. In *Fourth International Conference on Web Engineering (ICWE2004), Munich*, number 3140 in Lecture Notes in Computer Science, pages 459–472. Springer-Verlag Berlin Heidelberg.

Fiala, Z., Hinz, M., MeiSSner, K., and Wehner, F. (2003). A component-based approach for adaptive dynamic web documents. *Journal of Web Engineering, Rinton Press*, 2(1&2):058–073.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine.

Fons, J., Pelechano, V., Pastor, O., Valderas, P., and Torres, V. (2008). Applying the oows model-driven approach for developing web applications. the internet movie database case study. In Rossi, G., Pastor, O., Schwabe, D., and Olsina, L., editors, *Web Engineering: Modelling and Implementing Web Applications*, Human-Computer Interaction Series, chapter 5, pages 65–108. Springer London.

Frasincar, F. (2005). Hera presentation generator. In *Hypermedia Presentation Generation for Semantic Web Information Systems*, pages 67–87. Eindhoven University of Technology Press Facilities.

Frasincar, F., Houben, G.-J., and Barna, P. (2006). Hpg: The hera presentation generator. *Journal of Web Engineering*, 5(2):175–200.

Gaedke, M. and Meinecke, J. (2008). The web as an application platform. In Rossi, G., Pastor, O., Schwabe, D., and Olsina, L., editors, *Web Engineering: Modelling and Implementing Web Applications*, Human-Computer Interaction Series, chapter 3, pages 33–49. Springer London.

Gellersen, H.-W., Wicke, R., and Gaedke, M. (1997). Webcomposition: an object-oriented support system for the web engineering lifecycle. *Comput. Netw. ISDN Syst.*, 29(8-13):1429–1437.

Golder, S. and Huberman, B. A. (2006). Usage patterns of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220.

Hardman, L., Aroyo, L., van Ossenbruggen, J., and Hyvonen, E. (2009). Using ai to access and experience cultural heritage. *IEEE Intelligent Systems*, 24:23–25.

Heath, T. (2010). Linked data community. `http://linkeddata.org/`.

Heckmann, D., Schwartz, T., Brandherm, B., Schmitz, M., and von Wilamowitz-Moellendorff, M. (2005). Gumo - the general user model ontology. In *Proceedings of the 10th International Conference on User Modeling (UM'05)*, number 3538 in LNAI, pages 428–432. Springer.

Henze, N., Nejdl, W., and Wolpers, M. (1999). Modeling constructivist teaching functionality and structure in the kbs hyperbook system. In *Proceedings of the 1999 conference on Computer support for collaborative learning*, CSCL '99. International Society of the Learning Sciences.

Hildebrand, M., van Ossenbruggen, J., and Hardman, L. (2006a). /facet: A Browser for Heterogeneous Semantic Web Repositories. In Cruz, I. F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., and Aroyo, L., editors, *5th International Semantic Web Conference (ISWC)*, volume 4273 of *Lecture Notes in Computer Science*, pages 272–285. Springer.

Hildebrand, M., van Ossenbruggen, J., and Hardman, L. (2006b). /facet: A browser for heterogeneous semantic web repositories. In *Proceedings of the International Semantic Web Conference (ISWC '06)*, pages 272–285, Berlin / Heidelberg. Springer.

Hillairet, G. (2007). Atl use case - odm implementation (bridging uml and owl). Eclipse Project. `http://www.eclipse.org/m2m/atl/usecases/ODMImplementation/`.

Hirtle, D., Boley, H., Grosof, B., Kifer, M., Sintek, M., Tabet, S., and Wagner, G. (2006). Schema specification of ruleml 0.91. `http://ruleml.org/0.91/`.

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., and Rudolph, S. (2009). Owl 2 web ontology language primer. W3C Recommendation 27 October 2009. `http://www.w3.org/TR/owl-primer/`.

Hockemeyer, C., Conlan, O., Wade, V., and Albert, D. (2003). Applying competence prerequisite structures for elearning and skill management. *j-jucs*, 9(12):1428–1436.

Horrocks, I., Parsia, B., Patel-Schneider, P., and Hendler, J. (2005). Semantic web architecture: Stack or two towers? In Fages, F. and Soliman, S., editors, *Principles and Practice of Semantic Web Reasoning (PPSWR 2005)*, volume 3703 of *LNCS*, pages 37–41. Springer.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004a). Swrl: A semantic web rule language combining owl and ruleml. W3C Member Submission 21 May 2004. http://www.w3.org/Submission/SWRL/.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004b). Swrl: A semantic web rule language combining owl and ruleml. http://www.w3.org/Submission/SWRL/.

Houben, G.-J., Fiala, Z., van der Sluijs, K., and Hinz, M. (2005). Building self-managing web information systems from generic components. In *Proceedings of the 1st International Workshop on Adaptive and Self-Managing Enterprise Applications (ASMEA'05), Workshop at CAiSE'05, The 17th Conference on Advanced Information Systems Engineering*, pages 053–067, Porto, Portugal. FEUP Ediçőes.

Houben, G.-J., van der Sluijs, K., Barna, P., Broekstra, J., Casteleyn, S., Fiala, Z., and Frasincar, F. (2008). *Hera*, volume 12 of *Human-Computer Interaction Series*, pages 263–301. Springer.

Hyvönen, E., Mäkelä, E., Kauppinen, T., Alm, O., Kurki, J., Ruotsalo, T., Seppälä, K., Takala, J., Puputti, K., Kuittinen, H., Viljanen, K., Tuominen, J., Palonen, T., Frosterus, M., Sinkkilä, R., Paakkarinen, P., Laitio, J., and Nyberg, K. (2008). Culturesampo – a collective memory of finnish cultural heritage on the semantic web 2.0. In *Semantic Computing Research Group, Helsinki University of Technology and University of Helsinki*.

IMS Global Learning Consortium, I. (2001). Ims learner information packaging information model specification. Final Specification. Version 1.0. http://www.imsglobal.org/profiles/lipinfo01.html.

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323.

Jaro, M. A. (1989). Advances in record linking methodology as applied to the 1985 census of tampa florida. *Journal of the American Statistical Society*, 84(406):414–420.

Karagiannis, G., Vavliakis, K., Sotiropoulou, S., Damtsios, A., Alexiadis, D., and Salpistis, C. (2009). Using signal processing and semantic web technologies to analyze Byzantine iconography. *IEEE Intelligent Systems*, 24:73–81.

Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J. (1997). Aspect-oriented programming. In Akşit, M. and Matsuoka, S., editors, *Proceedings of the European Conference on Object-Oriented Programming*, volume 1241, pages 220–242, Berlin, Heidelberg, and New York. Springer-Verlag.

Kifer, M. and Boley, H. (2010). Rif overview. Automatically generated Mediawiki page (June 2010). http://www.w3.org/2005/rules/wiki/Overview.

Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M., and Tran, L. (2004a). *Composite Capability/Preference Profiles (*CC/PP*): Structure and Vocabularies 1.0*. W3C Recommendation.

Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M. H., and Tran, L. (2004b). Composite capability/preference profiles (cc/pp): Structure and vocabularies 1.0. W3C Recommendation 15 January 2004. http://www.w3.org/TR/CCPP-struct-vocab/.

Knuth, D. E. (1973). The soundex algorithm. *The Art of Computer Programming Volume 3: Sorting and Searching*, pages 334–395.

Knutov, E., De Bra, P., and Pechenizkiy, M. (2009). Ah 12 years later: a comprehensive survey of adaptive hypermedia methods and techniques. *New Rev. Hypermedia Multimedia*, 15:5–38.

Koch, N., Knapp, A., Zhang, G., and Baumeister, H. (2008). Uml-based web engineering: An approach based on standard. In Rossi, G., Pastor, O., Schwabe, D., and Olsina, L., editors, *Web Engineering: Modelling and Implementing Web Applications*, Human-Computer Interaction Series, chapter 7, pages 157–191. Springer London.

Koch, N., Pigerl, M., Zhang, G., and Morozova, T. (2009). Patterns for the model-based development of rias. In *Proceedings of the 9th International Conference on Web Engineering (ICWE2009)*, pages 283–291, Berlin, Heidelberg. Springer-Verlag.

Koza, Y. (2004). 3gpp generic user profile (gup). stage 2: Data description method (ddm). 3GGP Proposal, June 2004. `http://www.3gpp.org/ftp/Specs/html-info/23241.htm`.

Krištofič, A. and Bieliková, M. (2005). Improving adaptation in web-based educational hypermedia by means of knowledge discovery. In *Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, HYPERTEXT '05, pages 184–192, New York, NY, USA. ACM.

Leonardi, E., Houben, G.-J., van der Sluijs, K., Hidders, J., Herder, E., Abel, F., Krause, D., and Heckmann, D. (2009). User profile elicitation and conversion in a mashup environment. In *Proceedings of the International Workshop on Lightweight Integration on the Web (ComposableWeb'09)*.

Manola, F. and Miller, E. (2004). Rdf primer. W3C Recommendation 10 February 2004. `http://www.w3.org/TR/rdf-primer/`.

Manolescu, I., Brambilla, M., Ceri, S., Comai, S., and Fraternali, P. (2005). Model-driven design and deployment of service-enabled web applications. *ACM Trans. Internet Technol.*, 5(3):439–479.

Meinecke, J., Gaedke, M., and Thiele, C. (2007). Enabling architecture changes in distributed web-applications. In *Proceedings of the 2007 Latin American Web Conference*, LA-WEB '07, pages 92–99, Washington, DC, USA. IEEE Computer Society.

Melenhorst, M., Grootveld, M., van Setten, M., and Veenstra, M. (2008). Tag-based information retrieval of video content. In *Proceeding of the 1st international conference on Designing interactive user experiences for TV and video*, pages 31–40. ACM Press.

Mika, P. (2005). Ontologies are us: A unified model of social networks and semantics. In *Proceedings of the International Semantic Web Conference*, pages 522–536, Heidelberg. Springer.

Miles, A. and Bechhofer, S. (2009). Simple knowledge organization system. `http://www.w3.org/TR/skos-reference/`.

Millard, D. E. and Ross, M. (2006). Web 2.0: hypertext by any other name? In *HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia*, pages 27–30, New York, NY, USA. ACM.

Morgan, D. (1997). *Focus Groups As Qualitative Research*. Number 16 in Qualitative Research Methods Series. Sage Publications, second edition edition.

Nagy, I., Bergmans, L. M. J., and Akşit, M. (2005). Composing aspects at shared join points. In *Proceedings of International Conference NetObjectDays(NODe2005)*, volume P-69 of *Lecture Notes in Informatics*, pages 19–38, Bonn, Germany. Gesellschaft für Informatik (GI).

Niederhausen, M., van der Sluijs, K., Hidders, J., Leonardi, E., Houben, G.-J., and Meißner, K. (2009). Harnessing the power of semantics-based, aspect-oriented adaptation for amacont. In *Proceedings of the 9th International Conference on Web Engineering (ICWE 2009)*, volume 5648 of *Lecture Notes in Computer Science*, pages 106–120. Springer.

Nunes, D. A. and Schwabe, D. (2006). Rapid prototyping of web applications combining domain specific languages and model driven design. In *Proceedings of the 6th international conference on Web engineering*, ICWE '06, pages 153–160, New York, NY, USA. ACM.

Passepartout (2005-2007). Itea passepartout project. http://wwwis.win.tue.nl/∼ppartout.

Pastor, O., Gómez, J., Insfrán, E., and Pelechano, V. (2001). The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. *Inf. Syst.*, 26(7):507–534.

Petersen, F., Brown, W., and Pluke, M. (2005). Eg 202 325, user profile management. ETSI Recommendation, Oktober 2005. `http://portal.etsi.org/stfs/stf_homepages/stf342/eg_202325v010101p.pdf`.

Protégé (2011). The protégé ontology editor and knowledge acquisition system. `http://protege.stanford.edu/`.

Prud'hommeaux, E. and Seaborne, A. (2008). Sparql query language for rdf. W3C Recommendation 15 January 2008. `http://www.w3.org/TR/rdf-sparql-query/`.

Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350.

Rossi, G. and Schwabe, D. (2008). Modeling and implementing web applications with oohdm. In Rossi, G., Pastor, O., Schwabe, D., and Olsina, L., editors, *Web Engineering: Modelling and Implementing Web Applications*, Human-Computer Interaction Series, chapter 6, pages 109–155. Springer London.

Rossi, G., Schwabe, D., Olsina, L., and Pastor, O. (2008). *Web Engineering: Modelling and Implementing Web Applications*, chapter Overview of Design Issues for Web Applications Development, pages 49–63. Human-Computer Interaction Series. Springer London.

Ruotsalo, T., Aroyo, L., and Schreiber, G. (2009). Knowledge-based linguistic annotation of digital cultural heritage collections. *Intelligent Systems, IEEE*, 24(2):64–75.

