DISSERTATION

WEESA - Web Engineering for Semantic Web Applications

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaften

unter der Leitung von

Univ.-Prof. Dipl.-Ing. Dr.techn. Harald Gall
Institut für Informatik
Universität Zürich

und

o.Univ.-Prof. Dipl.-Ing. Dr.techn. Mehdi Jazayeri
Institut für Informationssysteme
Technische Universität Wien

eingereicht an der

Technischen Universität Wien
Fakultät für Informatik

von

Univ.-Ass. Dipl.-Ing. Gerald Reif

reif@infosys.tuwien.ac.at

Matrikelnummer: 9130640

Wien, im Mai 2005

Kurzfassung

Die wachsende Popularität des World Wide Webs hat zu einer exponentiellen Steigerung der Zahl der Webseiten geführt. Die große Anzahl der verfügbaren Webseiten macht es Benutzern immer schwerer benötigte Informationen zu finden. Sucht man im Web nach einer spezifischen Information, läuft man Gefahr, die relevanten Daten in der großen Anzahl von irrelevanten Suchergebnissen zu übersehen. Web-Applikationen stellen derzeit Webseiten in HTML Format zur Verfügung, in denen der Inhalt in natürlicher Sprache ausgedrückt ist. Daher ist die Semantik des Inhalts für Computer nicht zugängig. Um es Computern zu ermöglichen dem Benutzer bei Informationsproblemen zu unterstützen, schlägt das Semantik Web eine Erweiterung des existierenden Webs vor, welche die Semantik der Webseiten für Computer verarbeitbar macht. Die Semantik des Inhalts einer Webseite wird dabei mit RDF Meta-Daten beschrieben. Diese Meta-Daten beschreiben den Inhalt der Webseite in einer maschinen-verarbeitbaren Form. Die Existenz von semantisch annotierten Webseiten ist daher die Voraussetzung für das Semantik Web.

Semantische Annotation beschäftigt sich mit diesem Problem und zielt darauf ab, semantische Meta-Daten zu natürlichsprachigen Dokumenten hinzuzufügen, um den Inhalt maschinenverarbeitbar zu machen. Viele Werkzeuge wurden entwickelt, um den Benutzer beim Annotierungsprozess zu unterstützen. Der Annotierungsprozess ist jedoch immer noch ein eigenständiger Prozess der nicht in den Entwicklungsprozess der Web-Applikation integriert ist. Auf der anderen Seite hat die Forschung im Bereich von Web Engineering zu Methoden geführt, um Web-Applikationen zu entwickeln und zu warten. Die vorgeschlagen Methoden unterstützen jedoch nicht das hinzufügen von semantischen Meta-Daten.

Diese Dissertation stellt eine Technik vor, um existierende XML-basierte Web Entwicklungsmethoden zu erweitern, um semantisch annotierte Webseiten zu erzeugen. Die Innovation des vorgestellten Ansatzes, genannt WEESA, ist die Verknüpfung von Elementen aus einem XML Schema mit den Konzepten die in einer Ontologie definiert sind. Diese Verknüpfung wird dann verwendet um aus XML-Dokumenten RDF Meta-Daten zu generieren. Weiters stellen wir die Integration des WEESA Meta-Daten Generators in die Apache Cocoon Webentwicklungsumgebung vor, das die Entwicklung von semantisch annotierten Webapplikationen erleichtert.

Betrachtet man nur die Meta-Daten einer einzelnen Webseite, hat man nur eine eingeschränkte Sicht auf die Meta-Daten die die Web-Applikation zur Verfügung stellt. Für Anfragen und logische Schlüsse ist es besser man hat die vollständigen Meta-Daten der ganzen Web-Applikation zur Verfügung. In dieser Dissertation stellen wir die WEESA Wissensbasis vor. Diese Wissensbasis wir serverseitig durch das Akkumulieren der Meta-Daten der individuellen Webseiten erzeugt und steht dann für Anfragen und zum Herunterladen zur Verfügung.

Die Wiener Festwochen Fallstudie zeigt den praktischen Einsatz von WEESA in einer Apache Cocoon Web-Applikation. Wir diskutieren die Erfahrungen aus der Entwicklung der Fallstudie und präsentieren Richtlinien zum Entwickeln von semantisch annotierten Web-Applikationen mit WEESA.

Abstract

In the last decade the increasing popularity of the World Wide Web has lead to an exponential growth in the number of pages available on the Web. This huge number of Web pages makes it increasingly difficult for users to find required information. In searching the Web for specific information, one gets lost in the vast number of irrelevant search results and may miss relevant material. Current Web applications provide Web pages in HTML format representing the content in natural language only and the semantics of the content is therefore not accessible by machines. To enable machines to support the user in solving information problems, the Semantic Web proposes an extension to the existing Web that makes the semantics of the Web pages machine-processable. The semantics of the information of a Web page is formalized using RDF meta-data describing the meaning of the content. The existence of semantically annotated Web pages is therefore crucial in bringing the Semantic Web into existence.

Semantic annotation addresses this problem and aims to turn human-understandable content into a machine-processable form by adding semantic markup. Many tools have been developed that support the user during the annotation process. The annotation process, however, is a separate task and is not integrated in the Web engineering process. Web engineering proposes methodologies to design, implement and maintain Web applications but these methodologies lack the generation of meta-data.

In this thesis we introduce a technique to extend existing XML-based Web engineering methodologies to develop semantically annotated Web pages. The novelty of this approach is the definition of a mapping from XML Schema to ontologies, called WEESA, that can be used to automatically generate RDF meta-data from XML content documents. We further demonstrate the integration of the WEESA meta-data generator into the Apache Cocoon Web development framework to easily extend XML-based Web applications to semantically annotated Web application.

Looking at the meta-data of a single Web page gives only a limited view of the of the information available in a Web application. For querying and reasoning purposes it is better to have the full meta-data model of the whole Web application as a knowledge base at hand. In this thesis we introduce the WEESA knowledge base, which is generated at server side by accumulating the meta-data from individual Web pages. The WEESA knowledge base is then offered for download and querying by software agents.

Finally, the Vienna International Festival industry case study illustrates the use of WEESA within an Apache Cocoon Web application in real life. We discuss the lessons learned while implementing the case study and give guidelines for developing Semantic Web applications using WEESA.

Acknowledgements

I would like to express my deep gratitude to all of the people who have supported me during my research.

In particular, I offer my sincerest gratitude to my supervisors, Prof. Dr. Harald C. Gall and Prof. Dr. Mehdi Jazayeri, who introduced me to research, gave me the opportunity to pursue my research ideas, and kept me on the right track with their advice.

It is a pleasure to acknowledge the great collaboration with the MOTION team at the Distributed Systems Group of the TU Vienna: Harald Gall, Pascal Fenkam, and Engin Kirda. The countless nights on the train to Milan (which would have been incomplete without our dearest Giovanni), completing the "mission impossible", writing a user interface in six days (on the seventh day we had to rest), and making a Finnish man smile – were unforgettable experiences. Many thanks are due to Clemens Kerer for the invaluable discussions on Web engineering. Your work on xGuide inspired me to start the WEESA project.

The Distributed Systems Group deserves my gratitude for sharing an enjoyable working environment. This thesis would not have been possible without your technical and administrative support: organizing trips, helping me with the right form for almost every purpose, filtering the spam, etc.

Many thanks again to Harald Gall who invited me to spend a research summer with his group at the University of Zurich. Working in this fertile environment resulted in the prototype implementation presented in this thesis and a WWW publication. Besides the thesis, Zurich was also a great palace to live and for having BBQs. – Merci vielmals!

Thanks to Suzanne Whitby for proofreading the thesis, making it more readable for native speakers and non-native speakers alike.

Finally, I would like to thank my parents who brought me up, taught me to ask questions, and helped me to find answers, which is the basis for all science. They supported me in whatever I did and however they could throughout my (long) student years.

Gerald Reif
Vienna, Austria, May 2005

Meinen Eltern

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

> The killer app will not be a shrink-wrapped program that sells millions.
> The killer app will be a Web site that touches millions of people and
> helps them to do what they want to do.
>
> Lou Gerstner

1.1 MOTIVATION

Over the last decade the World Wide Web (WWW) has emerged to an important part of our everyday life. Companies, organizations, and people use the WWW as a powerful tool to share information: Companies offer Web pages to advertise and sell their products. Educational institutions present teaching material and online training services on the Web. Public authorities provide eGovernment services to make administrative tasks more efficient and citizen-friendly. User groups maintain Web portals to exchange information within their community.

The popularity of the WWW lead to an exponential growth in the number of Web pages available in the global information space. This success, however, leads to several problems: The huge number of available Web pages makes it increasingly difficult for users to find and access required information. In searching the Web for a specific information, one gets lost in the huge amount of irrelevant search results and may miss the relevant material.

Electronic commerce is currently hampered by the way information is presented. Since the semantics of the Web pages in not directly accessible for machines, for example, shopping agents have to use wrappers and heuristics to extract relevant product information from weakly structured HTML documents to compile a market overview.

Currently, data can be shared between applications via copy and paste only in (rich) text format. The semantics of the data gets lost. Users, however, could benefit if data from Web

applications could directly be further processed by desktop applications without loosing its semantics.

The emerging awareness to these problems resulted in the insight that information can only be shared and reused across application, enterprise, and community boundaries if the semantics of the data is accessible to machines. By this means search results can be improved, the semantics of data on Web pages can directly be accessed and further processed in software agents and desktop applications.

Current Web applications provide Web pages in HTML format only that can be presented in a Web browser to human users. The content of the pages is expressed via natural language and therefore the semantics is not accessible for machines. Hence, an addition to the current Web is needed to make the semantics of the Web pages machine-processable.

Tim Berners-Lee defines the *Semantic Web* as an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation [9]. The meaning of the information on a Web page is formalized using semantic meta-data that is based on concepts defined in ontologies. Therefore, the existence of semantically annotated Web pages is crucial to bring the Semantic Web to life.

Looking at the meta-data of a single Web page, however, gives only a very limited view of the information available in a Web application. For querying and reasoning purpose it would be better to have the whole meta-data model of the Web application at hand. Therefore, to bring the Semantic Web to its full potential, Web applications in the Semantic Web have to offer semantic meta-data describing the content of the HTML Web pages, and the accumulated meta-data model of the application in a knowledge base for querying and download.

1.2 Problem Domain: Engineering Semantic Web Applications

The success of the Semantic Web crucially depends on the existence of semantically annotated Web pages. But it is still costly to develop and maintain Web applications that offer both: human-understandable information that can be displayed by a Web browser and machine-processable meta-data that can be processed by computers. Therefore, methodologies are needed to engineer semantically annotated Web applications, so called *Semantic Web applications*.

Not much work has been done in developing methodologies to engineer Semantic Web applications. Related work, however, can be found in the following areas:

Semantic annotation aims to turn human-understandable content into a machine-processable form by adding semantic markup [36]. Many tools have been developed that support the user during the annotation process. But still, the annotation process is a separate task and is not integrated in the Web engineering process.

Web Engineering focuses on the systematic and cost efficient design, development, maintenance, and evolution of Web applications [31]. The outcome of the Web Engineering process are Web applications that provide Web pages that can be displayed in a Web browser but these applications lack semantic markup.

To benefit from the experiences in the related research areas a methodology to engineer Semantic Web applications should be based on the research in the Web engineering community and aim to integrate the research results from the semantic annotation community. Common Web engineering concepts and artifacts should be reused to enable the integration into existing Web engineering methodologies. For example, most Web engineering methodologies are based on the concept of separation-of-concerns to define strict roles in the development process and to enable parallel development. In addition, XML documents and XSLT stylesheets are frequently used to separate the content of Web pages and the graphical appearance of the Web application. Therefore, a methodology to engineer Semantic Web applications should not violate the concept of separation-of-concerns and use the XML content documents as basis for the semantic annotation process. In addition, to support developers when developing Semantic Web applications the Semantic Web engineering methodology has to be integrated into a Web application development framework.

A Semantic Web engineering methodology has also to account for the generation and maintenance of the knowledge base of the Semantic Web application. The semantic meta-data from single Web pages have to be accumulated and integrated in the knowledge base. In addition, the knowledge base has to be kept consistent with the data in the Web application during the life time of the Semantic Web application.

1.3 CONTRIBUTION

In this thesis we introduce a methodology to design, implement and maintain Semantic Web applications. Semantic Web applications are Web applications that not only offer content documents in HTML format but also semantic meta-data describing the content of the Web pages in a machine-processable way. The proposed methodology further provides the possibility to accumulate the meta-data of the individual Web pages to build the meta-data model of the whole Web application as a knowledge base. The knowledge base is then offered for querying and download by software agents.

The approach to engineer Semantic Web applications we present in this thesis is called WEESA (WEb Engineering for Semantic web Applications). The contribution of WEESA is the conceptual definition and prototype implementation of a mapping from XML Schema to ontologies that enables the efficient design of Semantic Web applications. Once the mapping is defined in the design phase the mapping is used at runtime to automatically generate RDF meta-data descriptions from XML content documents. WEESA follows the concept of separation-of-concerns and can be used to extend XML-based Web engineering methodologies to semantically

annotate Web applications. To our knowledge, WEESA is the first implemented approach that integrates semantic annotation in the Web engineering process.

In this thesis we further show the integration of the WEESA meta-data generation into Apache Cocoon Web applications. Apache Cocoon [16] is a component based Web development framework that uses XML/XSLT for separation-of-concerns. We integrated WEESA into Cocoon transformer components that can then be used to realize Semantic Web applications from scratch or to extend existing Cocoon Web applications.

The WEESA enabled Cocoon transformers also support the accumulation of meta-data from single Web pages into the knowledge base of the Semantic Web application. The knowledge base management component is responsible to keep the knowledge base up-to-date, to remove outdated entries, to offer the knowledge base for download, and to provide access to the query service of the knowledge base.

Finally, the Vienna International Festival industry case study illustrates the use of WEESA within an Apache Cocoon Web application in real life. We discuss the lessons learned while implementing the case study and give guidelines for developing Semantic Web applications using WEESA.

1.4 Structure of the Thesis

The remainder of this thesis is structured as follows.

Chapter 2 introduces the vision of the Semantic Web and discusses the data-model, standards, and technologies used to bring this vision into being. These building blocks form the foundation of the idea presented in this thesis. The chapter further introduces the terms used throughout the thesis.

Chapter 3 presents WEESA, our approach to engineer semantically annotated Web applications, that can be used to extend existing XML-based Web engineering methodologies. We introduce the WEESA mapping from XML Schema to ontologies that can be used to automatically generate semantic meta-data from XML content documents, present the prototype implementation, and discuss the asymptotic complexity of the proposed algorithm.

Chapter 4 shows the integration of the WEESA meta-data generator into the Cocoon Web application development framework. We present two WEESA enabled transformer components and show how they are used to develop Semantic Web applications. We further discuss different possibilities to associate HTML Web pages and RDF meta-data.

Chapter 5 introduces the the WEESA knowledge base that is built at server side by accumulating the meta-data descriptions from individual Web pages. The knowledge base is offered for download and querying by software agents. We show the integration of the knowledge base into the existing Cocoon infrastructure and the use of RDF reification to keep the knowledge base up-to-date.

Chapter 6 shows the use of WEESA in the Vienna International Festival (VIF) industry case study. We discuss the WEESA mapping definitions needed, the used Cocoon component

configuration, and give example scenarios how the WEESA knowledge base can be used by software agents.

Chapter 7 discusses the experiences gained and the lessons learned while implementing the case study, we list the qualifications a developer should have to semantically annotate a Web application, and give guidelines for developing a Semantic Web application using WEESA.

Chapter 8 presents related work. Not much work has been done to integrate the semantic annotation process in a Web engineering methodology. This Chapter, therefore, presents the related research areas Web engineering and semantic annotation.

Chapter 9 concludes the thesis and gives an outlook on future work.

CHAPTER 2

SEMANTIC WEB REVIEW

> We've all heard that a million monkeys
> banging on a million typewriters
> will eventually reproduce the entire works of Shakespeare.
> Now, thanks to the Internet, we know this is not true.
>
> Robert Wilensky

The first time the term "Semantic Web" came up was in 1998 when Tim Berners-Lee published the *Roadmap to the Semantic Web* [7] on the homepage of the World Wide Web Consortium (W3C) [97]. From then it took till 2001 to the well known Scientific America publication "The Semantic Web" [9] and the *Semantic Web Kick-Off Seminar in Finland* [42,91]. Since then a very active research community established having their own conferences and workshops. The community came up with various standards and tools to bring the vision of the Semantic Web into being.

This chapter gives an introduction to the vision of the Semantic Web. We give an overview of the Semantic Web related W3C recommendations this thesis is based on. This chapter further defines the meaning of the terms used throughout the thesis.

2.1 THE SEMANTIC WEB VISION

The increasing popularity of the World Wide Web (WWW) has changed the way we think about our computers. Originally computers were used for computing numerical calculations [4]. With the growing number of personal computers the dominant application domain shifted to text processing, spread sheets, and gaming. At the present the increasing popularity of mobile computing

devices such as notebooks, PDAs, and mobile phones and the possibility to connect to the Internet even from public places through wireless LAN, GPRS, and UMTS further shifted our view of computers towards an entry point to the global information space of the WWW.

Since the WWW was "born" at the CERN laboratories in 1989 the Web evolved to a global information space that is made up of billions of Web pages. This large number of We pages ensures that information on almost every topic is provided on the Web but makes it difficult to find wanted information. Search engines such as Google[1] and Yahoo[2] aims to assist users when searching the Web for information. Since the content of the Web pages is presented in human language search engines have no access to the semantics of the content. This reduces the possibilities of search engines mainly to keyword search.

For example, a user is interested in the play-time of a song on the "Alanis Unplugged" CD of the artist "Alanis Morissette". With current search engines a user cannot directly query the Web for the wanted information but has to reformulate his information problem. The user has to think of keywords that are likely to be found on a Web page that provides the play-times of the tracks on the CD. As search result the user gets a list of Web pages that contain the keywords and the user has to browse these pages to find the wanted information.