Schauerhuber, A., Wimmer, M., Schwinger, W., Kapsammer, E., and Retschitzegger, W. (2007). Aspect-oriented modeling of ubiquitous web applications: The aspectwebml approach. In *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 569–576, Washington, DC, USA. IEEE Computer Society.

Schmitz, P., Yu, J., and Santangeli, P. (1998). Timed interactive multimedia extensions for html (html+time), extending smil into the web browser. W3C Submission 18 September 1998. `http://www.w3.org/TR/NOTE-HTMLplusTIME`.

Schraefel, M., Smith, D. A., Owens, A., Russell, A., Harris, C., and Wilson, M. (2005). The evolving mspace platform: leveraging the semantic web on the trail of the memex. In *HYPERTEXT '05: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, pages 174–183, New York, NY, USA. ACM.

Schwabe, D., Szundy, G., de Moura, S. S., and Lima, F. (2004). Design and implementation of semantic web applications. In *WWW Workshop on Application Design, Development and Implementation Issues in the Semantic Web*.

Schwinger, W., Retschitzegger, W., Schauerhuber, A., Kappel, G., Wimmer, M., Pr/"oll, B., Castro, C. C., Casteleyn, S., Troyer, O. D., Fraternali, P., abd Franca Garzotto, I. G., Ginige, A., Houben, G.-J., Koch, N., Moreno, N., Pastor, O., Paolini, P., Ferragud, V. P., Rossi, G., Schwabe, D., Tisi, M., Vallecillo, A., van der Sluijs, K., and Zhang, G. (2008). A survey on web modeling approaches for ubiquitous web applications. *International Journal of Web Information Systems*, 4(3):234–305.

Shadbolt, N., Berners-Lee, T., and Hall, W. (2006). The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101.

Simile (2008). Rdfizers. The RDFizer project is directory of tools for converting various data formats into RDF. `http://simile.mit.edu/wiki/RDFizers`.

Sirin, E. and Parsia, B. (2004). Pellet: An owl dl reasoner. In Haarslev, V. and Möller, R., editors, *Proceedings of the 2004 International Workshop on Description Logics (DL2004)*, volume 104 of *CEUR Workshop Proceedings*.

Specia, L. and Motta, E. (2007). Integrating folksonomies with the semantic web. In *ESWC '07: Proceedings of the 4th European conference on The Semantic Web*, pages 624–639, Berlin, Heidelberg. Springer-Verlag.

Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., and Montero, F. (2005). A transformational approach for multimodal web user interfaces based on usixml. In *Proceedings of the 7th international conference on Multimodal interfaces*, ICMI '05, pages 259–266, New York, NY, USA. ACM.

Sutterer, M., Coutand, O., Droegehorn, O., David, K., and van der Sluijs, K. (2007). Managing and delivering context-dependent user preferences in ubiquitous computing environments. In *Proceedings of the 2007 International Symposium on Applications and the Internet Workshops (SAINTW'07)*, pages 4–4, Hiroshima, Japan. IEEE CS Press.

Tafazolli, R., editor (2006). *Technologies for the Wireless Future: Wireless World Research Forum (WWRF)*, volume 2. Wiley.

Taylor, J. (2008). Introducing the freebase rdf service. Freebase Blog. `http://blog.freebase.com/2008/10/30/introducing_the_rdf_service/`.

Thiran, P., Hainaut, J.-L., and Houben, G.-J. (2005). Database wrappers development: Towards automatic generation. In *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering (CSMR'05)*, pages 207–216. IEEE CS Press.

Trujillo, S., Batory, D., and Diaz, O. (2007). Feature oriented model driven development: A case study for portlets. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 44–53, Washington, DC, USA. IEEE Computer Society.

TV-Anytime (2003). The TV-Anytime Forum, Requirement Series: RQ001v2.0, TV140. Available at: ftp://tva:tva@ftp.bbc.co.uk/pub/Plenary/0-Plenary.html.

van der Sluijs, K., Hidders, J., Leonardi, E., and Houben, G.-J. (2009). Gal: A generic adaptation language for describing adaptive hypermedia. In *Proceedings of the International Workshop on Dynamic and Adaptive Hypertext: Generic Frameworks, Approaches and Techniques (DAH'09)*.

van der Sluijs, K. and Houben, G.-J. (2005). Towards a generic user model component. In *Proceedings of the 1st Workshop on Personalization on the Semantic Web (PerSWeb'05), Workshop at the 10th International Conference on User Modeling (UM'2005)*, pages 043–052, Edinburgh, Scotland.

van der Sluijs, K. and Houben, G.-J. (2006a). A generic component for exchanging user models between web-based systems. *International Journal of Continuing Engineering Education and Life-Long Learning (IJCEELL)*, 16(1/2):64–76.

van der Sluijs, K. and Houben, G.-J. (2006b). Query discovery for translating user model data. In *Proceedings of the 4th International Workshop on Applications of Semantic Web Technologies for E-Learning (SW-EL'06), Workshop at the 4th International Conference on Adaptive Hypermedia (AH'2006)*, pages 109–111, Dublin, Ireland.

van der Sluijs, K. and Houben, G.-J. (2008a). Metadata-based access to cultural heritage collections: the rhce use case. In *Proceedings of the 2nd International Workshop on Personalized Access to Cultural Heritage (PATCH'2008)*, pages 15–25, Hannover, Germany.

van der Sluijs, K. and Houben, G.-J. (2008b). Relating user tags to ontological information. In *Proceedings of the 5th International Workshop on Ubiquitous User Modeling (UbiqUM'2008)*, Gran Canaria, Spain.

van der Sluijs, K. and Houben, G.-J. (2008c). Tagging and the semantic web in cultural heritage. *ERCIM News - Special: The Future Web*, (72):22–23.

van der Sluijs, K. and Houben, G.-J. (2009). Automatic generation of semantic metadata as basis for user modeling and adaptation. In *Advances in Ubiquitous User Modelling*, volume 5868 of *Lecture Notes in Computer Science*, pages 73–93. Springer.

van der Sluijs, K. and Houben, G.-J. (2010). Exploiting user tags to build a semantic cultural heritage portal. *IEEE Intelligent Systems*, 25:84–92.

van der Sluijs, K., Houben, G.-J., Broekstra, J., and Casteleyn, S. (2006). Hera-s - web design using sesame. In *Proceedings of the 6th International Conference on Web Engineering (ICWE'06)*, pages 337–345, Palo Alto, California, USA. ACM.

van der Sluijs, K., Houben, G.-J., Leonardi, E., and Hidders, J. (2010). Hera: Engineering web applications using semantic web-based models. In de Virgilio, R., Giunchiglia, F., and Tanca, L., editors, *Semantic Web Information Management*, pages 521–544. Springer Berlin Heidelberg. 10.1007/978-3-642-04329-1_22.

van der Sluijs, K. and Höver, K. M. (2009a). Integrating adaptive functionality in a LMS. *International Journal of Emerging Technologies in Learning (iJET)*, 4(4):46–50.

van der Sluijs, K. and Höver, K. M. (2009b). Integrating adaptive functionality in a lms. In *Proceedings of the International Conference on E-Learning in the Workplace (ICELW2009)*.

van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, 35:26–36.

van Setten, M. (2005). Supporting people in finding information: Hybrid recommender systems and goal-based structuring. *Telematica Instituut Fundamental Research Series, No.016 (TI/FRS/016). Universal Press.*

van Setten, M., Brussee, R., van Vliet, H., Gazendam, L., van Houten, Y., and Veenstra, M. (2006). On the importance of "who tagged what". In *Workshop on the Social Navigation and Community-Based Adaptation Technologies*.

Vdovjak, R., Frasincar, F., Houben, G. J., and Barna, P. (2003). Engineering semantic web information systems in Hera. *Journal of Web Engineering*, 2(1-2):3–26.

Verheijen, W. (2008). Efficient query processing in distributed rdf databases. Master's thesis, Technische Universiteit Eindhoven. /urlhttp://alexandria.tue.nl/extra1/afstversl/wsk-i/verheijen2008.pdf.

Wartena, C. and Brussee, R. (2008). Topic detection by clustering keywords. In *DEXA Workshops*, pages 54–58.

Wireless Application Protocol Forum (2001). User agent profiling specification. WAP-248-UAProf-20011020-a, Version 20-Oct-2001.

Witte, R., Krestel, R., Kappler, T., and Lockemann, P. (2009). Converting a historical encyclopedia of architecture into a semantic knowledge base. *IEEE Intelligent Systems*, 25:58–67.

Yee, K.-P., Swearingen, K., Li, K., and Hearst, M. (2003). Faceted metadata for image search and browsing. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '03)*, pages 401–408, New York, NY, USA. ACM Press.

List of Publications

This is a list of the publications which are the basis for this thesis. The items in the list refer to the publication details in the bibliography.

- **Book Chapters**
  - [De Bra et al., 2012]
  - [van der Sluijs et al., 2010]
  - [Houben et al., 2008]
- **Journal Papers**
  - [van der Sluijs and Houben, 2010]
  - [van der Sluijs and Höver, 2009a]
  - [van der Sluijs and Houben, 2009]
  - [Casteleyn et al., 2009]
  - [van der Sluijs and Houben, 2008c]
  - [Schwinger et al., 2008]
  - [van der Sluijs and Houben, 2006a]
- **Conference Papers**:
  - [De Bra et al., 2010a]
  - [van der Sluijs and Höver, 2009b]
  - [Niederhausen et al., 2009]
  - [Bellekens et al., 2008]
  - [Bellekens et al., 2007]
  - [van der Sluijs et al., 2006]
  - [Casteleyn et al., 2006a]
  - [Casteleyn et al., 2006b]
- **Workshop Papers**
  - [van der Sluijs et al., 2009]
  - [Leonardi et al., 2009]
  - [Abel et al., 2009]

- [van der Sluijs and Houben, 2008a]
- [van der Sluijs and Houben, 2008b]
- [Sutterer et al., 2007]
- [van der Sluijs and Houben, 2006b]
- [van der Sluijs and Houben, 2005]
- [Houben et al., 2005]

List of Figures

Abbreviations

ALE Adaptive Learning Environment 133
AM Application Model 27
AMP AM (instance) page 28
CAM Conceptual Adaptation Model 136
CSS Cascading Style Sheets 40
DM Domain Model 27
DMOZ Open Directory Project 17
CM Context Model 28
FS Filter Service 109
GAL Generic Adaptation Language 136
GALE GRAPPLE adaptive learning environment 134
GAT GRAPPLE authoring tools 137
GEB GRAPPLE Event Bus 136
GDR Grapple Derivation Rules 140
GTAA Common Thesaurus for Audiovisual Archives 116
GUMF Generic User Model Framework 137
GUP Generic User Profile 131
IMDb International Movie Database 29
IMEI International Mobile Equipment Identifier 128
LMS Learning Management System 133
MDA Model-Driven Architecture 23
MDE Model-driven engineering 24
MDWE Model-Driven Web Engineering 1
OOHDM Object-Oriented Hypermedia Design Model 19
OOWS Object-Oriented Web Solutions Approach 19
OS Ontology Service 109
OWL Web Ontology Language 13
PCF Personal Context Function 128
PIM Platform-Independent Model 23

PM Presentation Model . . . 28
PPF Personal Profiling Function . . . 128
PSM Platform-Specific Models . . . 23
RDF Resource Description Framework . . . 2
RDF(S) Resource Description Framework (Schema) . . . 12
RDFS Resource Description Framework Schema . . . 11
RHCe Regionaal Historisch Centrum Eindhoven . . . 93
RIF Rule Interchange Format . . . 15
SeAL Semantics-based Aspect-Oriented Adaptation Language . . . 52
SHDM Semantic Hypermedia Design Method . . . 21
SKOS Simple Knowledge Organization System . . . 84
SMIL Synchronized Multimedia Integration Language . . . 20
SPARQL SPARQL Protocol and RDF Query Language . . . 15
STL SPARQL Transformation Language . . . 80
SWRL Semantic Web Rule Language . . . 15
Turtle Terse RDF Triple Language . . . 12
UID User Interaction Diagrams . . . 21
UM User Model . . . 156
UMS User Model Service . . . 108
UWE UML-based Web Engineering approach . . . 19
URI Uniform Resource Identifier . . . 9
URL Uniform Resource Locator . . . 9
WebML Web Modeling Language . . . 19
WML Wireless Markup Language . . . 20
WIS Web Information Systems . . . 2
XSLT EXtensible Stylesheet Language . . . 25
XML Extensible Markup Language . . . 9

Summary

Model Driven Design and Data Integration in Semantic Web Information Systems

The Web is quickly evolving in many ways. It has evolved from a Web of documents into a Web of applications in which a growing number of designers offer new and interactive Web applications with people all over the world. However, application design and implementation remain complex, error-prone and laborious. In parallel there is also an evolution from a Web of documents into a Web of 'knowledge' as a growing number of data owners are sharing their data sources with a growing audience. This brings the potential new applications for these data sources, including scenarios in which these datasets are reused and integrated with other existing and new data sources. However, the heterogeneity of these data sources in syntax, semantics and structure represents a great challenge for application designers. The Semantic Web is a collection of standards and technologies that offer solutions for at least the syntactic and some structural issues. If offers semantic freedom and flexibility, but this leaves the issue of semantic interoperability.