Even worse is the situation when the needed information cannot be found on a single Web page. In this case the user has to formulate several queries, browse through the various search results, and combine the information found. Computers can hardly support the user with this problem since machines do not have access to the semantics of the Web pages.

To overcome this limitations the Semantic Web aims to make the Web's content machine-processable[3]. Tim Berners-Lee defines the Semantic Web as follows:

> *"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."* – Tim Berners-Lee, James Hendler, Ora Lassila; May 2001 [9]

Following this definition, the Semantic Web is not a new Web but an extension of the existing Web. This extension consists of meta-data describing the semantics of the Web pages in a machine-processable way. Before the Web pages can be described with semantic meta-data, an *ontology* has to be defined for the domain of discourse. An ontology formally describes the concepts found in the domain, the relationships between these concepts, and the properties used to describe the concepts. For example, in the domain of an online record shop we define concepts such as "composer", "album", and "tack"; relationships such as "composed by", and "has track"; and properties such as "has play-time", and "has title". The meta-data description of a Web page

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

[2] `http://www.yahoo.com`

[3] In the literature the term machine-understandable is used quite often instead of machine-processable. We follow the view of Antoniou and van Harmelen in [4] and believe that machine-understandable is the wrong word because it gives the wrong impression. It is not necessary for software agents to *understand* the information; it is sufficient for them to *process* the information efficiently, which sometimes causes people to think the machine really understands.

uses the concepts, properties, and relationships defined in the ontology to give the information a well-defined meaning.

Since the Semantic Web provides machine-processable information data can be shared and reused across application, enterprise, and community boundaries [90]. For example, in the Semantic Web the search engine problem, discussed above, can be solved by defining a database like query to a Semantic Web search engine for the play-time of the wanted track. The search result is then directly the requested information instead of a list of Web pages that might contain the information.

2.2 STANDARDS, RECOMMENDATIONS, AND TOOLS

In oder to bring the vision of the Semantic Web into being the research community came with standards, W3C recommendations, development frameworks, APIs, and databases. These languages and tools form the basic building blocks the approach presented in this thesis is based on. In this section we give a brief introduction into the background technologies used.

2.2.1 RESOURCE DESCRIPTION FRAMEWORK (RDF)

The Resource Description Framework (RDF) [57, 61] is the data-model for representing metadata in the Semantic Web. Before we take a look at RDF we discuss the deficits of XML. XML is a universal meta-language for defining markup [4]. Many tools such as parsers have been developed that enable the information exchange between applications. However, there is no inherent meaning associated with the nesting of the XML elements. It is up to the application to interpret the nesting. For example, if we want to express the following fact:

The CD with the item number "1234" has the title "Alanis Unplugged".

There are various ways to express this fact in XML. In Figure 2.1 we give two examples.

```
<cd item="1234">
  <title>Alanis Unplugged</title>
</cd>

<cd>
  <item>1234</item>
  <title>Alanis Unplugged</title>
<cd>
```

Figure 2.1: Different XML documents to express the same fact.

As we can see, no standard way exists to assign meaning to the nesting of the XML elements. In RDF, however, we are able to express the meaning of fact above unambiguously. The RDF

data-model is based on *subject – predicate – object* triples, so called RDF statements, to formalize meta-data. RDF is domain independent in that no assumptions about a particular domain of discourse are made. It is up to the users to define their own ontologies for the user's domain in an ontology definition language such as RDF Schema (RDFS) [11]. The definition of ontologies is discussed in the following section. Unfortunately, the name RDF Schema is not a good choice, since it suggests that RDF Schema has a similar relation to RDF as XML Schema to XML. This, however, is not the case. XML Schema constraints the *structure* of the XML document, whereas RDF Schema defines the *vocabulary* used in the RDF data-model. For example, the ontology to express the fact above has to define the concept of a "CD" and the relationship "has title" in its vocabulary.

Before we are able to express the fact above as RDF statement we have to introduce the concept of a resource. A resource can be seen as a "thing" we want to make a statement about. A resource can be everything; a book, a person, a Web page, a CD, a track on a CD, and so on. Every resource is identified by a Uniform Resource Identifier (URI) [8]. In the case of a Web page, the URI can be the Unified Resource Locator (URL) of the page. The URI does not necessarily enable the access via the Web to the resource; it simply has to unambiguously identify the resource.

Now we can formulate the fact from above as RDF statement. The subject, predicate, and object are defined as follows:

Subject: The Subject is the thing (the resource) we want to make a statement about. In our example we want to make a statement about a CD. To be able to make a statement about the CD we use the URI `"http://mytunes.com/album_id1234"` as resource identifier for the CD.

Predicate: The predicate defines the kind of information we want to express about the subject. In our example we want to make a statement about the title of the CD. To define the kind of information we want to state about a the subject we use the URI `"http://mytunes.com/ontology#hasTitle"` that references the property defined in the ontology. How ontologies are defined we will see in the following section. The predicate is also called the *property* that describes the subject resource.

Object: The object defines the value of the predicate. In our example we want to state that the title of the CD is `"Alanis Unplugged"`. The object can be a literal, like in our example, or another resource represented by the object's URI.

In the RDF data-model the statements are represented as nodes and arcs in a graph. The RDF graph model is defined in [57]. In this notation, a statement is represented by:

- a node for the subject,
- a node for the object, and
- an arc for the predicate, directed from the subject node to the object node.

Figure 2.2: RDF graph of a single statement.

Figure 2.2 shows the RDF graph for the statement discussed above. Resources in the graph are depicted as ellipses and literals are depicted as rectangles. To write down RDF statements there exists several concrete syntaxes. In the *triples notation*, each statement in the graph is written as a simple triple of subject, predicate, and object, in that order followed by a full stop. For example, the statement shown in the graph in Figure 2.2 would be written in the triple notation as:

```
<http://mytunes.com/album_id1234> <http://mytunes.com/ontology#hasTitle> "Alanis Unplugged" .
```

RDF also provides an XML syntax for writing down and exchanging RDF graphs, called *RDF/XML*. Unlike the triples notation, which are intended as a shorthand notation, RDF/XML is the normative syntax for writing RDF and is defined in [26]. The RDF/XML notation for the graph shown in Figure 2.2 is given in Figure 2.3.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:mytunes="http://mytunes.com/ontology#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  <rdf:Description rdf:about="http://mytunes.com/album_id1234">
    <mytunes:hasTitle>Alanis Unplugged</mytunes:hasTitle>
  </rdf:Description>
</rdf:RDF>
```

Figure 2.3: RDF/XML syntax of the statement defined in Figure 2.2.

2.2.2 Ontology Languages

RDF provides a way to express simple statements about resources, using subject – predicate – object triples. However, to use RDF we also need the possibility to the define the vocabulary that is used in the RDF statements. This controlled vocabulary is also called *ontology*. For a given domain the ontology defines the concepts found in the domain, the relationships between these concepts, and the properties used to describe the concepts.

For example, if we take the domain of an online record shop, we can identify concepts (also called classes) such as “CD”, “tape”, “record”, “storage media”, “artist”, and “track”. As relationship between these concepts we can think of “has track”, “has artist”, or the subclass relationship between “storage media” and the classes “CD”, “tape”, and “record”. As properties to describe the classes we use “has artist name”, “has album title”, “has track name”, or “has play-time”.

Ontologies are defined in an ontology definition language. In the following sections we introduce the two ontology languages the W3C defined for the Semantic Web: RDF Schema and the Web Ontology Language (OWL).

2.2.2.1 RDF SCHEMA

RDF Schema (RDFS) [11] is a simple ontology definition language that allows users to define the vocabulary needed to describe the resources in the domain with meta-data. To define the ontology RDFS uses the RDF triples format. Therefore, an ontology in RDFS is modeled as RDF graph. In RDFS users can define classes, properties, and relationships to model the concepts in the domain.

In the following we introduce some of the basic modeling primitives to define an ontology in RDFS. Terms that are defined in the RDFS language specification have the XML namespace prefix `"rdfs:"`; terms defined in the RDF specification have the prefix `"rdf:"`. The modeling primitives consist of classes and properties. We start with the list of classes that are used as resources in the ontology definition. The complete list of classes can be found in [11].

- `rdfs:Class` This is the class of resources that are RDF classes.
- `rdfs:Literal` This is the class of literal values such as strings and integers.
- `rdfs:Property` This is the class of RDF properties.

The property primitives are used to define relationships in the ontology. The complete list of properties can be found in [11].

- `rdf:type` Relates a resource to its class. In other words, the `rdf:type` property is used to declare a resource to be an instance of the given class.
- `rdfs:subClassOf` Relates a class to one of its superclasses. All instances of a class are instances of its superclass. A class can be a subclass of more than one class.
- `rdfs:domain` Specifies the domain of the property *P*, that is, the class of resources that may appear as subject in a triple with the predicate *P*. If the domain is not specified any resource can be the subject.
- `rdfs:range` Specifies the range of a property P, that is, the class of those resources that may appear as object in a triple with the predicate *P*. If the domain is not specified any resource can be the object.

So far we have introduced some of the modeling primitives of RDFS. In Figure 2.4 we show the RDF graph of a simple ontology for our online record shop example. For simplicity reasons we omit the `"mytunes:"` namespace prefix in the discussion of the example. In the ontology we defined three classes: `StorageMedia`, `CD`, and `Track`. These resources are classes, since

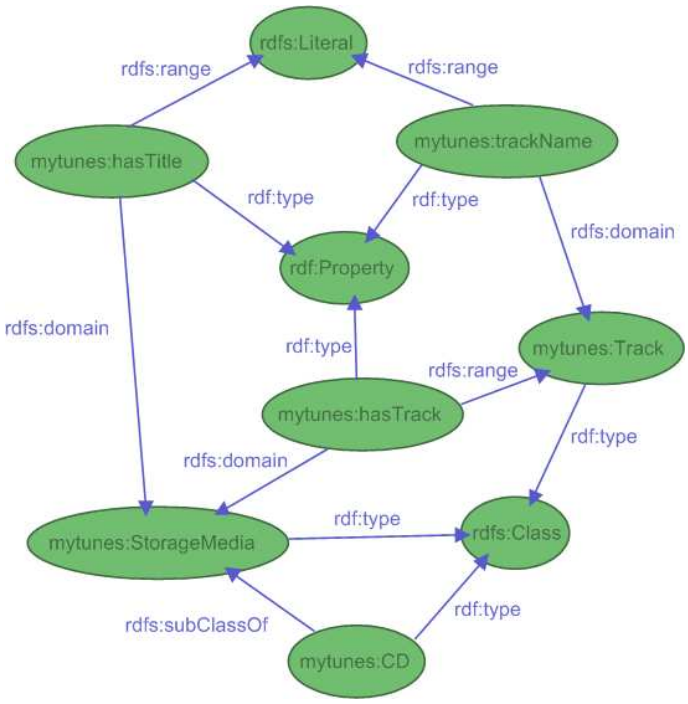

Figure 2.4: RDF graph of the example ontology for an online record shop.

they are related to the `rdfs:Class` resource with the `rdf:type` property. The class `CD` is also declared to be a `rdfs:subClassOf` the class `StorageMedia`.

The ontology further defines three properties: `hasTitle`, `trackName`, and `hasTrack`. The `hasTrack` property can only be used to describe an instance of the `StorageMedia` class (and all its subclasses), since the `rdfs:domain` is specified. The `hasTrack` property can further only take objects as value, that are an instance of the `Track` class, since the `rdfs:range` is specified. The properties `hasTitle` and `trackName` take a literal as object, since the range is defined to be a `rdfs:Literal`.

An RDF graph that uses this ontology to describe a CD is shown in Figure 2.5. The `hasTrack` property points to an *anonymous resource*, also called *blank node*, that is an instance of the class `Track`. Anonymous resources are used for resources that never need to be referred to directly from outside the RDF description. The anonymous resource, however, is needed to represent the instance of the `Track` class, that are described by the `trackTitle` property.

2.2.2.2 Web Ontology Language OWL

The expressiveness of RDF Schema introduced in the previous section is very limited. RDF Schema can be used to define subclass hierarchies, properties, and domain and range restrictions of those properties [4]. However, the Web Ontology working Group of the W3C [69] identified

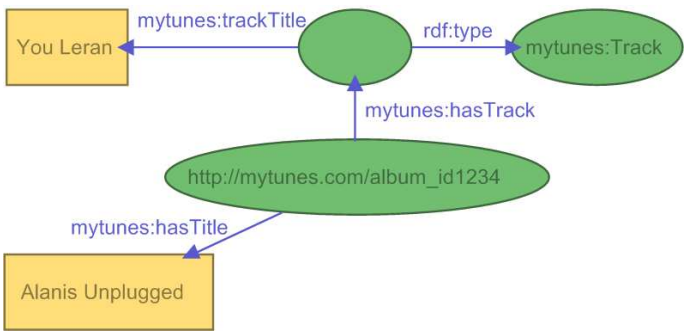

Figure 2.5: RDF graph of ontology that uses the ontology from Figure 2.4.

a number of characteristic use cases [27] that cannot be covered by RDF Schema.

A number of researchers in Europe and the United States identified the need for a more powerful ontology modeling language. This lead to a joint initiative to define a more expressive language, called DAML+OIL[4] [18]. DAML+OIL was in turn the staring point for the W3C working group to define the Web Ontology Language (OWL) [72]. OWL is now a W3C recommendation.

OWL provides a number of additional modeling primitives that increase the expressiveness compared to RDFS. In the following we discuss some of the shortcomings of RDFS that can be expressed in OWL [4]:

- Locale scope of properties. For example, we want to define a property `eats` that is used in the domain of the `Sheep` and `Tiger` classes. The `rdfs:range` primitive cannot be used define that a `Sheep` only eats plants, while the `Tiger` only eats meat.

- Disjoint classes. RDFS does not provide primitives to declare two classes to be disjoint. For example, the classes `Male` and `Female` are disjoint.

- Boolean combination of classes. In some cases we have to define new classes by building the *union*, *intersection*, or *complement* of other classes. For example, we want to define the class `Person` to be the disjoint union of the classes `Male` and `Female`.

- Cardinality restrictions. RDFS does not provide any means to restrict the number of distinct values a property may or must take. For example, we want to specify that a person has exactly two parents, or that a course is taught by at least one lecturer.

- Special characteristics of properties. In RDFS we cannot define that a property is *transient* (e.g. “greater than”), *unique* (e.g. “is mother of”), or the *inverse* of another property (e.g. “eats” and “is eaten by”)

[4]The name is a join of the names of the U.S. language DAML-ONT [19] and the European proposal OIL [67].

When the W3C's Web-Ontology Working Group specified OWL it followed two goals: (1) To define an ontology language with maximum expressiveness, (2) while providing efficient reasoning support. Since these goals hardly goes together, the working group defined OWL as three different sublanguages, each geared toward fulfilling different aspects of this full set of requirements [62].

OWL Light: OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies. OWL Lite also has a lower formal complexity than OWL DL [22].

OWL DL: OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (For example, while a class may be a subclass of many classes, a class cannot be an instance of another class.). OWL DL got its name due to the correspondence with *description logics*, a field of research that has studied the logics that form the formal foundation of OWL.

OWL Full: OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full we can impose a cardinality constraint on the class of all classes, limiting the number of classes that can be described in an ontology. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

2.2.3 RDF REIFICATION

Since the approach presented in this thesis uses RDF reification, we introduce this advanced RDF modeling primitive in this section. As explained in Section 2.2.1 RDF statements are used to describe a resource. In some cases, however, we want to make statements about an RDF statement. For example, we want to record who provided the particular information.

RDF provides a build in vocabulary intended for describing RDF statements. A description of a statement using this vocabulary is called *reification* of a statement. The RDF reification vocabulary consists of the type `rdf:Statement` and the properties `rdf:subject`, `rdf:predicate`, and `rdf:object`.

For example, we want to express that the creator of the statement in Figure 2.2 is a person with the staff identifier `"9876"`. To identify this person we use the URI `http://example.com/staffid/9876`. The RDF graph of the reified statement is shown in Figure 2.6. The anonymous resource is used to represent the reified statement. The `rdf:type` property defines that the anonymous resource represents the `rdf:Statement`. The properties are used to describe the subject, predicate, and object of the statement. The `http:`

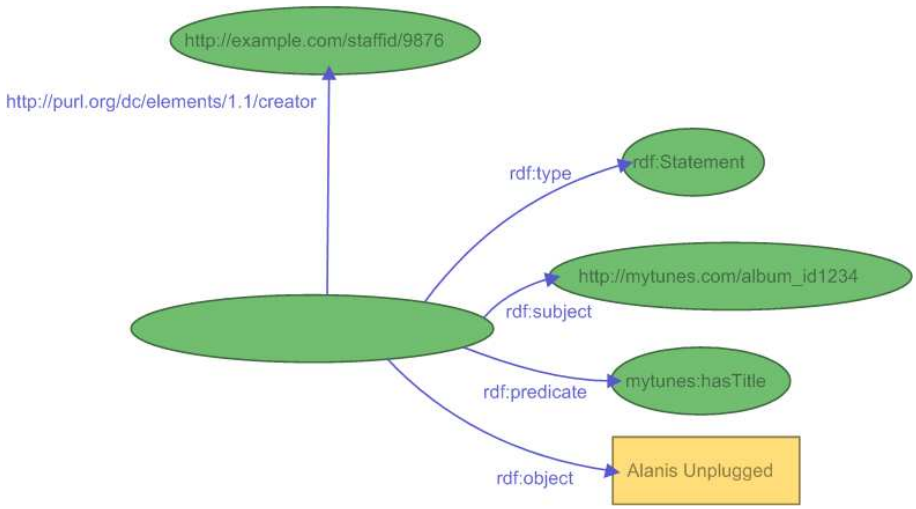

Figure 2.6: RDF graph of a reified RDF statement.

`//purl.com/dc/elements/1.1/creator` property is defined in the Dublin Core ontology (DC) [24]. DC is a meta-data standard for describing digital objects and defines properties such as creator, title, and publisher. The `creator` property is used to record that the person that is identified with the URI `http://example.com/staffid/9876` has made this statement. Figure 2.7 shows the reified statement in RDF/XML syntax.

When using RDF reification the original statement is typically also stored in the RDF graph. This means, after reifying a statement we result with four more RDF statements in the graph plus the statements used for the description (in our example, the `creator` statement). Therefore, using RDF reification increases the number of statements in the graph significantly.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:mytunes="http://mytunes.com/ontology#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Statement>
    <dc:creator rdf:resource="http://example.com/staffid/9876"/>
    <rdf:subject rdf:resource="http://mytunes.com/album_id1234"/>
    <rdf:predicate rdf:resource="http://mytunes.com/ontology#hasTitle"/>
    <rdf:object>Alanis Unplugged</rdf:object>
  </rdf:Statement>
</rdf:RDF>
```

Figure 2.7: Reified RDF statement in RDX/XML Syntax.