In this thesis we present Hera-S, an evolution of the Model Driven Web Engineering (MDWE) method Hera. MDWEs allow designers to create data centric applications using models instead of programming. Hera-S especially targets Semantic Web sources and provides a flexible method for designing personalized adaptive Web applications. Hera-S defines several models that together define the target Web application. Moreover we implemented a framework called Hydragen, which is able to execute the Hera-S models to run the desired Web application. Hera-S' core is the Application Model (AM) in which the main logic of the application is defined, i.e. defining the groups of data elements that form logical units or subunits, the personalization conditions, and the relationships between the units. Hera-S also uses a so-called Domain Model (DM) that describes the content and its structure. However, this DM is not Hera-S specific, but instead allows any Semantic Web source representation as its DM, as long as its content can be queried by the standardized Semantic Web query language SPARQL. The same holds for the User Model (UM). The UM can be used for personalization conditions, but also as a source of user-related content if necessary. In fact, the difference between DM and UM is conceptual as their implementation within Hydragen is the same. Hera-S also defines a presentation model (PM) which defines presentation details of elements like order and style. In order to help designers with building their Web applications we have introduced a toolset, Hera Studio, which allows to build the different models graphically. Hera Studio also provides some additional functionality like model checking and deployment of the models in Hydragen.

Both Hera-S and its implementation Hydragen are designed to be flexible regarding the user of models. In order to achieve this Hydragen is a stateless engine that queries for relevant information from the models at every page request. This allows the models and data to be changed in the datastore during runtime. We show that one way to exploit this flexibility is by applying aspect-orientation to the AM. Aspect-orientation allows us

to dynamically inject functionality that pervades the entire application. Another way to exploit Hera-S' flexibility is in reusing specialized components, e.g. for presentation generation. We present a configuration of Hydragen in which we replace our native presentation generation functionality by the AMACONT engine. AMACONT provides more extensive multi-level presentation generation and adaptation capabilities as well aspect-orientation and a form of semantic based adaptation.

Hera-S was designed to allow the (re-)use of any (Semantic) Web datasource. It even opens up the possibility for data integration at the back end, by using an extendible storage layer in our database of choice Sesame. However, even though theoretically possible it still leaves much of the actual data integration issue. As this is a recurring issue in many domains, a broader challenge than for Hera-S design only, we decided to look at this issue in isolation. We present a framework called Relco which provides a language to express data transformation operations as well as a collection of techniques that can be used to (semi-)automatically find relationships between concepts in different ontologies. This is done with a combination of syntactic, semantic and collaboration techniques, which together provide strong clues for which concepts are most likely related.

In order to prove the applicability of Relco we explore five application scenarios in different domains for which data integration is a central aspect. This includes a cultural heritage portal, χ Explorer, for which data from several datasources was integrated and was made available by a mapview, a timeline and a graph view. χ Explorer also allows users to provide metadata for objects via a tagging mechanism. Another application is SenSee: an electronic TV-guide and recommender. TV-guide data was integrated and enriched with semantically structured data from several sources. Recommendations are computed by exploiting the underlying semantic structure. ViTa was a project in which several techniques for tagging and searching educational videos were evaluated. This includes scenarios in which user tags are related with an ontology, or other tags, using the Relco framework. The MobiLife project targeted the facilitation of a new generation of mobile applications that would use context-based personalization. This can be done using a context-based user profiling platform that can also be used for user model data exchange between mobile applications using technologies like Relco. The final application scenario that is shown is from the GRAPPLE project which targeted the integration of adaptive technology into current learning management systems. A large part of this integration is achieved by using a user modeling component framework in which any application can store user model information, but which can also be used for the exchange of user model data.

Samenvatting

Het Web evolueert in veel opzichten. Zo heeft het zich ontwikkeld van een Web van documenten naar een Web van applicaties waarin een groeiend aantal ontwerpers nieuwe en interactieve Web-toepassingen kan maken voor een wereldwijd publiek. Maar het ontwerpen en implementeren van Web applicaties blijft een moeizame bezigheid die ontzettend veel tijd kost en waarin fouten snel worden gemaakt. Naast het Web van applicaties is er een parallelle evolutie van een Web van documenten naar een Web van kennis, waarbij steeds meer mensen en instituten hun data beginnen te delen met een groeiend publiek. Hieruit volgen nieuwe kansen voor applicaties die deze beschikbaar gestelde databronnen gebruiken, inclusief scenario's waarin deze bronnen worden herbruikt en geïntegreerd met andere bestaande en nieuwe bronnen. Maar de heterogeniteit van deze bronnen qua syntax, betekenis en structuur zorgt voor nieuwe uitdagingen voor ontwerpers. Het Semantische Web is een verzameling standaarden en technologieën dat standaardoplossingen biedt voor in ieder geval de syntax en de structuur van de data. Het biedt semantische vrijheid en flexibiliteit, maar semantische interoperabiliteit blijft daarmee wel een probleem.

In dit proefschrift presenteren we Hera-S, een verdere ontwikkeling van de model-gedreven Web-ontwerpmethode (MDWE) genaamd Hera. MDWE's laten ontwerpers data-centrische applicaties ontwerpen middels modellen de plaats van programmeren. Hera-S is speciaal ontworpen voor Semantische Web-bronnen en levert een flexibele methode voor het ontwerpen van gepersonaliseerde adaptieve applicaties. Hera-S definieert verschillende modellen die samen een volledige Web-applicatie beschrijven. We hebben daarnaast ook een platform geïmplementeerd dat Hydragen heet en dat Hera-S modellen kan uitvoeren als Web-applicatie. De kern van Hera-S is het applicatie model (AM) dat de belangrijkste applicatie-logica vastlegt, waarin groepen van data-elementen worden gedefinieerd in termen van eenheden en subeenheden, de personalisatie condities worden vastgelegd, en de relaties tussen de verschillende eenheden worden vastgelegd. Hera-S baseert zich in het AM op een zogenaamd domein model (DM) waarin de eigenlijke inhoud van de Web-applicatie wordt vastgelegd. Het DM is echter niet Hera-S specifiek, aangezien elke Semantische Webbron kan worden gebruikt als DM, zolang deze maar kan worden bevraagd via de gestandaardiseerde Semantisch Web zoek-taal genaamd SPARQL. Hetzelfde geldt voor het gebruikersmodel (UM). Het UM kan gebruikt worden voor personalisatiecondities, maar net zo goed ook voor gepersonaliseerde inhoud als dat nodig is. In feite is er alleen een conceptueel verschil tussen het DM en het UM, want de eigenlijke implementaties in Hydragen zijn vergelijkbaar. Hera-S definieert ook een presentatiemodel (PM) dat presentatie details definieert zoals volgorde en stijl van de elementen. Om ontwikkelaars te helpen met het bouwen van hun Web-applicaties hebben we daarvoor een toolset ontwikkeld, Hera Studio, die kan worden gebruikt om de verschillende modellen met grafische ondersteuning te bouwen. Hera Studio bevat daarnaast nog aanvullende mogelijkheden zoals modelverificatie en de mogelijkheid om

het model rechtstreeks in Hydragen te activeren.

Zowel Hera-S als de Hydragen implementatie zijn ontwikkeld om flexibel om te gaan met modellen. Zo is Hydragen een toestandsloze machine die bij elk paginaverzoek gaat zoeken naar de relevante informatie in de betreffende modellen. Hierdoor kunnen modellen en databronnen worden aangepast terwijl de applicatie in gebruik is. Een manier waarop we dit exploiteren is bij het toepassen van aspect-oriëntatie op het AM. Aspect-orië ntatie geeft de mogelijkheid om dynamisch functionaliteit te injecteren dat de gehele Web applicatie beïnvloedt. Nog een manier om de flexibiliteit van Hera-S te exploiteren is middels het hergebruik van gespecialiseerde componenten, bijvoorbeeld voor presentatie-generatie. We schetsen bijvoorbeeld een configuratie van Hydragen waarin we onze standaard presentatiegeneratie-module vervangen door het AMACONT platform. AMACONT heeft uitgebreidere meerlaagse presentatiegeneratie mogelijkheden met de optie voor adaptatie, aspect-oriëntatie en semantisch gebaseerde adaptatie. Hera-S was speciaal ontwikkeld om Semantisch Web databronnen te (her)gebruiken. Het staat zelfs data-integratie toe aan de achterkant van het platform, door het gebruik van een uitbreidbare opslaglaag in onze achterliggende database, Sesame. Deze mogelijkheid neemt het eigenlijke data integratie probleem nog niet volledig weg. Aangezien dit een steeds terugkerend probleem is in vele domeinen, en het dus een Hera-S overstijgend probleem is, hebben we ervoor gekozen om dit probleem nader en in isolatie te onderzoeken. We presenteren daarvoor een nieuw platform, genaamd Relco, dat een taal levert die kan worden gebruikt om datatransformatie uit te drukken en daarnaast ook een collectie van technieken aanbiedt die kunnen worden gebruikt voor het (semi-)automatisch vinden van overeenkomsten tussen concepten in verschillende ontologieë n. Dit wordt gedaan met een combinatie van syntactische, semantische en collaboratieve technieken, die samen sterke hints opleveren voor welke concepten hoogstwaarschijnlijk aan elkaar verwant zijn.

Om de toepasbaarheid aan te tonen van Relco verkennen we het gebruik ervan in vijf verschillende toepassingsscenario's in verschillende domeinen, waarin data-integratie steeds een centraal aspect vormt. Een van die toepassingen is een cultuurhistorische applicatie, χ Explorer, waarin data van verschillende bronnen is geïntegreerd en toegankelijk is gemaakt met een kaartinterface, een tijdlijn en een graaf-gebaseerde interface. χ Explorer biedt ook een mechanisme waarmee gebruikers metadata kunnen invoeren via een tagging-mechanisme. Een andere applicatie is SenSee: een elektronische programmagids en aanbeveler. Daarvoor is programmagidsinformatie verrijkt met gestructureerde data uit verschillende bronnen. Aanbevelingen worden daarbij berekend door gebruik te maken van die onderliggende semantische structuur. ViTa was een project waarin verschillende 'label en zoek' technieken werden geëvalueerd voor het ontsluiten van educatieve video's. In een van de scenario's daarin werden labels van gebruikers gerelateerd aan concepten in een ontologie of aan andere labels met behulp van het Relco raamwerk. Het MobiLife project was er op gericht om een nieuwe generatie van mobiele applicaties mogelijk te maken die gebruik maken van context-gebaseerde personalisatie. Dit kan worden gedaan met een context gebaseerd gebruikersmodelleerplatform dat ook kan worden gebruikt voor het uitwisselen van gebruikersgegevens met technologieën zoals Relco. Het laatste scenario dat we hebben verkend is in het kader van het GRAPPLE project, waarin we tot doel hadden om adaptieve technologieën te integreren met de huidige systemen voor training en onderwijs. Een groot deel van deze integratie werd bewerkstelligd door middel van een gebruiksmodelleerplatform waarin elke applicatie gebruikersgegevens kan opslaan en dat ook kan worden gebruikt voor het uitwisselen van gebruikersgegevens.

Curriculum Vitae

Kees van der Sluijs was born on 22 July 1978 in Maasdriel, the Netherlands. After completing pre-university education at the Stedelijk Gymnasium 's-Hertogenbosch he started his study onin technical computer science at the Technische Universiteit Eindhoven. After writing a master thesis on the subject of "Searching the Semantic Web" he received his M.Sc. degree in 2004.

Subsequently Kees started his Ph.D. research at the same department of Information Systems at the Technische Universiteit Eindhoven. His research was part of the Hera research program that combines research on Web information systems, including design and engineering of WIS, automated adaptive presentation generation, Semantic Web technology, data integration and distribution, user profiling, enriched information delivery and (Semantic) Web services. During his Ph.D. research he participated in several large national and international projects. He has published more than two dozen publications, including two bookchapters, journal publications, and publications in international conferences and workshop proceedings. Kees also has teaching experience. He has been teaching within several courses and has supervised a total of seven students for their Master's projects and subsequent theses.