2.2.4 JENA – SEMANTIC WEB FRAMEWORK FOR JAVA

In this chapter we introduced so far W3C recommendations that were developed for the Semantic Web. To bring the Semantic Web into being, however, program libraries and tools are needed that are based on these recommendations and enable developers to efficiently develop applications for the Semantic Web. In this section we introduce Jena, the framework used to implement the approach presented in this thesis.

Jena [46] is a Java framework for building applications for the Semantic Web. Jena is open source and has grown out of work with the HP Labs Semantic Web Programme[5]. It provides a programmatic environment for RDF, RDFS, and OWL. The framework includes an RDF parser that provides methods for reading and writing RDF in various formats such as triple notation and RDF/XML.

The Jena Java API provides classes respectively interfaces for the language primitives of RDF and OWL. For example, the API provides the interfaces for the `Resource`, `Property`, and `Literal` primitives. In Jena, a graph is called model and is represented by the `Model` interface. Figure 2.8 shows the definition of an RDF statement in a Java program using the Jena framework.

```
// some definitions
String albumURI = "http://mytunes.com/album_id1234";
String title    = "Alanis Unplugged";

// create an empty Model
Model model = ModelFactory.createDefaultModel();

// create the resource
Resource alanisUnplugged = model.createResource(albumURI);

// create the property
Property hasTrack = model.createProperty("http://mytunes.com/ontology#hasTitle");

// add the property to make an RDF statement
alanisUnplugged.addProperty(hasTrack, title);
```

Figure 2.8: Java programm to define an RDF statement using the Jena framework.

2.3 TERMINOLOGY

Understanding the meaning of terms used in a given context is crucial for effective and unambiguous communication. Unfortunately some confusion exists in the way terms such as Web application, Web Service, or Web engineering are used. In this section we present our understanding of the terms in these context of this thesis.

Web page A Web page is a document in HTML format that is typically retrieved by Web browsers to be displayed on screen.

[5] `http://www.hpl.hp.com/semweb/`

Web application A Web application is a collection of static and/or dynamically generated Web pages that form a unit in terms of content they provide, share a common look-and-feel, and are available through the same base URL. A Web application can be seen as a software application leveraging the Web as user interface.

Web Service The term Web Service is one of the most over-used terms in the Web area. In the early days of the WWW the term was used as a synonym for Web application. More recently the term was redefined in the context of machine-to-machine communication. These services exchange machine readable information utilizing Web technologies such as the Simple Object Access Protocol (SOAP) that communicates via XML messages that are transmitted over HTTP. In the remainder of this work we restrict the use of the term Web service to the later meaning.

Web engineering Web Engineering includes all activities involved in the design, implementation, deployment, maintenance and evolution of a Web application.

Resource A resource is a "thing" that can be referenced by an identifier. In the Semantic Web a resource is identified by an Uniform Resource Identifier (URI). A resource can either be accessible via the Web such as a Web page, or only be referenced such as persons, books, wines, or places. In the Semantic Web resources are used to reference the thing that should be described with meta-data.

RDF triple An RDF triple is a subject – predicate – object triple that is used to describe a resource in the Semantic Web. The subject represents the resource to be described, the predicate the property to be expressed, and the object the value of the property. RDF triples are used to formalize semantic meta-data in the Semantic Web.

RDF statement The term RDF statement is used synonymously with RDF triple.

Semantic Web page A Semantic Web page consists of the Web page presenting the content in HTML format, and in addition, a machine-processable meta-data description of the content of the Web page. The machine-processable meta-data enables machines to have access to the semantics of the content. In the Semantic Web the meta-data is represented using the Resource Description Framework (RDF).

Semantic Web application A Semantic Web application is a Web application that provides Semantic Web pages instead of Web pages. Therefore, software agent have access to the semantics of the content of the Web pages. In addition, a Semantic Web application offers the meta-data of the whole Web application in a knowledge base for download and querying.

CHAPTER 3

WEESA – WEB ENGINEERING FOR SEMANTIC WEB APPLICATIONS

> The important thing in science is not so much to obtain new facts
> as to discover new ways of thinking about them.
>
> Sir William Bragg

In the previous chapter we introduced the vision of the Semantic Web. To bring this vision into being, however, Web applications have to provide Web pages that are semantically annotated. Unfortunately, adding semantic meta-data to Web pages is still a time consuming task. Semantic annotation addresses this problem and aims to make the content of human-understandable documents machine-processable by adding semantic markup [36]. Research projects on semantic annotation lead to the development of various tools that support the user during the annotation process. But still, the annotation process is an additional task and not yet integrated in the engineering process of the Web application.

Web engineering aims to provide methodologies for the systematic and cost efficient design and development of Web applications [31]. Current Web engineering methodologies can be used to realize Web applications that provide Web pages in HTML format but these applications do not provide semantic meta-data. Therefore, new methodologies are needed to engineer Semantic Web applications.

In this chapter we introduce WEESA (WEb Engineering for Semantic web Applications), a technique that can be used to extend existing XML-based Web engineering methodologies to engineer semantically annotated Web applications. WEESA defines a mapping from XML Schema to ontologies that can be used to automatically generate RDF meta-data from XML documents. In the following chapter we show the integration of WEESA into Apache Cocoon transformer components and the use of this transformer to develop Semantic Web applications.

3.1 Semantic Annotation

The aim of semantic annotation is to transform documents into machine-processable artifacts by augmenting them with meta-data that describes their meaning. In the Semantic Web, this meta-data description is done using the Resource Description Framework (RDF) [57] that references the concepts defined in one or more ontologies. Ontologies formally define concepts used in a domain and the relationship between these concepts. An ontology is defined in an ontology definition language such as RDFS [11], DAML+OIL [18], or OWL [72].

When adding semantic meta-data to documents, one faces several problems [74]:

- Annotating documents with Semantic markup is a time consuming task and has to be performed in addition to the authoring process.
- The authors that annotate the documents are typically not the people who profit from the existence of meta-data. This reduces the author's motivation to annotate Web pages.
- The granularity of the information found in the document does not meet the needs of granularity in the ontology. Several information items that can be found in the document might be needed to compute the value that fits a property in the ontology.
- Looking at Web pages that provide RDF meta-data we recognize that important parts of the content are stored twice. First, in HTML format that is displayed to the user via the Web browser and second in the RDF description. This redundancy leads to inconsistency problems when maintaining the content of the Web page. Changes must always be made consistently for both types of information. Therefore, support is needed for the creation and maintenance of Semantic Web pages.
- Many Web pages are not static documents but are generated dynamically e.g. using a database. Annotating dynamic documents leads to performing the same task over and over for a specific pattern of documents.

Several annotation tools have been proposed to overcome the problems listed above. Early tools such as the SHOE Knowledge Annotator [38] concentrated mainly on avoiding syntactic mistakes and typos when referencing ontologies. Current tools such a CREAM/OntoMat [36] are sophisticated authoring frameworks that support the user while writing and annotating the document and help maintaining the generated meta-data. Still, the annotation process is not integrated in the engineering process of a Web application as proposed by the Web engineering community.

3.2 XML-Based Web publishing

Web Engineering focuses on the systematic and cost efficient design, development, maintenance, and evolution of Web applications [31]. Most Web engineering methodologies are based on

separation-of-concerns to define strict roles in the development process and to enable parallel development [51]. The most frequently used concerns are the *content*, the *graphical appearance*, and the *application logic*. When we plan to design Web applications that offer semantic markup in addition, we have to introduce a new concern, the *meta-data* concern.

Most Web engineering methodologies use XML and XSLT for strict separation of content and graphical appearance. XML focuses only on the structure of the content, whereas XSLT is a powerful transformation language to translate an XML input document into an output document such as another XML document, HTML, or even plain text. Many existing Web development frameworks such as Cocoon [16] or MyXML [52] use XML and XSLT for separation-of-concerns.

Based on this technology, editors responsible for the content only have to know the structure of the XML file and the permitted elements to prepare the content pages. Similarly, designers responsible for the layout of the Web application, only have to know the structure and elements of the XML file to write the XSLT stylesheets. Finally, programmers responsible for the application logic have to generate XML documents (or fragments) as output. An XML Schema precisely defines the structure and the permitted elements in an XML file that is valid according to this schema. Therefore, XML Schema can be seen as a contract the editors, designers and programmers have to agree on [51].

Since XML is widely used in Web engineering, our approach to engineer Semantic Web applications also uses the XML content to generate the RDF meta-data description from a Web page. We also use the XML Schema as a contract and map the elements defined in the schema to concepts defined in an ontology. Our goal is to use the structure and the content of the XML document to populate the RDF triples with data.

In the proposed approach, the XML document is the basis for the HTML page as well as for the RDF description. This helps to overcome the inconsistency problem pointed out in the previous section.

3.3 Mapping XML Schema to Ontologies

In our WEESA mapping we use the content of an XML document to derive its RDF meta-data description. In the design phase of the Web application, however, we have no XML documents at hand. We do, however, have the XML Schema definition that provides us with information about the structure of valid XML documents. We use this information to define a mapping from XML elements or attributes to concepts used in an ontology. Figure 3.1 shows the definition of the WEESA mapping on the design level and how this mapping is used at the instance level to automatically generate RDF meta-data from XML documents.

In Section 3.1 we introduced the granularity problem that arises when annotating documents. We face the same problems when defining the WEESA mapping. It is possible that the concept of an XML element/attribute can be mapped one-to-one to a concept defined in an ontology. In general, however, this will not be the case. We therefore propose to dynamically compute the missing information from the information available in the XML document. In some cases

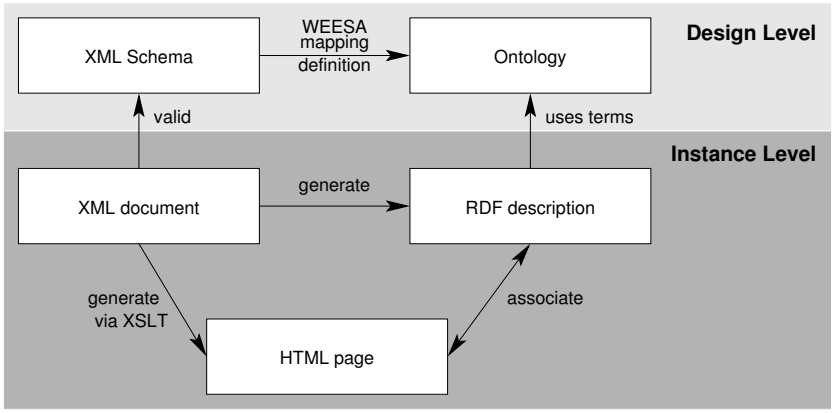

Figure 3.1: Definition of the WEESA mapping at the design level and RDF meta-data generation at the instance level.

processing is needed to reformat the element's content to match the datatype used in the ontology. In other situations it might be necessary to use the content of more than one XML element to generate the content for the RDF description.

For demonstration purpose, we use the fictitious MyTunes online CD store as an illustrative example. For each artist listed on the online store, MyTunes offers a Web page listing their albums and for each album, a page with the album details. Some XML elements such as the artist name or the track titles can be mapped one-to-one to properties defined in the corresponding class of the ontology. Other properties defined in the ontology such as the total play-time of an album cannot be found in the XML document but can be calculated from the play-times of the individual tracks. In this case some additional processing is needed to generate the information required by the ontology from the data provided by the XML document. In addition to the above, the MyTunes application offers a list of live performances for each artist. Therefore an XML document with the begin time and the duration of the performance is provided. The ontology, however, uses a different way to express the performance times, defining properties for the start and end time of a performance in the event class. Therefore the content of the begin time and the duration element have to be processed to match the two properties.

Another way in which we can address the mismatch in granularity between the XML elements and the ontology concepts is to adjust the XML Schema definition in the design phase of the Web application. The structure of the XML document could be adapted to the kind of information needed by the given ontology. This, however, would lead to several problems:

1. The data format to display information on the Web page to human users can conflict with the format needed by the meta-data representation. For example, the Web page displays a date in a human readable format such as "`Tuesday, May 24 2005, 12:00am`" but the ontology defines the range of the date property to use the XML Schema `xsi:dateTime` format (e.g. "`2005-05-24T12:00:000+01:00`").

2. Some information needed for the Web page might be no longer available in the XML document. In the example above the information that May 24, 2005 is a Tuesday is not available in the XML Schema `xsi:dateTime` format.

3. Over time a new ontology may become the standard ontology for the domain of the Web application. Therefore, the XML Schema - ontology mapping should be flexible enough to allow to change the used ontology later in the life cycle of the Web application. The change of the ontology would result in the change of the XML Schema that represents the contract agreed by all parties. This would result in the redesign of the whole Web application.

4. It is possible that we may have to define the mapping for already existing XML documents and do not have the opportunity to change the schema.

Therefore, a flexible way to map the content of one or more XML elements/attributes to the information required by the used ontology is needed. The way in which this mapping can be implemented is detailed below.

3.4 IMPLEMENTATION

The generation of RDF descriptions based on the XML content document is done in two steps. First, in the design phase for each XML Schema, a mapping to the ontologies has to be defined. Second, for each XML content document the mapping rules defined in the previous step are applied to generate the RDF representation.

3.4.1 DEFINING THE MAPPING

The starting point of the mapping is, on the one hand the XML Schema that acts as a contract in the development process, and on the other hand, the ontologies to be used. The XML Schema provides us with the information of the structure of a valid XML document and the elements and attributes being used. This information can be used to define XPath [15] expressions to select an element or attribute from an XML document. Once an element/attribute is selected, its content is mapped to a position in an RDF statement.

The goal of the mapping definition is to populate the subject, predicate and object of the RDF statements with data. In the mapping definition there are a number of ways in which the content for the RDF statements can be specified: (1) a constant value, (2) an XPath expression, (3) the return value of a Java method, and (4) a resource reference. We describe each of these methods in more detail below:

1. A constant value can be, for example, the URI reference to a concept defined in the ontology or terms from the RDF vocabulary, such as `rdf:type`, to state that a resource is an instance of a given class.

```
<?xml version="1.0" encoding="UTF-8"?>
<album id="1234">
  <artist>Alanis Morissette</artist>
  <name>Alanis Unplugged</name>
  <price>9.99</price>
  <tracks>
    <track number="1">
      <name>You Learn</name>
      <time>4:21</time>
    </track>
    <track number="2">
      <name>Joining You</name>
      <time>5:09</time>
    </track>
    <track number="3">
      <name>No Pressure over Cappuccino</name>
      <time>4:41</time>
    </track>
    <!-- ... -->
    <track number="12">
      <name>Uninvited</name>
      <time>4:37</time>
    </track>
  </tracks>
</album>
```

Figure 3.2: XML document for an album.

2. An XPath expression is used to select the content of an element/attribute in the XML document. In this case, an RDF triple is generated for each XPath match.

3. The content of more than one element/attribute might be needed to compute the information to match a property in the ontology or a datatype conversion may have to be performed. We use Java methods for this purpose. These methods take the content of one or more elements/attributes or constants as input parameters and return a string value as content for an RDF triple. In the mapping definition we are able to specify that the Java method has to be called for each XPath match, resulting in the generation of a triple for each match, or that all XPath matches are handed over as a `Vector` to the Java method, resulting in the generation of only one RDF triple.

4. Unique resource identifiers are needed to populate the subject. Since most XML documents provide more information that is related to the same resource, we offer the possibility to define a resource identifier that can later be referenced to fill the RDF triples. The mapping also allows the user to define anonymous resources. These are used for resources that never need to be referred to directly from outside the RDF description. To define an anonymous resource in the mapping, the resource is labeled as anonymous.

Figure 3.2 shows a sample XML document for an album in our MyTunes online CD store. This example is used to demonstrate the use of the four ways to specify the content for the RDF triples as described above.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping xmlns="http://www.infosys.tuwien.ac.at/WEESA#">
  <resources>
    <resource id="album">
      <method>
        <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.addPrefix</name>
        <param const="http://example.com/album#" type="java.lang.String"/>
        <param xpath="/album/@id" type="java.lang.String"/>
      </method>
    </resource>
    <resource id="track" anonymous="true" var="track_id" xpath="/album/tracks/track/@number"/>
  </resources>
  <triples>
    <triple>
      <subject ref="album"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
      <object const="http://example.com/MyTunes#Album" resource="true"/>
    </triple>
    <triple>
      <subject ref="album"/>
      <predicate const="http://example.com/MyTunes#hasTitle"/>
      <object xpath="/album/name/text()"/>
    </triple>
    <triple>
      <subject ref="album"/>
      <predicate const="http://example.com/MyTunes#totalTime"/>
      <object>
        <method>
          <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.sumTimes</name>
          <param xpath="/album/tracks/track/time/text()" type="java.util.Vector"
                 xresultAsVector="true"/>
        </method>
      </object>
    </triple>
    <triple>
      <subject ref="album"/>
      <predicate const="http://example.com/MyTunes#hasTrack"/>
      <object ref="track"/>
    </triple>
    <triple>
      <subject ref="track"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
      <object const="http://example.com/MyTunes#Track" resource="true"/>
    </triple>
    <triple>
      <subject ref="track"/>
      <predicate const="http://example.com/MyTunes#trackNumber"/>
      <object const="$$track_id$$"/>
    </triple>
    <triple>
      <subject ref="track"/>
      <predicate const="http://example.com/MyTunes#playTime"/>
      <object xpath="/album/tracks/track[@number='$$track_id$$']/time/text()"/>
    </triple>
    <triple>
      <subject ref="track"/>
      <predicate const="http://example.com/MyTunes#trackTitle"/>
      <object xpath="/album/tracks/track[@number='$$track_id$$']/name/text()"/>
    </triple>
  </triples>
</mapping>
```

Figure 3.3: WEESA mapping definition for the album example.