Kees has worked as the project manager of the EU FP7 GRAPPLE Project between 2008 and 2011,. The main objective of GRAPPLE was supporting life-long learning by means of a personalized and adaptive technology-enhanced learning (TEL) environment that integrates with major learning management systems (LMSs) using a service-oriented (Web) framework approach. GRAPPLE combined expertise scattered among some of the core players in adaptive TEL in Europa, with the aim to integrate the benefits of the key technologies in a modular way and also integrate or interface with large existing open source and commercial LMSs.

Currently Kees works as an IT-specialist at the Ministry of the Interior and Kingdom Relations (BZK), one of the eleven ministries of the Dutch central government.

Kees' research interests include Web and Semantic Web technologies, Information Systems, Distributed Databases, Web 2.0, Adaptive Hypermedia, Personalization and User Modeling.

SIKS Dissertatiereeks

====
1998
====

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects
1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information
1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective
1998-4 Dennis Breuker (UM)
Memory versus Search in Games
1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

====
1999
====

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling;
Automated modelling of Quality Change of Agricultural Products
1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets
1999-3 Don Beal (UM)
The Nature of Minimax Search
1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects
1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems
1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems
1999-7 David Spelt (UT)
Verification support for object database design
1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

====
2000
====

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance
2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management
2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie;
een procesbenadering en actorperspectief.
2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design
2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.
2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication
2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management
2000-8 Veerle Coup (EUR)
Sensitivity Analyis of Decision-Theoretic Networks
2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization
2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture
2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

====
2001
====

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks
2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models
2001-3 Maarten van Someren (UvA)
Learning as problem solving
2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with
Instance-Based Boundary Sets
2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style
2001-6 Martijn van Welie (VU)
Task-based User Interface Design
2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization
2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.
2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models,
Views of Packages as Classes 2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice
BRAHMS: a multiagent modeling and simulation language
for work practice analysis and design
2001-11 Tom M. van Engers (VUA)
Knowledge Management:
The Role of Mental Models in Business Systems Design

====
2002
====

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis
2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections
2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval
2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining
2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments
inhabited by Privacy-concerned Agents
2002-06 Laurens Mommers (UL)
Applied legal epistemology;
Building a knowledge-based ontology of the legal domain
2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications
2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative
E-Commerce Ideas
2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems
2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble
2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications
2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems
2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications
2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and
Verifying Multi-Agent Systems
2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling
2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications
2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance

====
2003
====

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments
2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems
2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy
2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology
2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach
2003-06 Boris van Schooten (UT)
Development and specification of virtual environments
2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks
2003-08 Yongping Ran (UM)
Repair Based Scheduling
2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour
2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture
2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks
2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval
2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models
2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations
2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems
2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses
2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing
2003-18 Levente Kocsis (UM)
Learning Search Decisions

====
2004
====

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic
2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business
2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving
2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures
2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity
2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques
2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes
2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise
2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning
2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects
2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies
2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents
2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play
2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining
2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning
2004-17 Mark Winands (UM)
Informed Search in Complex Games
2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models
2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval
2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

====
2005
====

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications
2005-02 Erik van der Werf (UM))
AI techniques for the game of Go
2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language
2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data
2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing
2005-06 Pieter Spronck (UM)
Adaptive Game AI
2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems
2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications
2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages
2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments
2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search
2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry
2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen
2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics
2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes
2005-16 Joris Graaumans (UU)
Usability of XML Query Languages
2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components
2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks
2005-19 Michel van Dartel (UM)
Situated Representation
2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives
2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by
Exploiting Application Semantics

====
2006
====

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting
2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations
2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems
2006-04 Marta Sabou (VU)
Building Web Service Ontologies
2006-05 Cees Pierik (UU)

Validation Techniques for Object-Oriented Proof Outlines
2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools
for Graphical Service Modeling
2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering
2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web
2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion
2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems
2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types
2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts
2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents
2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change
2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain
2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks
2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device
2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing
2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach
2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining
2006-21 Bas van Gils (RUN)
Aptness on the Web
2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation
2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web
2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources
2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC
2006-26 Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval
2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories
2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval

====
2007
====

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures
2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach
2007-03 Peter Mika (VU)
Social Networks and the Semantic Web
2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach
2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy:
a Legislative Framework for Agent-enabled Surveillance
2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs
2007-07 Natasa Jovanovic' (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings
2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations
2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation
2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System
2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support:
A Rational Approach to Dynamic Decision-Making under Uncertainty
2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology
2007-14 Niek Bergboer (UM)
Context-Based Image Analysis
2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model
2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in
Institutions and Organizations for Multi-agent Systems
2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice
2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations
2007-19 David Levy (UM)
Intimate relationships with artificial partners
2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network
2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of
broadband internet in the Netherlands between 2001 and 2005
2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns
2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems
2007-24 Georgina Ramírez Camps (CWI)
Structural Features in XML Retrieval
2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement
====
2008
====

2008-01 Katalin Boer-Sorbán (EUR)
Agent-Based Simulation of Financial Markets: A modular,continuous-time approach
2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations
2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach
2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration
2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems
from a cost perspective
2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective
2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning
2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference
2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective
2008-10 Wauter Bosma (UT)
Discourse oriented summarization
2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach
2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation
2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks
2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort
2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms
for the Markov Decision Process Framework in First-Order Domains.
2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective
2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises
2008-18 Guido de Croon (UM)

Adaptive Active Vision
2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search
2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie
van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven
2008-21 Krisztian Balog (UVA)
People Search in the Enterprise
2008-22 Henk Koning (UU)
Communication of IT-Architecture
2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia
2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching
2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair
using Spender-signed Currency
2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled
2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design
2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks
2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans
2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content
2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval
2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes
2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues
2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining
2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

====
2009
====

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models
2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques
2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT
2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering
2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks -
Based on Knowledge, Cognition, and Quality
2009-06 Muhammad Subianto (UU)
Understanding Classification
2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion
2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments
2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems
2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications
2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web
2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
Operating Guidelines for Services
2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems
2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies
(making ontologies work in e-science with ONTO-SOA)
2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess
2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data
2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System
2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets
2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making
2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification
2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence
2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment
2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations
2009-25 Alex van Ballegooij (CWI)
"RAM: Array Database Management through Relational Mapping"
2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services
2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web
2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models
2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications
2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage
2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text
2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors
2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?
2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach
2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling
2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks
2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks
2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context
2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets
2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language
2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings
2009-42 Toine Bogers
Recommender Systems for Social Bookmarking
2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients
2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations
2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful
2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion

====
2010
====

2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter
2010-02 Ingo Wassink (UT)
Work flows in Life Science
2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents
2010-04 Olga Kulyk (UT)

Do You Know What I Know? Situational Awareness of Co-located Teams
in Multidisplay Environments
2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems
2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI
2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance
2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software.
Protecting user freedoms in a world of software communities and eGovernments
2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging
2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children 2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning
2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis
2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques
2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration
2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models
2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice
2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications
2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation
2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems
2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative
2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation
2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to Heterogeneous Linked Data
2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions
2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies
2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective
2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines
2010-27 Marten Voulon (UL)
Automatisch contracteren
2010-28 Arne Koopman (UU)
Characteristic Relational Patterns
2010-29 Stratos Idreos(CWI)
Database Cracking: Towards Auto-tuning Database Kernels
2010-30 Marieke van Erp (UvT)
Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval
2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web
2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems
2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval
2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions
2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval
2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning; Facilitating competence development
through a learning path specification
2010-37 Niels Lohmann (TUE)
Correctness of services and their composition
2010-38 Dirk Fahland (TUE)
From Scenarios to components
2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents
2010-40 Mark van Assem (VU)

Converting and Integrating Vocabularies for the Semantic Web
2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search
2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach
2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies
2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain
2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services
2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment
2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples
2010-48 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets
2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions
2010-50 Bouke Huurnink (UVA)
Search in Audiovisual Broadcast Archives
2010-51 Alia Khairia Amin (CWI)
Understanding and supporting information seeking tasks in multiple sources
2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention
2010-53 Edgar Meij (UVA)
Combining Concepts and Language Models for Information Access

====
2011
====

2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models
2011-02 Nick Tinnemeier(UU)
Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language
2011-03 Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems
2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms
2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.
2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage
2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction
2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues 2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning
2011-10 Bart Bogaert (UvT)
Cloud Content Contention
2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective
2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining
2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling
2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets
2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval
2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity
2011-17 Jiyin He (UVA)
Exploring Topic Structure: Coherence, Diversity and Relatedness
2011-18 Mark Ponsen (UM)
Strategic Decision-Making in complex games
2011-19 Ellen Rusman (OU)
The Mind ' s Eye on Personal Profiles
2011-20 Qing Gu (VU)

Guiding service-oriented software engineering - A view-based approach
2011-21 Linda Terlouw (TUD)
Modularization and Specification of Service-Oriented Systems
2011-22 Junte Zhang (UVA)
System Evaluation of Archival Description and Access
2011-23 Wouter Weerkamp (UVA)
Finding People and their Utterances in Social Media
2011-24 Herwin van Welbergen (UT)
Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior
2011-25 Syed Waqar ul Qounain Jaffry (VU)
Analysis and Validation of Models for Trust Dynamics
2011-26 Matthijs Aart Pontier (VU)
Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots
2011-27 Aniel Bhulai (VU)
Dynamic website optimization through autonomous management of design patterns
2011-28 Rianne Kaptein(UVA)
Effective Focused Retrieval by Exploiting Query Context and Document Structure
2011-29 Faisal Kamiran (TUE)
Discrimination-aware Classification
2011-30 Egon van den Broek (UT)
Affective Signal Processing (ASP): Unraveling the mystery of emotions
2011-31 Ludo Waltman (EUR)
Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
2011-32 Nees-Jan van Eck (EUR)
Methodological Advances in Bibliometric Mapping of Science
2011-33 Tom van der Weide (UU)
Arguing to Motivate Decisions
2011-34 Paolo Turrini (UU)
Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations
2011-35 Maaike Harbers (UU)
Explaining Agent Behavior in Virtual Training
2011-36 Erik van der Spek (UU)
Experiments in serious game design: a cognitive approach
2011-37 Adriana Burlutiu (RUN)
Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference
2011-38 Nyree Lemmens (UM)
Bee-inspired Distributed Optimization
2011-39 Joost Westra (UU)
Organizing Adaptation using Agents in Serious Games
2011-40 Viktor Clerc (VU)
Architectural Knowledge Management in Global Software Development
2011-41 Luan Ibraimi (UT)
Cryptographically Enforced Distributed Data Access Control
2011-42 Michal Sindlar (UU)
Explaining Behavior through Mental State Attribution
2011-43 Henk van der Schuur (UU)
Process Improvement through Software Operation Knowledge
2011-44 Boris Reuderink (UT)
Robust Brain-Computer Interfaces
2011-45 Herman Stehouwer (UvT)
Statistical Language Models for Alternative Sequence Selection
2011-46 Beibei Hu (TUD)
Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work
2011-47 Azizi Bin Ab Aziz(VU)
Exploring Computational Models for Intelligent Support of Persons with Depression
2011-48 Mark Ter Maat (UT)
Response Selection and Turn-taking for a Sensitive Artificial Listening Agent
2011-49 Andreea Niculescu (UT)
Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

====
2012
====

2012-01 Terry Kakeeto (UvT)
Relationship Marketing for SMEs in Uganda
2012-02 Muhammad Umair(VU)
Adaptivity, emotion, and Rationality in Human and Ambient Agent Models
2012-03 Adam Vanya (VU)

Supporting Architecture Evolution by Mining Software Repositories
2012-04 Jurriaan Souer (UU)
Development of Content Management System-based Web Applications
2012-05 Marijn Plomp (UU)
Maturing Interorganisational Information Systems
2012-06 Wolfgang Reinhardt (OU)
Awareness Support for Knowledge Workers in Research Networks
2012-07 Rianne van Lambalgen (VU)
When the Going Gets Tough: Exploring Agent-based Models of
Human Performance under Demanding Conditions
2012-08 Gerben de Vries (UVA)
Kernel Methods for Vessel Traject
2012-09 Ricardo Neisse (UT)
Trust and Privacy Management Support for Context-Aware Service Platforms
2012-10 David Smits (TUE)
Towards a Generic Distributed Adaptive Hypermedia Environment
2012-11 J.C.B. Rantham Prabhakara (TUE)
Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
2012-12 Kees van der Sluijs (TUE)
Model Driven Design and Data Integration in Semantic Web Information Systems