```
Class: Artist                        Class: Track
  ->hasArtistName                      ->trackTitle
  ->hasAlbum (range: Album)            ->playTime
                                       ->trackNumber

Class: Album                         Class: Event
  ->hasTitle                           ->hasEventName
  ->hasTrack (range: Track)            ->hasLocation
  ->totalTime                          ->beginTime
  ->year                               ->endTime
```

Figure 3.4: Ontology used for our MyTunes example.

A WEESA mapping definition (see Figure 3.3) consists of two parts. In the first part, we define the resource identifiers that may later be used. In the second part the subject, predicate, and object of the actual RDF triples are defined. The XML Schema for the WEESA mapping definition is shown in Appendix A.

Figure 3.3 shows the WEESA mapping definition for our album example. At the beginning of the mapping we define the resources (lines 3-12). In the first resource with the `id="album"` attribute we define an XPath expression to select the `id` attribute of the `<album>` element in Figure 3.2. The according XPath expression is `/album/@id`. The content of the attribute is then handed over to a Java method. The method name is defined in line 6. In this case the Java method adds a prefix to the attribute value to generate a resource identifier. The parameters for the method are defined in line 7 and 8. The first parameter is a constant used for the prefix and the second parameter is the XPath expression to select the attribute. The return value of the method is then used as the content in the RDF triple whenever the resource with the `id="album"` is referenced.

In the resource with the `id="track"` (line 11) we show how to define an anonymous resource. This is done using the `anonymous="true"` attribute. In this case, for each XPath match an anonymous resource is generated. This resource definition also contains the `var` attribute which will be discussed later in this section.

Once we have defined the resources, we can start defining the RDF triples. This is done in the `<triples>` section (lines 13-60). In the first triple (lines 14-18) the subject uses the `ref="album"` attribute to reference the resource with the `id="album"`. In the predicate we use the `rdf:type` constant to define the class the subject is an instance of. The object of this triple is the URI reference to the class in the ontology (`http://example.com/MyTunes#Album`). The `resource="true"` attribute is used to indicate that the value of the object should be interpreted as an RDF resource. The default interpretation would be a literal. Our sample ontology is shown in Figure 3.4. For easy understanding and clarity, we do not use the OWL Syntax in the example, but use a trivial textual syntax instead. The MyTunes sample ontology in OWL syntax can be found in Appendix B.

The following triples in our mapping example fill the properties of the `#Album` class. The predicate defines the name of the property and the object the value. The `xpath` attribute of the `<object>` element defines the XPath expression that has to be evaluated. The object element

can also contain a `<method>` element to define the Java method to compute the content of the object. Lines 28-32 show the use of a Java method where all XPath matches are handed over as a `Vector` to the method indicated by the `xresultAsVector="true"` attribute. The `sumTimes` Java method takes the individual play-times of the tracks in a `Vector` as parameter and calculates the total play-time of the album.

In some cases we need additional information to select a specific element/attribute by an XPath expression. When an XML document consists of multiple elements with the same name at the same hierarchy level, we need a technique to select a specific element. For this purpose we use variables. In line 11 we use the `var` attribute to define the `track_id` variable. This variable can be used in any constant or XPath expression using the `$$` escape sequence at the beginning and the end of the variable name. The variable is replaced at runtime with the actual value. Variables can be defined together with the `xpath` attribute in resource definitions only.

At the end of the triples section we show that anonymous resources can be used in the same way as any other resources in the triple definition. In the triple defined in lines 35-39 the object uses the reference to the anonymous resource `track`. The following triples define the class and the properties for this resource and show the use of the `track_id` variable in the WEESA mapping.

So far, we have shown how the WEESA mapping is defined. The following section shows how this mapping can be used to generate RDF descriptions from XML documents.

3.4.2 GENERATING THE RDF DESCRIPTION

When a Web page is queried the corresponding XML document is fetched and first, the XSLT transformation is used to generate the HTML page. Second, the RDF description has to be generated and associated with the HTML page. The way in which the RDF description is generated is discussed in the this section.

To generate the RDF description all mappings defined for the XML document have to be executed. Therefore, the XPath expressions have to be evaluated on the document and the result is either used directly to populate a position in an RDF triple that is defined in the mapping or is handed over to a Java method first. If the XPath expression matches multiple elements/attributes in the XML document the procedure must be repeated for each match or alternatively, the matches are handed over to the Java method as a `Vector`.

The pseudo-code in Figure 3.5 shows the principle steps that have to be applied to process a WEESA mapping definition. Before the `recursiveMapping()` method can be called all dependencies between the defined resources have to be analyzed and stored in the global `resDependencyHash`. A resource R_1 is dependent on resource R_2 if R_1 uses a variable that is defined in R_2 or R_1 is used as object in a triple with the subject R_2. In this case R_2 has to be processed before R_1 can be resolved. The second condition is necessary since triples that relate two resources (as defined in lines 34-38) can only be generated if the values for both resources have been processed. The initial parameter for the `recursiveMapping()` method is the `Set` of resources that do not depend on any other resources and can therefore be processed directly.

```
1  recursiveMapping(resourceSet)
2    if resourceSet is empty
3      generateTriples()
4      return
5
6    forall resource in resourceSet
7      if there are XPath matches
8        resourceStack.push(all XPath matches)
9        stackHash.put(resource, resourceStack)
10
11       while (stack is not empty)
12         xpath_match = stack.pop()
13         content = processContent(xpath_match)
14         globaleResourceHash.put(resource, content)
15         if resource defines variable
16           globaleVariableHash.put(variable, xpath_match)
17         recursiveMapping(resDependencyHash.get(resource))
18     else
19       recursiveMapping(empty Set)
```

Figure 3.5: Pseudo-code for processing the WEESA mapping.

For better understanding, the MyTunes mapping example from Figure 3.3 is used to illustrate the execution of the pseudo-code. We start by analyzing the resource dependencies and discover that the triple defined in lines 34-38 uses the resource `album` as subject and `track` as object. Following the rule defined above, `track` depends on the resource `album`. This leaves us with `album` as the only independent resource.

Now the method `recursiveMapping()` is called with `album` as parameter. Since the `resourceSet` is not empty the processing continues with the two nested loops. In the outer loop the XPath expression is executed on the XML document and the matches are put on the stack. In our example the XPath `/album/@id` has only one match: `1234`.

If there are XPath matches, the inner loop is processed. In the inner loop, the XPath matches are taken from the stack, the `processContent()` method is called, the variables are assigned their values, and a recursive method call is performed to process all resources that depend on the current resource/variable environment. The `processContent()` method takes the XPath matches as parameter and computes the content for the resource as defined in the mapping. If a Java method is defined, it causes the method call and uses the return value as content. In the case that the Java method encounters a problem, it will return the value `null` and no triple will be generated. If there are no more dependent resources, the recursive call is done with an empty `Set`. This causes the call of the `generateTriples()` method. The `generateTriples()` method iterates through all triples defined in the mapping and generates an instance for those where the required resources and variables are defined in the `globalVariableHash` and `globalResourceHash`.

Returning to our example, in the inner loop we take the first match (`1234`) from the stack, call the Java method defined in line 6, compute the content for the resource (`http://example.com/album#1234`), and do the recursive method call. The `resourceSet` for the recursive call contains the `track` resource since it depends on the `author` resource as explained above. In the recursive method call the execution of the XPath expression returns with a match for

each track on the album and in the inner loop all the matches are processed. Since no other resource depends on the `track` resource the recursive call is done with an empty `Set` and the `generateTriples()` method is called. A part of the generated RDF graph for our example may be found in Figure 3.6.

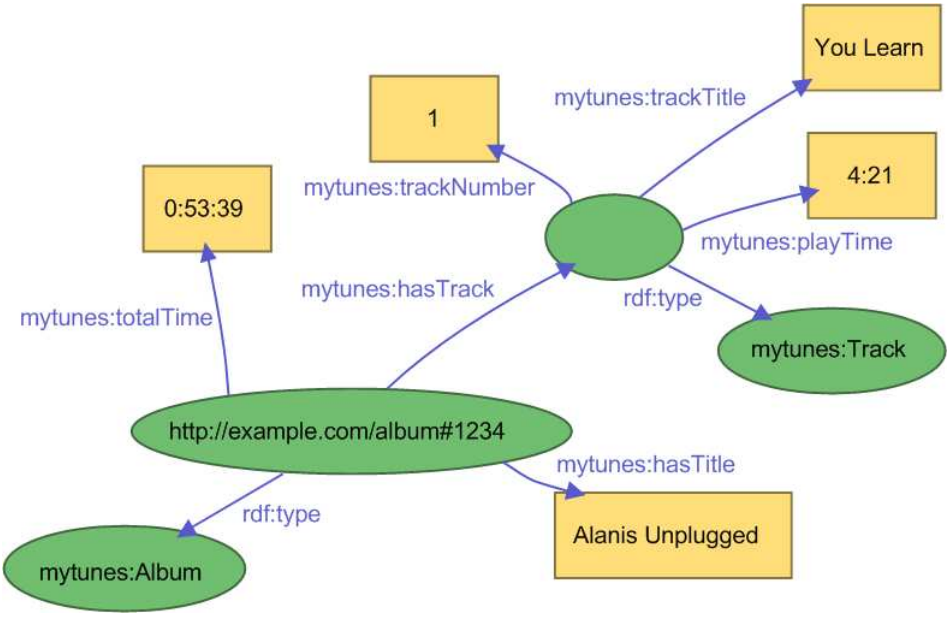

Figure 3.6: Snippet from the generated RDF graph.

3.4.3 Constraints of the WEESA implementation

In this section we discuss two known limitations of the current WEESA prototype implementations and show work-abounds where possible:

Permutations of Method parameters

If a method call is defined that takes several XPath expressions as parameter, the WEESA prototype does not generate all permutations of the matches of the XPath expressions. Instead we use the `iter="true"` attribute to indicate over which XPath matches should be iterated. For the other parameters we use the first XPath match.

In our experience, this is not a critical constraint, since in the case we have an XML document where all permutations of the matches from two XPath expressions from are needed, there is typically a subject-predicate-object relationship between the XML structures referenced by the XPath expressions. To express this relation we do not need both XPath expressions in a method call, but one XPath expression is used in the resource definition for the subject and the second XPath expression is used for the object. This way, all permutations are generated.

CIRCLES IN THE RESOURCE DEPENDENCY CHAIN

Because of the dependency rule between resources, introduced in the section above, WEESA cannot generate RDF graphs that include circles. In this case no independent resource can be found to start the recursion. The WEESA prototype implementation prints out a warning, when a circle is detected.

As work-around we recommend to break the circle of dependencies when specifying the WEESA mapping definition. This can be done if one triple does not use the reference attribute `ref="resource name"` as object but the constant, XPath expression, or method definition to generate the resource identifier for the object. This work around has the disadvantage that we cannot reuse an existing resource definition but helps to break the dependency chain.

3.4.4 COMPLEXITY

In the previous section we introduced the algorithm used in the WEESA prototype to process the WEESA mapping definition to generate RDF meta-data from XML content documents. In this section we present an estimation of the upper complexity bound of this algorithm.

As described above, before the WEESA mapping algorithm can be processed the resource dependencies must be analyzed. The mapping algorithm introduced in Figure 3.5 is a recursive algorithm. In the first recursion, all independent resources are processes. In each recursion, the two nested loops are processed. The outer loop iterates through the resources to be processed in this step, and the inner loop processes the XPath matches for the resource under consideration. Finally, when a recursive method call is done. If this call is done with an empty resource set, the recursion ends and the RDF triples are generated. Therefore we result with the following numbers that have to be considered when estimating the upper complexity bound:

- The number of steps to analyze the resource dependencies.
- The number of recursions to process the dependent resources.
- The number of times the outer loop has to be processed.
- The number of times the inner loop has to be processed.
- The number of triples that have to be generated for the given resource/variable environment.

To be able to estimate the upper complexity bound of the WEESA mapping algorithm, we first analyze the upper bound of the numbers listed above. To do so, we introduce a formal notation for the dependency between two resources. If R_1 and R_2 are resources and resource R_2 depends on R_1, we write $R_1 \rightarrow R_2$.

In the first step to process the WEESA mapping the resource dependencies are analyzed. The complexity of this step is determined by the number of resources n_{res} plus the number of triples n_{tripl} in the mapping definition. Therefore we have the upper complexity bound $O(n_{res}+n_{triple})$.

The complexity of the `recursiveMapping()` method is made up by the following iterations:

1. The number of recursions is determined by the dependencies between the resources. The longest possible dependency chain can be found when starting from one independent resource, all resources depend on exactly one resource, and no resource pair is dependent on the same resource. For example assume, in the mapping definition the resources R_1, R_2, and R_3 are defined. An example for the longest possible dependency chain with the independent resource R_1 is $R_1 \rightarrow R_2 \rightarrow R_3$. The length of longest dependency chain is therefore equal to the number of resources defined in the WEESA mapping definition n_{res} and corresponds to the upper bound of the number of recursions.

   In outer loop the set of resources of the current recursion step are processed. Since every resource defined in the mapping is at most one time part of this set, the upper bound of times the loop is processed is equal to the number of resources defined in the mapping n_{res}. In this loop the XPath expression is evaluated on the XML document and the XPath matches are put on the stack.

   If we analyze the number of recursions and the number of iterations of the outer loop in more detail, we realize that an increasing number of recursions results in a decreasing number of iterations of the outer loop, and vice versa. Let us discuss different cases in an example with four resources R_1, R_2, R_3, and R_4 defined.

   - In the first case we assume to have four independent resources. The `recursiveMapping()` call is done with the set containing the four elements R_1, R_2, R_3, and R_4. Therefore the inner loop is processed four times to iterate through the four resources. The recursive method is only called one time.
   - In the next case we assume to have the longest possible dependency chain $R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_4$ starting with the independent resource R_1. The first recursion call is done with the set containing only the element R_1 and the outer loop is processed only one time. The second recursive call is done with R_2 in the set and the outer loop is processed on time and so on. All together, we realize the outer loop is processed four times.
   - In the last case, let us assume the following resource dependencies: $R_1 \rightarrow R_2$, $R_2 \rightarrow R_3$, and $R_2 \rightarrow R_4$. The first recursive call is done with the element R_1 in the set and the outer loop is processed one time. The second recursive call is done with the element R_2 and the loop is processed one time. The third recursion is done with the set elements R_3 and R_4 and the outer loop is processed two times. Again, the outer loop is overall processed four times.

   The discussion of the cases above shows that the dependencies between the resources and the recursion ensures, that the outer loop is processed exactly one time for each resource in the WEESA mapping definition. The upper bound of the number of recursions and iterations of the outer loop is therefore n_{res}.

2. In the inner loop the XPath matches on the stack are processed to compute the content for the resources. Therefore the number of runs of the inner loop is equal to the number of XPath matches found for the resource R_i and is written as $n_{XPath}(R_i)$.

3. After processing the content for the resource R_i in the inner loop the recursive method call is done. This leads either to the processing of a dependent resource or to the generation of the RDF triple in the `generateTriples()` method. The case of the processing of a dependent resource is already covered in the discussion of item 1 above. In the `generateTriples()` method the algorithm iterates through the list of all triples defined in the WEESA mapping definition. The number of triples is abbreviated as n_{triple}. The XPath expressions are evaluated on the XML document and an RDF triple is generated for each match. The number of XPath matches for the triple T_k when resource R_i is processed is written as $n_{XPath}(R_i, T_k)$.

When we put together the results of the complexity discussion from above, we get the upper bound complexity shown in Formula 3.1. The first part results from the dependency analysis, and the second part from the RDF graph generation in the `recursiveMapping()` method.

$$\underbrace{O\big(n_{res} + n_{triple}\big)}_{\text{analyze dependencies}} + \underbrace{O\Bigg(\overbrace{\sum_{i=1}^{n_{res}}}^{\text{recursion + outer loop}} \overbrace{\sum_{j=1}^{n_{XPath}(R_i)}}^{\text{inner loop}} \overbrace{\sum_{k=1}^{n_{triple}} n_{XPath}(R_i, T_k)}^{\text{gernerate triples}}\Bigg)}_{\text{recursive mapping algorithm}} \tag{3.1}$$

Since the contribution to the complexity of the dependency analysis is asymptotic smaller than the complexity of the RDF graph generation, the upper bound complexity of the WEESA mapping algorithm can be simplified to:

$$O\Bigg(\sum_{i=1}^{n_{res}} \sum_{j=1}^{n_{XPath}(R_i)} \sum_{k=1}^{n_{triple}} n_{XPath}(R_i, T_k)\Bigg) \tag{3.2}$$

The result we got from the analysis of the pseudo-code in Figure 3.5 can also be explained when we analyze the generated RDF graph. $n_{XPath}(R_i, T_k)$ represents the number how many times the statement (triple) T_k is made about the resource R_i in the XML document. The sum $\sum_{k=1}^{n_{triple}}$ states that we have to check all n_{triple} triples whether a statement about the resource R_i is made. $n_{XPath}(R_i)$ represents the number of times an instance of the resource R_i can be found in the XML document. RDF statements have to be generated for each instance of R_i ($\sum_{j=1}^{n_{XPath}(R_i)}$). The sum $\sum_{i=1}^{n_{res}}$ states that we have to iterate through all n_{res} resources R_i defined in the mapping definition.

To further simplify the formula for the upper complexity bound we make the following assumptions. We have a given XML document and define n_{XML_res} as the maximum number of

XPath matches for a resource in the XML document, and n_{XML_triple} as the maximum number of XPath matches for a triple in the XML document.

$$\begin{aligned} n_{XML\_res} &= \max(n_{XPath}(R_i)) \quad i \in [1, n_{res}] \\ n_{XML\_triple} &= \max(n_{XPath}(R_i, T_k)) \quad i \in [1, n_{res}], k \in [1, n_{triple}] \end{aligned}$$

These assumptions simplify the upper complexity bound from Formula 3.2 to:

$$O(n_{res} \cdot n_{XML\_res} \cdot n_{triple} \cdot n_{XML\_triple}) \tag{3.3}$$

This result is intuitive, since the complexity depends on the number of resources and triples defined in the WEESA mapping definition, the number of resource instances that can be found in the XML document, and the number of statements used to describe the resources.

3.5 RESUME

In this chapter we started with an introduction into semantic annotation and XML-based Web engineering and pointed out the deficits of the approaches proposed in these research areas when designing and implementing Semantic Web applications. To address this problem we introduced WEESA that is based on the experiences of semantic annotation and follows the well established Web engineering principle of separation-of-concerns. Many Web engineering methodologies and Web development frameworks are based on XML content documents. WEESA reuses these existing engineering artifacts for the semantic annotation process and can therefore be integrated in existing XML-based Web engineering methodologies. In WEESA the XML document is the data source to generate the HTML Web page for human consumption and the RDF meta-data description representing the content of the Web page machine-processable.

XML Schema defines the structure of valid XML documents and can therefore be seen as the contract in XML-based Web engineering methodologies all parties involved development process have to agree on. WEESA uses the XML Schema and defines a mapping from the XML elements and attributes to the concepts defined in an ontology. This is done in the design phase of the Web application. In the implementation of the WEESA Semantic Web application the WEESA mapping definition is taken to automatically generate RDF meta-data descriptions from XML content documents.

WEESA can also be applied together with XML-based Web engineering methodologies that do not use XML Schema contracts. In this case, no formal contract for the Web application is defined. In some way, the structure of the XML document, the XSLT stylesheet is applied on, has to be defined. This can be done, for example, by defining a representative XML document. Such an example document can be taken to define the XPath expressions to select the XML elements/attributes to map them onto concepts defined in an ontology.

In this chapter we introduced the WEESA idea, the WEESA mapping definition and the prototype implementation of the WEESA RDF meta-data generator. Opposite to other semantic annotation tools, WEESA not only supports one-to-one mappings between the XML document and the ontology, but uses a more flexible approach. Java methods are used to enable data format conversations from the human-readable format needed for the Web page to the machine-processable format for the RDF meta-data description. The Java methods further enable the computation of information required for the meta-data description from the data available in the XML document.

During life time of a Web application a new version of the used ontology may be issued or a new ontology become the standard ontology to describe information in the domain of the Web application. In this case only the WEESA mapping definition has to be adopted to the new ontology, the rest of the Web application remains unchanged.

At the end of this chapter, we discussed the asymptotic complexity of the proposed WEESA mapping algorithm and proved that the complexity depends on the number of resources and triples defined in the mapping definition, the number of resource instances given in the XML document, and the number of RDF statements used to describe these resources.

CHAPTER 4

SEMANTIC WEB APPLICATIONS WITH WEESA AND APACHE COCOON

A complex system that works
is invariably found to have evolved
from a simple system that worked.

John Gall

In the previous chapter we introduced the WEESA mapping and illustrated how this mapping can be used to automatically generate RDF meta-data from XML content documents. In this chapter we show how WEESA is used to develop Semantic Web applications. To generate the semantically annotated Web pages we have to perform two steps on the schema valid XML document:

1. The XSLT transformation has to be performed to generate the HTML page.
2. The WEESA mapping has to be processed to generate the RDF meta-data description of the Web page.

Once we have generated the HTML page and its RDF meta-data description we have to put them into relation. Unfortunately, no standardized approach exists for associating RDF descriptions with HTML. In [71] Palmer discusses several possible approaches. In this chapter we show how some of the proposed associations can be implemented using WEESA.

Many Web development frameworks exist that are based on XML/XSLT that can be extended by WEESA to develop Semantic Web applications. In this thesis we show how we integrated WEESA into Apache Cocoon transformer components and the use of these transformers in Apache Cocoon Web applications.

4.1 Apache Cocoon Web Applications

In this section we briefly introduce the concept of Apache Cocoon Web applications. Apache Cocoon [16] is a Web development framework built around the concepts of separation-of-concerns and component-based Web development. Cocoon uses component pipelines to build Web applications where each component on the pipeline is specialized on a particular operation.

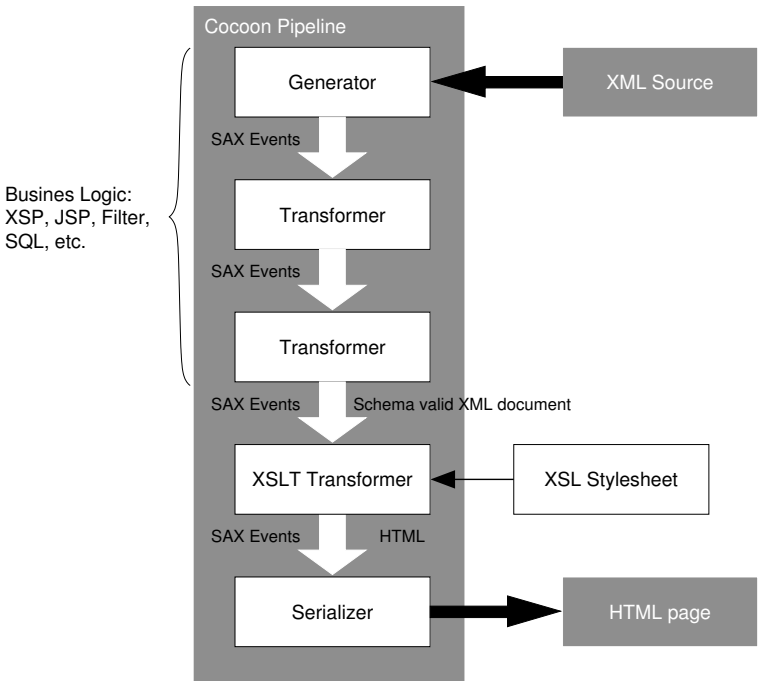

Figure 4.1: Pipeline of a typical Cocoon Web application

Figure 4.1 shows the pipeline of a typical Cocoon Web application. A Cocoon pipeline consists of a *generator*, an arbitrary number of *transformer*, and a *serializer*. An HTTP request triggers the pipeline and causes the generator to read XML from a data source and produces a stream of SAX[1] [87] events as output representing the XML document. This output is the input of a transformer or a serializer. A transformer takes the SAX events, does some transformation (e.g. XSLT transformation), and the results are again SAX events. These events can then be taken by another transformer or a serializer. In a typical Cocoon Web application, the business logic (e.g. SQL queries, Java code) is processed by the transformers at the beginning of the pipeline. The output of the business logic is a schema valid XML document that fulfills the Web engineering contract and has the structure the XSLT stylesheet was designed for. This document is then taken by the XSLT transformer which uses the XSLT stylesheet to produce the HTML page. The serializer finally takes the SAX events and processes them into a character stream for client consumption.

[1] SAX: Simple API for XML

4.2 WEESA COCOON TRANSFORMER

In this section we introduce two new Cocoon transformers in which the WEESA meta-data generator has been integrated. These transformers enable developers to realize Semantic Web applications using the Cocoon Web development framework. Before presenting the WEESA enabled cocoon transformer, we discuss ways to associate the HTML Web page with its RDF meta-data description.

4.2.1 ASSOCIATING HTML AND RDF

Several techniques have been proposed to associate the HTML page and the RDF meta-data description with one another [71]. These techniques can be classified in the following categories:

Embedding RDF in HTML: With this association style the RDF description is directly embedded in the HTML page. Several ways have been proposed such as adding RDF/XML in the `<script>` element, using XML notations and `CDATA` sections, or adding RDF base64 encoded in the `<object>` element. A detailed discussion of these methods can be found in [71].

Linking to an external RDF description: The RDF meta-data description is stored in an external document and the HTML page references to its meta-data description. This reference can be done using the HTML `<link>` element [71] or using an common HTML `<a href="">` link [37].

In the following two sections we introduce a WEESA enabled Cocoon transformer for both categories of association styles. We introduce the `WEESAReadDOMSession` transformer that embeds RDF/XML in the `<script>` element and the `AddRDFLink` transformer that adds the `<link>` element to an HTML Web page and references an external RDF description generated with the help of the `WEESA` transformer.

4.2.2 WEESA COCOON TRANSFORMER TO GENERATE HTML+RDF

One way to associate HTML and RDF is to embed the RDF/XML description in the `<script>` element within the HTML `<head>`. A fragment of such a HTML document is shown in Figure 4.2. The HTML `<script>` element can be used to include non-HTML media in HTML Web pages [41]. This section introduces the `WEESAReadDOMSession` transformer that can be used to realize this kind of RDF – HTML association.

When developing Semantic Web applications that add the RDF meta-data to the HTML page we have to introduce new steps to the pipeline discussed in Section 4.1. Since we need the schema valid XML document for the XSLT transformation and for the WEESA meta-data generation, we have two options. We can either integrate WEESA in a modified XSLT transformer that

```
<head>
  <title>My Document</title>
  <script type="application/rdf+xml">
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:dc="http://purl.org/dc/elements/1.1/">
      <rdf:Description rdf:about="http://www.w3.org/" dc:title="W3C Homepage"/>
    </rdf:RDF>
  </script>
</head>
```

Figure 4.2: RDF meta-data included in the HTML `<script>` element

generates RDF and HTML or we can split up the pipeline and use a specialized transformer for the RDF meta-data generation. One of the strength of Cocoon is, that every component is specialized on a specific task. This gives the developer the flexibility to configure the components in the pipeline according to his requirements. Therefore we decided, to split up the pipeline to generate HTML pages with embedded RDF/XML.

To split up the pipeline we use the `WriteDOMSession` transformer. This transformer takes the input document and writes it first as DOM[2] [23] into the servlet session, and second as SAX events to its output. This is how the pipeline is split up and the XML document can be reused later in the pipeline. After the HTML page is generated by the XSLT transformer the `WEESAReadDOMSession` transformer takes the DOM-XML from the session and uses the WEESA mapping definition to generate the RDF meta-data representation in RDF/XML format. The `WEESAReadDOMSession` transformer further embeds the generated RDF/XML in the `<script>` element which is then added in a user defined element of the HTML page. This element is typically the `<head>` element. The serializer finally delivers the HTML+RDF page to the client. The additional steps are shown in Figure 4.3 as light gray pipeline components. The problems that comes with embedding the RDF meta-data in the `<head>` element are discussed at the end of the this section.

The Cocoon framework uses the `sitemap.xmap` configuration file to define the pipelines. Figure 4.4 shows a fragment of the sitemap file for the pipeline from Figure 4.3. Lines 3-6 instruct Cocoon to start a servlet session. In line 9 the generator is instructed to read the XML document `AlanisMorissetteUnplugged.xml` from the `content/` directory. In lines 14-17 the `WriteDOMSession` transformer is defined to write the XML document to the session. The `dom-name` parameter gives the DOM-object the name `rdf` in the servlet session. The use of the `dom-root-element` parameter is explained below. Line 20 configures the XSLT transformer to use the XSLT stylesheet `albumInfo.xslt`. In lines 23-29 the `WEESAReadDOMSession` transformer is configured. The `dom-name` parameter tells which DOM-object should be taken from the session, the `weesa-mapping-definition` parameter names the mapping file, and the `trigger-element` and `position` parameters tell that the generated RDF/XML should be placed `in` the HTML `<head></head>` element. Finally, in line 32 the serializer is instructed to write the XML stream as `HTML` to the consumer.

In praxis, most Web pages consist of several parts, for example a header, the navigation

[2]DOM: Document Object Model

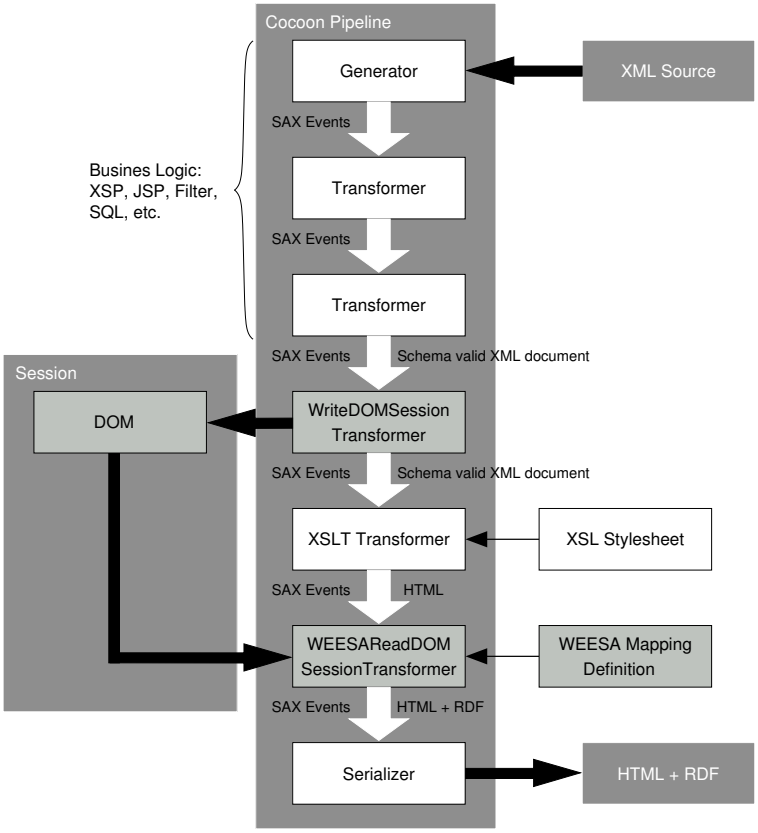

Figure 4.3: Cocoon pipeline for the WEESA HTML+RDF generation. The light gray components are added to embed RDF/XML in the HTML Web page.

```
1  <map:match pattern="AlanisMorissetteUnplugged.html">
2    <!-- create the session -->
3    <map:act type="session">
4      <map:parameter name="action" value="create"/>
5      <map:parameter name="mode" value="immediately"/>
6    </map:act>
7
8    <!-- read the XML file -->
9    <map:generate type="file" src="content/AlanisMorissetteUnplugged.xml"/>
10
11   <!-- here comes the business logic (if needed) -->
12
13   <!-- write the XML document to the session -->
14   <map:transform type="writeDOMsession">
15     <map:parameter name="dom-name" value="rdf"/>
16     <map:parameter name="dom-root-element" value="content"/>
17   </map:transform>
18
19   <!-- do the XSLT transformation -->
20   <map:transform type="xslt" src="style/albumInfo.xslt"/>
21
22   <!-- configure and start the WEESAReadDOMSession transformer -->
23   <map:transform type="WEESAReadDOMSession">
24     <map:parameter name="dom-name" value="rdf"/>
25     <map:parameter name="trigger-element" value="head"/>
26     <map:parameter name="position" value="in"/>
27     <map:parameter name="weesa-mapping-definition"
28                    value="mapping/albumMapping.xml"/>
29   </map:transform>
30
31   <!-- serialize the output -->
32   <map:serialize type="html"/>
33 </map:match>
```

Figure 4.4: Pipeline definition using the `WEESAReadDOMSession` transformer

part, and a part with the actual page content. A sample XML document following this structure is shown in Figure 4.5. Depending on the structure of the Web page, not all parts of the XML document have to be looked at when generating the RDF meta-data. For example, only the `<content>` part contains information that is needed for the meta-data generation. In this case, we have to define the WEESA mapping only for the subtree starting with the `<content>` element and we have to inform the `WriteDOMSession` transformer only to write the subtree following the `content` element as DOM to the session. This is done with the `dom-root-element` parameter in line 16 from Figure 4.4.

Including RDF/XML meta-data in the HTML page using the `WEESAReadDOMSession` transformer has the advantage that the business logic, typically processed at the beginning of the pipeline, has to be computed once only for both the HTML and RDF generation. Embedding the RDF/XML meta-data in the `<head>` tag of a HTML document, however, breaks HTML 4.01 [76] and XHTML [99] validity [78].

```
<?xml version="1.0" encoding="UTF-8"?>
<page>
  <header>
    <!-- here goes the XML for the header -->
  </header>
  <navigation>
    <!-- here goes the XML for the site navigation -->
  </navigation>
  <content>
    <!-- here goes the XML for the content of the Web page -->
  </content>
</page>
```

Figure 4.5: Sample XML document aggregated from several parts.

4.2.3 WEESA COCOON TRANSFORMER TO GENERATE RDF/XML

Another way of associating RDF and HTML to one another is to use the `<link>` element in the `<head>` of the HTML page to reference the corresponding external RDF/XML meta-data description [71]. To generate this stand-alone RDF/XML description within Cocoon Web applications we developed the `WEESA` transformer.

Cocoon Web applications that use the `<link>` element to associate RDF and HTML need two pipelines: one for the generation of the HTML page, and another for the WEESA meta-data generation. The pipeline for the HTML generation is similar to the one introduced in Section 4.1. Only the reference to the RDF description has to be added in the `<head>` of the HTML page. The reference looks as follows:

```
<link rel="meta" type="application/rdf+xml"
                 href="AlanisMorissetteUnplugged.rdf"/>
```

There are basically two ways of adding the `<link>` element to the HTML page. One is to modify the Web application to add the element in the business logic or in the XSLT stylesheet. The other possibility is to use the `AddRDFLink` transformer we developed. This transformer is added to the pipeline for the HTML page generation between the XSLT transformer and the serializer. The use of the `AddRDFLink` transformer is shown in Figure 4.6. The `AddRDFLink` transformer extracts the URL of the incoming request, replaces the `".html"` suffix of the path with `".rdf"`, and adds the `<link>` with the `".rdf"` URL in the `<head>` of the HTML page. The request parameters in the URL remain untouched. The fragment of the `sitemap.xmap` configuration file that defines the `AddRDFLink` transformer in a pipeline is shown in Figure 4.7.

Since the `AddRDFLink` transformer searches for a `".html"` suffix in the URL and replaces it with `".rdf"` it can only be used in Web applications that obey the following naming convention. The path in URLs that trigger the pipeline for the HTML pages have the suffix `".html"` such as:

```
http://www.mytunes.com/album.html?id=1234
```

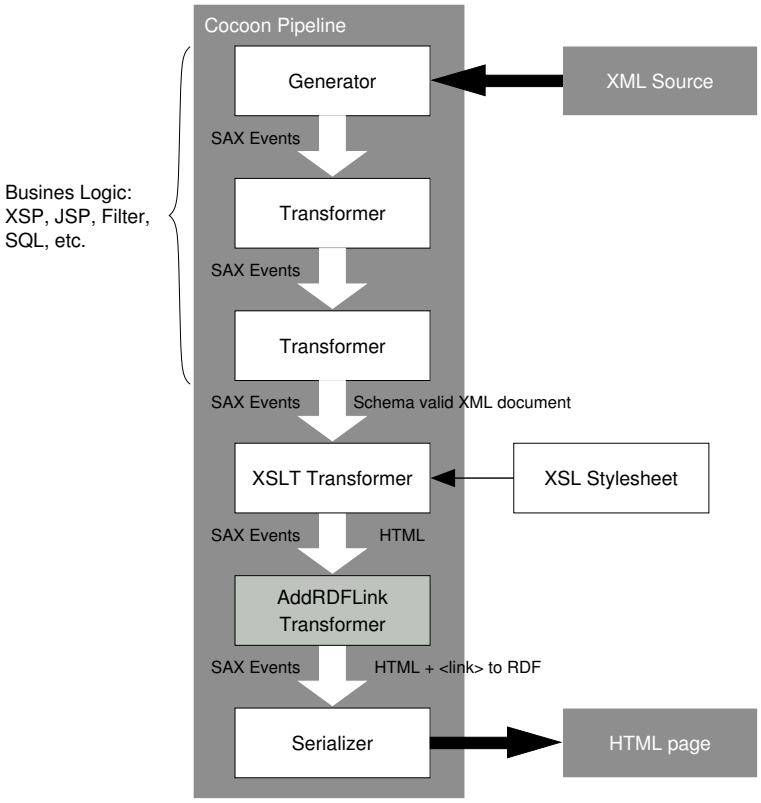

Figure 4.6: Pipeline for the HTML page generation that uses the `AddRDFLink` transformer to add the `<link>` element to the HTML page.

```
<map:transform type="AddRDFLink"/>
```

Figure 4.7: Configuration of the `AddRDFLink` transformer.

The path in URLs that trigger the pipeline for the RDF meta-data generation have the suffix `".rdf"` such as:

```
http://www.mytunes.com/album.rdf?id=1234
```

In the pipeline for the WEESA meta-data generation that uses the WEESA transformer the business logic is also processed at the beginning of the pipeline. The schema valid XML document is sent to the `WEESA` transformer that takes the mapping definition and processes the WEESA mapping. The RDF/XML output of the transformer is then taken by a serializer and sent to the client. The pipeline is shown in Figure 4.8 and Figure 4.9 shows a snipped from `sitemap.xmap` that is used to configure the `WEESA` transformer. Again, the `dom-root-element` parameter defines the start element of the XML subtree that should be considered for the WEESA transformation as described in the section above, and the

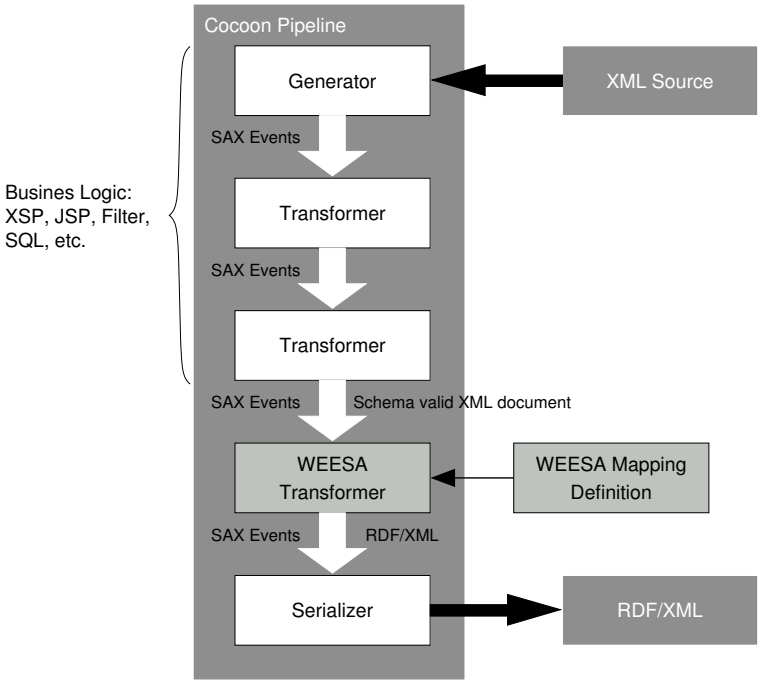

Figure 4.8: Cocoon pipeline for the WEESA RDF/XML generation.

`weesa-mapping-definition` parameter defines the mapping definition to be used.

```
<map:transform type="WEESA">
  <map:parameter name="dom-root-element" value="content"/>
  <map:parameter name="weesa-mapping-definition"
                 value="mapping/artistMapping.xml"/>
</map:transform>
```

Figure 4.9: Configuration of the `WEESA` transformer.

Using the `<link>` element to associate RDF and HTML has the advantage that the RDF description has to be generated on request only. This, however, has the drawback that the schema valid XML document has to be generated a second time. Fortunately this does not have to be a big disadvantage if the Cocoon caching mechanism can be used.

Although there is still no standard approach, using the `<link>` element to associate RDF and HTML is the recommended approach on the "RDF Issue Tracking" homepage [79]. The used `application/rdf+xml` media type (MIME type) has been defined in the IETF RFC 3870 [85]. This approach is for example used for the homepage of the mindswap project[3] which describes itself as the "first site on the semantic web" [64].

At the AIFB Karlsruhe homepage[4] a standard HTML `<a href="..."></a>` link is used

[3] `http://www.mindswap.org/`
[4] `http://www.aifb.uni-karlsruhe.de/`

to reference the meta-data description of a Web page. The image used for the link is shown in Figure 4.10. The AIFB homepage was developed within the SEAL (SEmantic portAL) project [37] which aims to provide a conceptual framework for semantic portals. This way of associating RDF and HTML can also be done using the `WEESA` transformer. The HTML link simply has to reference the pipeline for the RDF/XML meta-data generation.

Figure 4.10: OWL/RDF logo to reference the meta-data description.

4.3 RESUME

Many XML/XSLT based Web development frameworks exist that can be extended by WEESA to simplify the development of Semantic Web applications. In this chapter we presented the integration of the WEESA meta-data generator into the Apache Cocoon Web application development framework. WEESA, however, can also be integrated into other frameworks such as the MyXML Web development kit [52] provides a template language to define static and dynamic Web page components that are used to compose Web pages. The PageKit Web application framework [70] is an Apache HTTP server module that is based on the `mod_perl` module which uses templates and XML/XSLT to separate the model, view, content, and controller. The GlueX framework [33] is an XML/XSLT-centric library for building Java servlet-based Web applications.

Two techniques exist to associate the HTML Web page with its RDF meta-data description: The HTML page can embed its RDF meta-data description, or link to an external document containing the its RDF description. We developed a WEESA enabled transformer component for each association style. The `WEESAReadDOMSession` transformer is used to embed RDF into HTML. The `WEESA` transformer is used to generate an external RDF description that can be linked to the HTML page. To add this link the `AddRDFLink` can be used.

Since WEESA follows the principle of separation-of-concerns and the component architecture of Cocoon no additional programming is needed for developing a Semantic Web application compared to the development of a traditional Web application. In the development process only two new steps are needed: (1) The WEESA mapping from the XML Schema to the ontology has to be defined. (2) Depending on the association style between HTML and RDF, the developer has to modify the pipeline and add the corresponding WEESA enabled transformer component. Detailed guidelines for developing WEESA Semantic Web applications can be found in Chapter 7.

CHAPTER 5

BUILDING THE KNOWLEDGE BASE OF THE SEMANTIC WEB APPLICATION

New ideas pass through three periods:
- It can't be done.
- It probably can be done, but it's not worth doing.
- I knew it was a good idea all along!

Arthur C. Clarke

In the previous chapters we introduced the WEESA mapping and showed how this mapping can be used in Cocoon Web applications to annotate single Web pages to develop Semantic Web applications. Looking at the meta-data of a single Web page, however, gives only a limited view of the information available in a Web application. For querying and reasoning purposes it would be better to have the full meta-data model of the Web application at hand. Systems proposed in literature such as SHOE [39], CREAM [36], Ontobroker [28], and SEAL [60] use a Web crawler to collect the meta-data form single Web pages to build the meta-data model of a Web application. Using a crawler has the advantage that the Web application and the application using the collected meta-data are loosely coupled and can be distributed over the Internet. This loose coupling, however, has the disadvantage that the meta-data model does not recognize any update of the Web page and can therefore become inconsistent.

In this chapter we introduce the idea of generating the meta-data model at *server side* and offer the model for download and as a service for querying. We show how WEESA Web applications can be extended to enable the meta-data accumulation from single Web pages to generate and maintain the meta-data model of the Web application. We further discuss scenarios where applications such as software agents can take advantage of both the downloaded meta-data model and the query service of the Semantic Web application.

5.1 META-DATA MODEL OF THE WEB APPLICATION

As described in the previous chapters, the WEESA mapping can be used to semantically annotate Web pages. Looking at the meta-data of a single Web page, however, gives only a limited view on the meta-data available. For querying and reasoning purpose it is better to have the meta-data model of the whole Web application as a *knowledge base* (KB). The same strategy is followed in search engines on the Web. A query is answered with the help of an indexed cache and not by starting to extract information from pages on the Web. This indexed cache is typically generated by *Web crawlers*.

A Web crawler, also known as Web spider or gatherer, is a program that browses the WWW in an automated manner. Crawlers typically start with a list of Web pages to visit. As it visits the pages, it identifies all links to further Web pages and adds them to the list of pages to visit. The download pages are used for example to build the indexed cache of a search engine or to mirror Web pages.

In the Semantic Web we are not talking about an indexed cache but of a knowledge base. Ontobroker [28], SEAL [60], and SHOE [39] use the KB to enable reasoning and database like queries on top of the information system. CREAMS's OntoMat [36] uses the knowledge base to assist the semantic annotation from documents. When a user enters a literal such as "Gerald Reif" in the OntoMat editor the KB is queried to see if a resource identifier (i.e. URI) for this person exists. All the systems mentioned above use a Web crawler to build their KB. The Web crawler periodically collects information from semantically annotated documents and adds the meta-data to the KB.

Using a crawler allows the Web application and the application generating the KB to be loosely coupled and can be distributed over the Internet. This loose coupling, however, makes it difficult to track the deletion or update of a Web page. The KB can therefore become inconsistent with the actual information available on the Web server.

To overcome this inconsistency problem we suggest the generation of the KB at the server side and to offer the KB as a service for querying. In addition a snapshot of the KB can be offered in compressed format for download by clients. In this case the KB of a Web application can be downloaded as a single stream rather than requiring separate requests for every single Web page. This follows the idea proposed by the Harvester search engine architecture [10] where the index of Web applications is created locally at server side and is then offered to brokers for download.

The download of the KB, however, again leads to the inconsistency problem discussed above. But still, if the downloaded KB and the query service to the KB offered by the Semantic Web application are used in symbiosis, the application can benefit from it. Static information that does not change frequently can be retrieved from the locale copy of the KB without online access to the Web application. Dynamic information that changes frequently can be retrieved from the query service on demand.

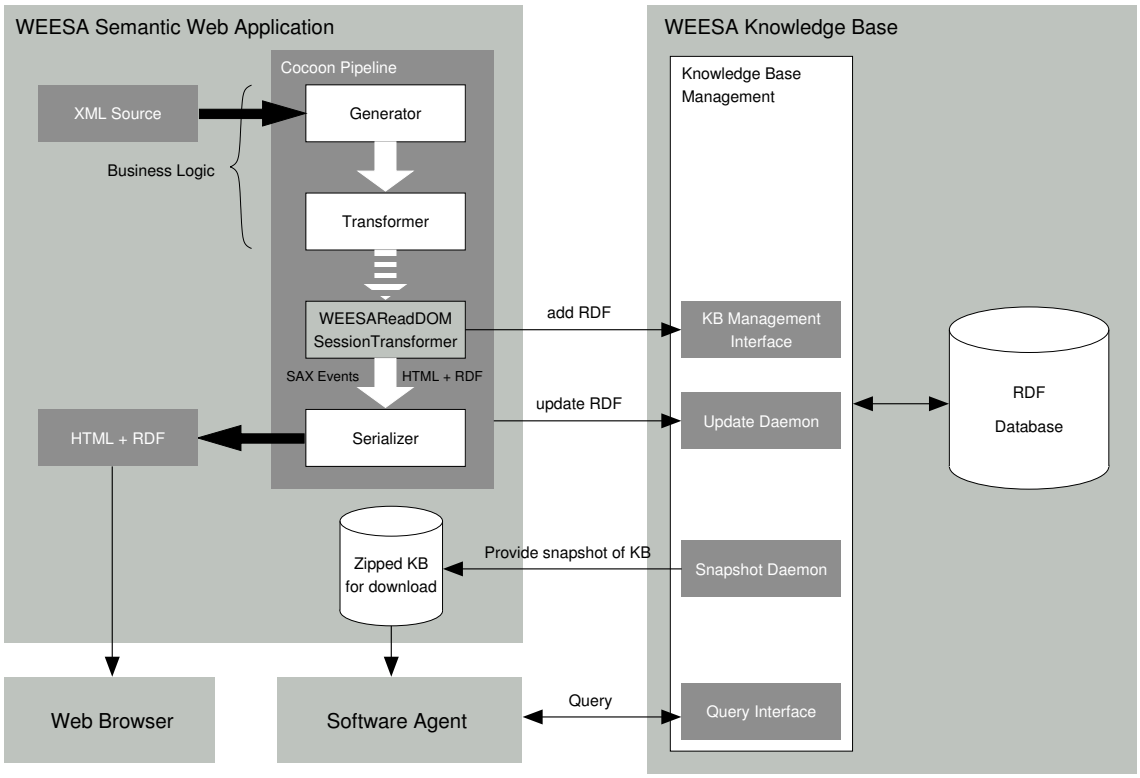

Figure 5.1: Architecture of the WEESA knowledge base.

5.2 Accumulating Meta-Data with WEESA

So far we have discussed the use of WEESA to develop Semantic Web applications within Cocoon. In this section we show how this infrastructure can be used to accumulate the meta-data description of single Web pages to obtain the meta-data model of the whole Web application. We introduce the architecture of the WEESA meta-data accumulation, discuss the tasks of the KB management, and show the important role of RDF reification when maintaining the KB.

5.2.1 WEESA Knowledge Base Architecture

The principle architecture is based on the idea that every time an RDF description is generated by the `WEESAReadDOMSession` transformer the description is also written to the KB of the Web application. The architecture is shown in Figure 5.1.

When a Web browser requests a Web page from a Cocoon Web application the corresponding pipeline is processed. The extended `WEESAReadDOMSession` transformer not only adds the generated RDF meta-data to the HTML page, but writes it via the *KB Management Interface* to the KB. The *Knowledge Base Management* is then responsible for adding the meta-data to the

RDF database. This way, each request to a Web page triggers an update of the RDF statements in the KB. The KB is then offered via the *Query Interface* for querying by software agents.

To keep the application scalable, the Cocoon caching mechanism can be used. When the XML document that is read by the generator component has not been changed since the last request, the subsequent requests are served by the Cocoon cache. This way the `WEESAReadDOMSession` transformer is not processed and the meta-data is not written to the WEESA KB.

At regular intervals the *Snapshot Daemon* takes an image of the KB in RDF/XML format [26], compresses the image, and offers it for download via the Cocoon Web application. The filename of the compressed snapshot is well defined as `weesa_kb.rdf.gz`. The KB of a Web application on the Server with the name `http://www.example.com` can therefore be downloaded using the URL `http://www.example.com/weesa_kb.rdf.gz`. Since this well defined schema to locate the download-able KB an interested application can simply check whether a Web application offers its KB for download. Using this well defined name follows the same principles of the Robots Exclusion Protocol [86] and its `robots.txt` file. This file is used by Web applications to control the behavior of Web crawlers from search engines. Details on the download-able snapshot of the KB are discussed in Section 5.2.3.

5.2.2 Maintaining the Knowledge Base

To build the KB of a Semantic Web application one possibility is to directly access the background database of the Web application to process the data for the meta-data model. This, however, results in the reimplementation of the business logic. Therefore, in our approach we reuse the existing infrastructure of the Cocoon Web application and use the extended `WEESAReadDOMSession` transformer to fill the KB with data. Reusing the existing infrastructure of the Semantic Web application has the advantage that the business logic, implemented by the Cocoon pipeline, is also used to process the data for the meta-data model. This loose coupling between the Web application and the KB, however, leads to problems when maintaining the meta-data model and keeping it up-to-date. In the following we discuss these problems and our proposed solution in detail.

Populating the KB: With the proposed procedure the KB is incrementally filled. Each time a Web page is requested its meta-data is written to the KB. This means, we initially start with an empty KB. To generate the KB of the whole Web application, each available Web page has to be requested. We use the Cocoon command line interface for this purpose. A built-in Cocoon crawler follows every internal link, requests the Web page, and therefore causes the meta-data generation. This is used to initially fill the KB.

Page Deletion Handling: When a Web page has been deleted, the corresponding RDF statements have to be removed from the KB. Therefore, the KB management has to be informed, when a requested Web page cannot be found. In this case the Web server typically sends a 404 “not found” page back to the client. In the Cocoon pipeline definition we can

catch the "not found" event and inform the KB management about the URL of the page that was not found. The KB management can then cause the deletion of all RDF statements that originated from this Web page. To do so, the KB management has to be able to identify the RDF statements that were generated for a given URL. This means we have to store the origin information of each RDF statement in the KB. This is done using RDF reification.

Content Change Handling: When an editor changes the content of a Web page, they typically browses to the page and check for the correct presentation of the changes. In this case, the RDF meta-data is generated within the Cocoon pipeline and an update in the KB is issued. The update first removes all RDF statements from the KB that originated from the Web page with the given URL and adds the newly generated meta-data.

However, if the content of the background database of the Web application is changed and the dynamic Web page that is based on the database is not queried, no update to the KB is issued and the KB becomes inconsistent with the content of the Web application. To avoid this, we add the last modification time and the update interval to the RDF statements for each Web page. This enables the Update Daemon in the KB management to periodically check for outdated RDF statements and to issue an update with the data from the Web application.

Figure 5.2 shows a fragment of the modified WEESA mapping definition to define the update interval. The `updateEvery` attribute in line 4 informs the update daemon that RDF statements originated from this mapping definitions have to be updated at least every 120 minutes. For static pages the `updateEvery` attribute can be omitted. In this case, the update daemon does not check for updates for this kind of Web page.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <mapping xmlns="http://www.infosys.tuwien.ac.at/WEESA#"
3           writeDB="false"
4           updateEvery="120">
5    <resources>
6      <!-- here goes the resources definitions-->
7    </resources>
8    <triples>
9      <!-- here goes the triples definitions-->
10   </triples>
11 </mapping>
```

Figure 5.2: WEESA mapping definition including the additional attributes to maintain the knowledge base.

Filtered Meta-model building: It is not necessary to add the meta-model of each Web page to the KB of the Web application. For example, the meta-data of the personal shopping cart page should not be added. But adding meta-data to such a page is still useful. Therefore an attribute is added to the WEESA mapping definition to instruct the WEESA enabled transformer not to write the meta-data to the KB for these types of pages.

The `writeDB="false"` attribute in line 3 of Figure 5.2 instructs the `WEESAReadDOMSession` transformer not to write the RDF meta-data to the WEESA KB. If the attribute is not given, the default behavior is that the RDF meta-data is written to the KB.

Providing access to up-to-date information: When a snapshot of the KB is downloaded by a client and queried locally, the local KB and the actual data of the Web application may become inconsistent. For some types of information this is not a problem since it is rather static and does not change often. The artist and the track titles on a CD are examples of information that does not change often. However, other types of information such as the price of a CD might change frequently and it may be necessary to have access to latest data from the Web application. Therefore we use reification to store the URL to the RDF statements of the Web page. This way the client is able to request the actual RDF statements on demand.

This overview summarized the tasks of the KB management component. Details on the actual realization can be found in the implementation section of this chapter.

5.2.3 RDF REIFICATION

As mentioned in the previous section, RDF reification has an important role when maintaining the meta-data in the WEESA KB. RDF reification can be used to store information about RDF statements. The KB management component uses reification to store information about the origin, the last update, and the update interval for each RDF statement in the KB. To do so an ontology is needed. Table 5.1 lists the attributes in the WEESA KB ontology that are used to maintain the WEESA KB. The namespace prefix for the WEESA KB ontology is `http://www.infosys.tuwien.ac.at/weesa/kb#`. The OWL syntax of the WEESA KB ontology can be found in Appendix C.

When using RDF reification for each RDF statement an anonymous resource is generated. The subject, predicate, and object of the RDF statement that should be reified are represented as `rdf:subject`, `rdf:predicate`, and `rdf:object` properties of this anonymous resource. In addition, the property `rdf:type` with the object `rdf:statement` indicates that this anonymous resource is used to represent an RDF statement. A reified RDF statement is shown in Figure 5.3 at the bottom left of the RDF graph with the light gray background. The other part of the graph in the figure uses the WEESA ontology to specify information that is needed to maintain the WEESA KB.

The use of RDF reification increases the number of RDF statements in the KB significantly. To represent the reification of the simple RDF statement that the artist with the artist ID `http://example.com/artist#4321` has the `hasArtistName` "`Alanis Morissette`" four RDF statements are needed. The necessary statements are shown in the light gray part at the bottom left of Figure 5.3. In addition we provide six more statements to represent the information needed to maintain the KB. This means for each RDF statement from a Web page we have to store ten more RDF statements in the WEESA KB.

Property	Description
`weesa:description`	Description of a reified RDF statement in the WEESA KB. The object is always an anonymous resource that is in turn described by the following properties.
`weesa:url`	URL of the Web page that caused the generation of the RDF statement.
`weesa:rdf`	URL to the Cocoon pipeline that generates the up-to-date RDF meta-data description of the Web page.
`weesa:lastUpdate`	Time and date of the last update of the RDF statement in `xsi:dateTime` format. (e.g. `2005-05-21T13:20:00.000-01:00`)
`weesa:updateEvery`	Update interval of the RDF statement in minutes. (optional)
`weesa:validUntil`	Time when the Update Daemon has to update the RDF statement. The time is equivalent to the sum of the `lastUpdate` time and the `updateEvery` update interval. The value is the time in milliseconds since midnight, January 1, 1970 UTC. The value is identical with the one obtained from the Java method `System.currentTimeMillis()` (optional)

Table 5.1: WEESA ontology to maintain the RDF statements in the knowledge base. The namespace `weesa` has to be substituted with: `http://www.infosys.tuwien.ac.at/weesa/kb#`

Not all applications that do download a snapshot of the KB need all the information provided by the reified statements in the WEESA KB. Therefore the Snapshot Daemon provides two snapshots of the KB for download:

1. A full snapshot of the WEESA KB including all reifications. A software agent that downloads this KB is able to locally check the time of the last update of a statement and can query the URL of the Web page the RDF statement originated from.

   The predefined name to download this snapshot is: `weesa_kb.rdf.gz` (as discussed in Section 5.2.1)

2. A snapshot of the WEESA KB that contains only the RDF statements accumulated from the Web pages of the Web application without the reified information added by the KB Management. But still, a software agent can use the Query Interface of the KB Management to obtain detailed information on the RDF statements.

   The predefined name to download this snapshot is: `weesa_kb_not_reified.rdf.gz`

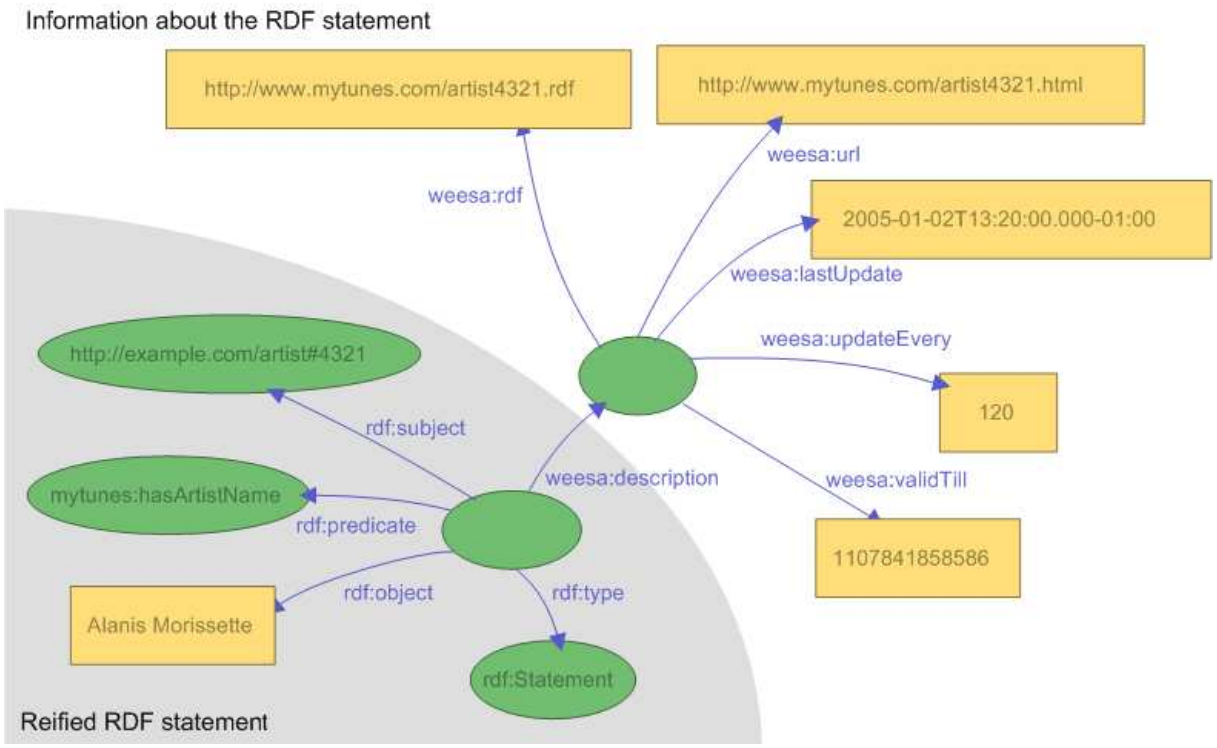

Figure 5.3: Reified RDF statement in the WEESA KB with the information needed to maintain the knowledge base.

5.3 WEESA KB Implementation

In this section we introduce the WEESA KB prototype implementation in detail. We discuss the technologies the prototype is built on, introduce the main Java classes, and explain the configuration of the WEESA KB.

A central component of the WEESA KB is the RDF database that is used to store and maintain the accumulated RDF statements. For the prototype implementation we chose to use the Sesame RDF database [12, 93]. Sesame is an open source RDF database (RDF DB) with support for RDF Schema inferencing and querying. To query the database Sesame supports the SeRQL [92], RDQL [80], and RQL [50] query language. For the WEESA KB we use the SeRQL query language.

As discussed in Section 5.2.1 the KB Management offers services to work on the WEESA KB. The KB Management Interface is used to add RDF statements to the KB and the Query Interface is used to query the WEESA KB. These services are implemented as XML-RPC services. XML-RPC [101] is a protocol that uses XML to encode remote procedure calls (RPC) and uses HTTP as its transport protocol. Implementations of the XML-RPC protocol exists for many platforms and programming languages such as perl, C/C++, Java, .Net, and Python. For the WEESA KB prototype we chose to use the Java Apache XML-RPC implementation [100].

Figure 5.4 shows the functional units of the WEESA KB architecture and the classes these units consist of. The main class is the `WeesaKB` class. This class is responsible for reading the configuration file, and to start and stop the WEESA KB. It instantiates and starts the XML-RPC Server that hosts the KB Management and Query service. These services are represented in the figure by the `KBManagementInterface` and the `QueryInterface`. The `WeesaKB` class further starts the Update and Snapshot Daemon. The `KBmanager` class and the `Reificator` helper class are responsible to write and query the RDF DB. A detailed discussion of the classes in Figure 5.4 is given is the subsequent sections.

The prototype implementation of the WEESA KB is based on a Semantic Web application that embeds the RDF meta-data description in the `<head>` of the Web page. Therefore the HTML+RDF pipeline that is based on the `WEESAReadDOMSession` transformer (see Section 4.2.2) is used in the Cocoon Web application. When the Update Daemon issues the update of the RDF description of a Web page, however, only the RDF description is needed and the HTML page does not have to be generated. Hence, we use the RDF/XML pipeline that is based on the `WEESA` transformer in addition (see Section 4.2.3). In order to select the correct type of pipeline we use the following naming convention: The path in a URL of an HTML Web page has the suffix "`.html`"; the path in the URL of the RDF description of a Web page has the suffix "`.rdf`".

The configuration file is used to specify the filename of the RDF DB and to set the port number the XML-RPC server listens to. It further defines the time intervals the Update and the Snapshot Daemon should run and sets the directory where the Snapshot Daemon should write the compressed images of the KB that they can be downloaded via the Web application. A sample configuration file and a detailed description of each parameter is shown in Figure 5.5.

To start the WEESA KB, the `main` method of the `WeesaKB` class takes the location of the configuration file as command line parameter. The command to start the WEESA KB looks as follows:

```
java WeesaKB wessaKB.config
```

We also provide a class to gracefully shutdown the WEESA KB. In the `main` method of the `ShutdownWeesaKB` class an XML-RPC call is sent to the `shutdown` method of the KB Management Interface to shutdown the WEESA KB. The command to shutdown the KB also takes the location of the configuration file as command line parameter to have access to the port the XML-RPC server listens to:

```
java ShutdownWeesaKB weesaKB.config
```

5.3.1 KNOWLEDGE BASE MANAGEMENT

As discussed in Section 5.2.1 the KB Management is responsible for maintaining the meta-data in the WEESA KB. The `KBmanager` class and the `Reificator` helper class implement this

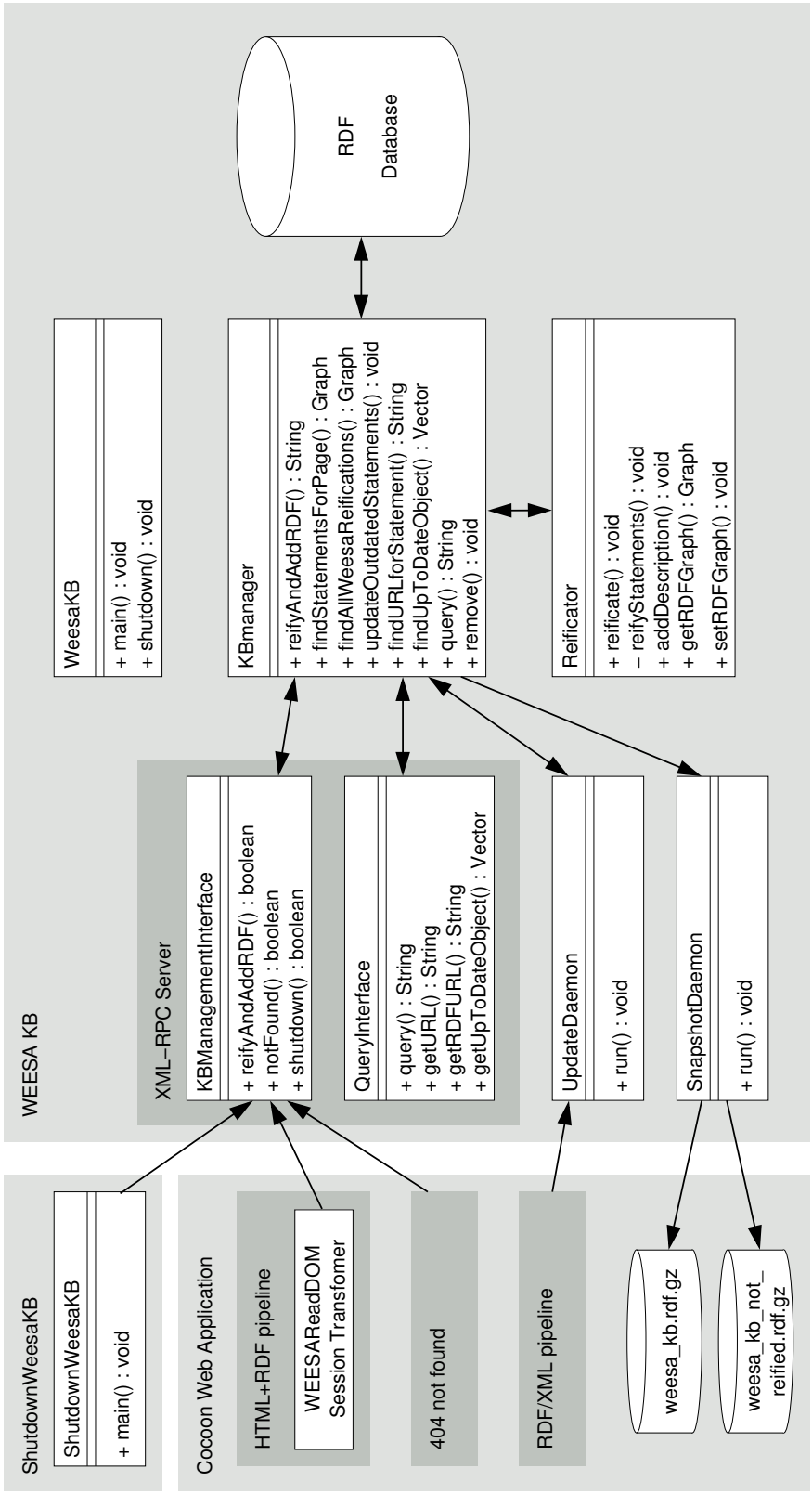

Figure 5.4: Functional units and classes of the WEESA KB.

```
# Filename of the WEESA knowledge base file
database_file=sesame.db

# Service port of the XML-RPC service
# Used to offer the KB Management and the Query service of the WEESA KB
xml_rpc_service_port=9999

# Time interval for the Update Daemon in minutes
# If the interval is set to 0 the Update Daemon is not started
update_daemon_interval=60

# Time interval for the Snapshot Daemon in minutes
# If the interval is set to 0 the Snapshot Daemon is not started
snapshot_daemon_interval=60

# Directory to put the snapshots of the WEESA KB
# Caution, the directory has to be readable by the Web server
snapshot_directory=/opt/htdocs/snapshots
```

Figure 5.5: Sample configuration file of the WEESA KB.

responsibility. The `KBmanager` class is the only class that directly accesses the Sesame RDF DB. It provides methods to the other classes to add, remove and query the DB. If a user decides to use a different RDF DB, such as the RDF Forth Suite [2], only this class has to be adapted.

The `reifyAndAddRDF` method is used to add the RDF description of a Web page to the WEESA KB. The parameters of the method are the RDF/XML description of the Web page and the information that should be added via reflection, such as the URL to the RDF description and the Web page, and the update interval for the Update Daemon. The `Reificator` helper class is initialized with this information and reifies all RDF statements. Before the reified statements are added to the RDF DB, the DB is queried if it already contains statements that originated from the Web page. If so, these statements are replaced by the up-to-date description of the Web page. The `Reificator` class further provides constants representing the URIs of the properties defined in the WEESA KB ontology as defined in Table 5.1.

Most of the other methods of the `KBmanager` class are used to query the RDF DB. For example the `findURLforStatement` method is used to find the URL of the Web page the given RDF statement originated from. The SeRQL query used in this method is shown in Figure 5.6. The `query` method can be used to issue a user defined query on the RDF DB. The result is returned in RDF/XML format. The `findAllReifications` method returns an RDF graph that contains all reifications that were added by the `Reificator` to be able to maintain the WEESA KB. This graph can be used to remove all added reifications from the WEESA KB.

5.3.2 Query Interface

The Query Interface is an XML-RPC service that allows the outside world to query the WEESA KB. The interface provides methods with predefined queries for typical use cases and a method for user defined SeRQL [92] queries. Figure 5.7 shows the methods provided by the interface. In the following we discuss the methods in more detail:

```
SELECT
  url
FROM
  {_Statement} rdf:type {rdf:Statement},
  {_Statement} rdf:subject {subject},
  {_Statement} rdf:predicate {predicate},
  {_Statement} rdf:object {object},
  {_Statement} weesa:description {descriptionId},
  {descriptionId} weesa:url {url}
WHERE
  subject = <subject-to-search-for> AND
  predicate = <predicate-to-search-for> AND
  object = object-to-search-for
NAMESPACE
  weesa http://www.infosys.tuwien.ac.at/weesa/kb#
```

Figure 5.6: Sample SeRQL query that returns the URL of the Web page the RDF statement originated from. The queried statement has the subject `subject-to-search-for`, the predicate `predicate-to-search-for`, and literal object `object-to-search-for`.

getURL() The `getURL` method takes three strings as parameters representing the subject, predicate, and object of an RDF statement. The fourth `objectIsResource` boolean parameter indicates whether the object is a resource (`true`) or the object is a literal (`false`). The return value of the method is the URL to the Web page from where the given RDF statement originated from.

This query can be used by a software agent that downloaded the not reified WEESA KB to get the URL of the Web page of a given RDF statement. This URL can then be used to display the Web page from where the statement originated.

getRDFURL() The `getRDFURL` method takes three strings as parameters representing the subject, predicate, and object of an RDF statement. The fourth `objectIsResource` boolean parameter indicates whether the object is a resource (`true`) or the object is a literal (`false`). The return value of the method is the URL of the RDF description of the Web page from where the given RDF statement originated.

This query can be used by software agents that need the up-to-date meta-data description of a Web page for a given RDF statement.

getUpToDateOject() The `getUpToDateObject` method takes two strings as parameters representing the subject and predicate of an RDF statement. The method queries the WEESA KB for the up-to-date object value for the given subject/predicate. Since it is possible for an RDF graph to contain more than one statement for a given subject/predicate, the return value is the `Vector` of all corresponding object values.

When a software agent works on the downloaded snapshot of the WEESA KB, the downloaded KB and the actual data of the Web application may become inconsistent. For some types of information this is not a problem since it is rather static and does not change often. The artist and the track titles on a CD are examples of information the does not change of-

```
/**
 * Interface of the WEESA KB Query Service
 */
public interface QueryInterface {
    /**
     * Query for the URL of the Web page the RDF statement originated from.
     *
     * @param subject of the statement
     * @param predicate of the statement
     * @param object of the statement
     * @param objectIsResource true if the object is a resource, false if
                               a literal
     * @return URL of the Web page
     */
    public String getURL(String subject, String predicate, String object,
                         boolean objectIsResource);
    /**
     * Query for the URL of the RDF description of the Web page.
     *
     * @param subject of the statement
     * @param predicate of the statement
     * @param object of the statement
     * @param objectIsResource true if the object is a resource, false if
                               a literal
     * @return URL of the RDF description
     */
    public String getRDFURL(String subject, String predicate, String object,
                            boolean objectIsResource);
    /**
     * Queries the up-to-date value of the object in statements with the
     * given subjec/predicate.
     *
     * @param subject of the statement
     * @param predicate of the statement
     * @return Vector of all up-to-date objects in the WEESA KB
     */
    public Vector getUpToDateObject(String subject, String predicate);
    /**
     * User defined SeRQL CONSTRAINT query on the WEESA KB
     *
     * @param query performed on the WEESA KB
     * @return query result in N triples format
     */
    public String query(String query);
}
```

Figure 5.7: Interface of the Query Service to the WEESA KB.

ten. Other types of information such as the price of a CD might change frequently and the `getUpToDateObject` method can be used to obtain the up-to-date value.

query() The `query` method takes a SeRQL `CONSTRUCT` query [92] as parameter and returns the query result as a String in RDF/XML format. This method can be used for any user defined query on the WEESA KB.

To access the WEESA KB Query Interface, a software agent has to implement an XML-RPC client. Figure 5.8 shows the sample Java code of an XML-RPC client using the Apache XML-RPC package [100]. In this example the XML-RPC server runs on the host `http://example.com` port `9999`. The parameters for the XML-RPC method call are put in a `Vector` following the sequence of the parameters in the signature of the method definition in the interface. The actual XML-RPC method call is done using the `XmlRpcClient.execute` method. The first parameter is the method name as defined in the interface and the second parameter the parameter-vector. The return value has to be casted to the type defined in the interface definition. The sample SeRQL query in the figure returns all RDF statements in the WEESA KB.

```
XmlRpcClient xmlrpc = new XmlRpcClient("http://example.com:9999");
Vector params = new Vector ();
params.add("CONSTRUCT\r␣*\r␣FROM\r␣{subject}␣predicate␣{object}");
String result = (String) xmlrpc.execute("query", params);
```

Figure 5.8: Java code sample of an Apache XML-RPC client.

5.3.3 KB MANAGEMENT INTERFACE

The KB Management Interface is an XML-RPC service that is internally used to maintain the WEESA KB. Figure 5.9 shows the methods provided by the interface. In the following we discuss the methods in more detail:

reifyAndAddRDF() The `reifyAndAddRDF` method is used by the Cocoon Web application to write the meta-data description of a Web page to the WEESA KB. The method is called by the `WEESAReadDOMSession` transformer in the HTML+RDF pipeline. The parameters of the `reifyAndAddRDF` method are the meta-data description in RDF/XML format and the information that is needed to reify the RDF statements (see Section 5.2.3). The method then reifies all RDF statements from the description and adds them to the WEESA KB.

The modifications that are needed to enable the `WEESAReadDOMSession` transformer to write the meta-data to the WEESA KB are discussed in Section 5.3.4.

notFound() When a not-existing Web page has been requested, it is not found by the Cocoon Web application and the WEESA KB has to be informed that the page with this URL

```
public interface KBManagementInterface {
    /**
     * Used to reify and add RDF meta-data descriptions to the WEESA KB.
     *
     * @param rdfxml String of RDF statements in RDF/XML format to be reified and
     *               added to the WEESA KB
     * @param url of the Web page the RDF statements originated from
     * @param rdfurl URL to the RDF description of the Web page
     * @param lastUpdateDate time of last update
     * @param updateEvery interval the RDF description ot this Web page should be
     *                    updated in the WEESA KB
     * @return true if everything was ok
     */
    public boolean reifyAndAddRDF(String rdfxml, String url, String rdfurl,
                                  Date lastUpdateDate, String updateEvery);
    /**
     * If a Web page no longer exists this method call removes all RDF
     * statements from the WEESA KB that originated from this Web page.
     *
     * @param url of the Web page that does no longer exist.
     * @return true if everything was ok
     */
    public boolean notFound(String url);
    /**
     * Shutdown the WEESA KB instance.
     *
     * @return true if everything was ok
     */
    public boolean shutdown();
}
```

Figure 5.9: Interface of the KB Management Service to the WEESA KB.

does not exist. This is done with the `notFound` method that takes the URL that caused the "404 not found" error as parameter. The method then causes the deletion of all RDF statements in the WEESA KB that originated from the Web page with the given URL.

The handling of the 404 not found error in the Cocoon pipeline definition is discussed in Section 5.3.5.

shutdown() The `shutdown` method is used in the `ShutdownWeesaKB` class to shutdown the WEESA KB from the command line.

5.3.4 WEESA READ DOM SESSION TRANSFORMER

To accumulate RDF meta-data descriptions from single Web pages in a WEESA Web application the meta-data has not only to be added to the HTML page when a Web page is queried, but also written to the WEESA KB. Therefore we extended the `WEESAReadDOMSession` transformer to write the generated meta-data via the `reifyAndAddRDF` method of the `KBManagement-Interface` to the WEESA KB. To be able to access the WEESA KB XML-RPC service, the transformer has to know the host and port of the service. This information is defined with two additional parameters in the `WEESAReadDOMSession` transformer definition in the

Cocoon `sitemap.xmap` configuration file. Figure 5.10 shows the definition of the transformer. The `weesa-kb-host` parameter gives the hostname the WEESA KB runs on and the `weesa-kb-port` parameter the port the XML-RPC service listens to.

```
<map:transform type="WEESAReadDOMSession">
  <map:parameter name="dom-name" value="rdf"/>
  <map:parameter name="trigger-element" value="head"/>
  <map:parameter name="position" value="in"/>
  <map:parameter name="weesa-mapping-definition"
                 value="mapping/albumMapping.xml"/>
  <map:parameter name="weesa-kb-host" value="http://example.com"/>
  <map:parameter name="weesa-kb-port" value="9999"/>
</map:transform>
```

Figure 5.10: Configuration of the `WEESAReadDOMSession` transformer to write to the WEESA KB.

As discussed in Section 5.2.2 it is not necessary for the meta-data of every Web page to be written to the WEESA KB. For user-specific pages, such as a shopping cart, the meta-data is not of public interest. Therefore, the modified `WEESAReadDOMSession` has to check the `writeDB` attribute in the WEESA mapping definition (see Figure 5.2) to decide whether the meta-data has to be added to the KB.

5.3.5 NOT FOUND (404) DETECTION

In the event that the Web application encounters a "404 not found" error, the WEESA KB has to be informed that the Web page with this URL does not exist. Therefore, we have to add an additional step in the pipeline responsible for the error handling in the Cocoon `sitemap.xmap` configuration file. Figure 5.11 shows the pipeline for the error handling. The `<map:act>` element in lines 4-7 defines the call of the `weesa-not-found` action. In this action the `notFound` method of the `KBManagementInterface` is called. To be able to access the XML-RPC service the `weesa-kb-host` parameter gives the hostname the WEESA KB runs on and the `weesa-kb-port` parameter gives the port of the XML-RPC service.

5.3.6 UPDATE DAEMON

In Section 5.2.2 we discussed the need for a "Content Change Handling" when maintaining the WEESA KB. When the information is changed in the background database of the Web application, the WEESA KB and the data available in the Web application can become inconsistent. Therefore the Update Daemon has to periodically check for RDF statements with an outdated update interval.

The `UpdateDaemon` class is implemented as a Java `TimerTask` that runs in regular intervals and searches for outdated RDF statements in the WEESA KB. The time interval in minutes is set by the `update_daemon_interval` parameter in the configuration file (see Figure 5.5).

```
<map:handle-errors>
  <map:select type="exception">
    <map:when test="not-found">
      <map:act type="weesa-not-found">
        <map:parameter name="weesa-kb-host" value="http://example.com"/>
        <map:parameter name="weesa-kb-port" value="9999"/>
      </map:act>
      <map:generate type="notifying"/>
      <map:transform src="stylesheets/system/error2html.xslt">
        <map:parameter name="contextPath" value="{request:contextPath}"/>
        <map:parameter name="pageTitle" value="Resource not found"/>
      </map:transform>
      <map:serialize status-code="404"/>
    </map:when>

    <map:otherwise>
      <map:generate type="notifying"/>
      <map:transform src="stylesheets/system/error2html.xslt">
        <map:parameter name="contextPath" value="{request:contextPath}"/>
      </map:transform>
      <map:serialize status-code="500"/>
    </map:otherwise>
  </map:select>
</map:handle-errors>
```

Figure 5.11: Handling of the 404 not found error in the Cocoon pipeline.

When the Update Daemon finds an outdated statement, it gets the URL of the RDF description of the Web page from the reified data using the `weesa:rdf` property and replaces the outdated description with the up-to-date meta-data description received from the RDF pipeline of the Cocoon Web page. For this purpose the Update Daemon uses the `updateOutdatedStatements` method of the `KBmanager` class.

5.3.7 SNAPSHOT DAEMON

The `SnapshotDaemon` is implemented as a Java `TimerTask` that runs periodically and takes a snapshot of the current status of the WEESA KB. The time interval in minutes is set by the `snapshot_daemon_interval` parameter in the configuration file (see Figure 5.5). As introduced in Section 5.2.3 two snapshots of the KB are taken: one including all reifications that were added to maintain the KB using the WEESA KB ontology (`weesa_kb.rdf.gz`) and one without these reifications (`weesa_kb_not_reified.rdf.gz`).

Both snapshots are in RDF/XML format and compressed with the gzip (GNU zip) compression utility [35]. The Snapshot Daemon stores the snapshots in the directory set with the `snapshot_directory` parameter in the WEESA KB configuration file (see Figure 5.5). The access rights of this directory and the two snapshot files have to be set so that the Web application is able to read them.

5.4 RESUME

In this chapter we argued that for querying and reasoning purpose the full meta-data model the of whole Web application is needed. Therefore the WEESA knowledge base (KB) accumulates the meta-data descriptions for the individual Web pages and maintains them in an RDF database. Opposite to Web crawlers proposed in literature, our approach generates the KB at server side. This has the advantage, that the KB can be kept consistent with the content of the Web application.

To fill the WEESA KB with data, the pipeline infrastructure of the Cocoon-based WEESA Semantic Web application is used. We extended the `WEESAReadDOMSession` transformer to write the generated RDF meta-data to the WEESA KB. This enables the reuse of the existing infrastructure such as the business logic, results, however, in some effort to keep the KB up-to-date. The WEESA KB management accomplishes this task. When a Web page is requested that does not exist on the server, the Web application informs the KB management to delete meta-data that originated form this Web page. To trace content changes in the Web application the Update Daemon of the KB management polls in regular intervals the Web application and issues an update of the KB.

The KB management stores the accumulated meta-data in an RDF database. For maintenance purpose, the stored RDF triples are reified with information such as the last modification date, the update interval, the URL to its HTML page, and the URL to the RDF description on the Web application. Reifying all accumulated RDF statements significantly increases the number of RDF statements stored in the RDF DB. Therefore we suggest that a future version of the WEESA KB should be based on the recently proposed RDF *Named Graphs* [13]. A Named Graph is an RDF graph which is assigned a name in form of a URI. In the case of the WEESA KB the RDF meta-data description of a Web page can be seen as a Named Graph with the URL as its name. Using Names Graphs has the advantage that not every individual statement has to be annotated with the origin and last update information, but the whole RDF graph generated for a Web page. This reduces the number of statements in the RDF DB significantly.

The Snapshot Daemon of the KB management periodically takes snapshots of the WEESA KB and offers them for download via the Web application to enable reasoning at the client side. Two snapshots are offered for download: one including all reifications that were added for maintenance purpose, and one without reification.

The WEESA KB management further provides an XML-RPC service to query the WEESA KB. The query interface provides predefined queries for common tasks and the possibility to formulate user defined queries in the SeRQL query language. The WEESA KB query service can be used by software agents to query and reason in the KB of the Semantic Web application. A Future Version of the WEESA KB should also support the SPARQL query language and protocol [75], which is currently a W3C working draft.

CHAPTER 6

THE VIENNA INTERNATIONAL FESTIVAL (VIF) CASE STUDY

> The most likely way for the world to be destroyed, most experts agree,
> is by accident. That's where we come in;
> we're computer professionals.
> We cause accidents.
>
> Anonymous

So far we have introduced the WEESA mapping, the Cocoon components that can be used to build Semantic Web applications, and the infrastructure to build the meta-data model of the Semantic Web application. In this chapter we show how these ideas are used in an industry case study. As a case study we have used the Vienna International Festival (VIF) Web application and reimplemented it as a WEESA Semantic Web application.

The Vienna International Festival[1] (German: Wiener Festwochen) is a major cultural event in Vienna. This annual festival usually lasts six to eight weeks over a period in May and June. The festivities take place in various theater locations and concert halls and consist of operas, plays, concerts, musicals, and exhibitions. The VIF Web application is a database supported application that provides a detailed event description of each hosted event, an online ticket shop, press reviews, and an archive over the events of the last 52 years. For the WEESA case study we use the event descriptions and the ticket shop and semantically annotate the corresponding Web pages. Below we list the Web pages that make up the case study and discuss the meta-data the pages provide:

[1] `http://www.festwochen.at`

VIF Homepage: Entry point to the VIF Web application. It provides general information about the festival and a navigation bar to the features of the Web application. The meta-data consists of the contact information of the festival.

Program overview pages: List of events of the festival. There is one list of all events and several lists for the events of a specific event category (such as concert, performing arts, etc.) The meta-data consists of the events, location, and the category of the event.

Event description: Detailed information about the event. It contains information such as the title, description, location, date, etc. of the event. Figure 6.1 shows the screen-shot of an event description Web page of the festival. The meta-data reflects the event details provided by the Web page.

Ticket shop receipt: Receipt of the bought ticket. It is the final acknowledgement of the shopping process in the online ticket shop containing all the details of the specific event. The meta-data reflects event details of the bought ticket.

When we chose the case study application we decided not to design a new Web application from scratch but to adopt an existing one. This decision has the advantage that we are able to work out the differences between the design of traditional Web applications and Semantic Web applications.

In this chapter we take the VIF case study and discuss several realizations of the Semantic Web application with different requirements. In Section 3.3 we introduced the WEESA mapping and argued that the mapping has to leave the flexibility to change the ontologies used without redesigning the whole Web application. To demonstrate this flexibility in this chapter we first show the implementation of the case study using a self defined VIF ontology and second the realization based on the iCalendar ontology [45, 77, 84]. We further demonstrate the use of both possibilities to associate RDF and HTML introduced in Chapter 4 and discuss example scenarios how the WEESA knowledge base of the VIF case study can be used by software agents. A discussion of the experiences gained in the case study and hints developers should follow when designing WEESA Semantic Web applications can be found in the following chapter.

6.1 Embedding RDF/XML into the HTML Page

In this section we introduce the realization of the VIF case study Web application that embeds the RDF meta-data description in the `<head>` element of the HTML page. We first show the Cocoon pipelines that configure the Web application and second introduce the WEESA mapping definitions used for the meta-data generation.

The VIF Web application was originally implemented using MyXML [52]. MyXML is a Web application framework that is based on separation-of-concerns. For our case study we took the existing database and reimplemented the Web application based on the Cocoon Web application framework. To do so, we followed the steps introduced in Chapter 3 and defined the XML

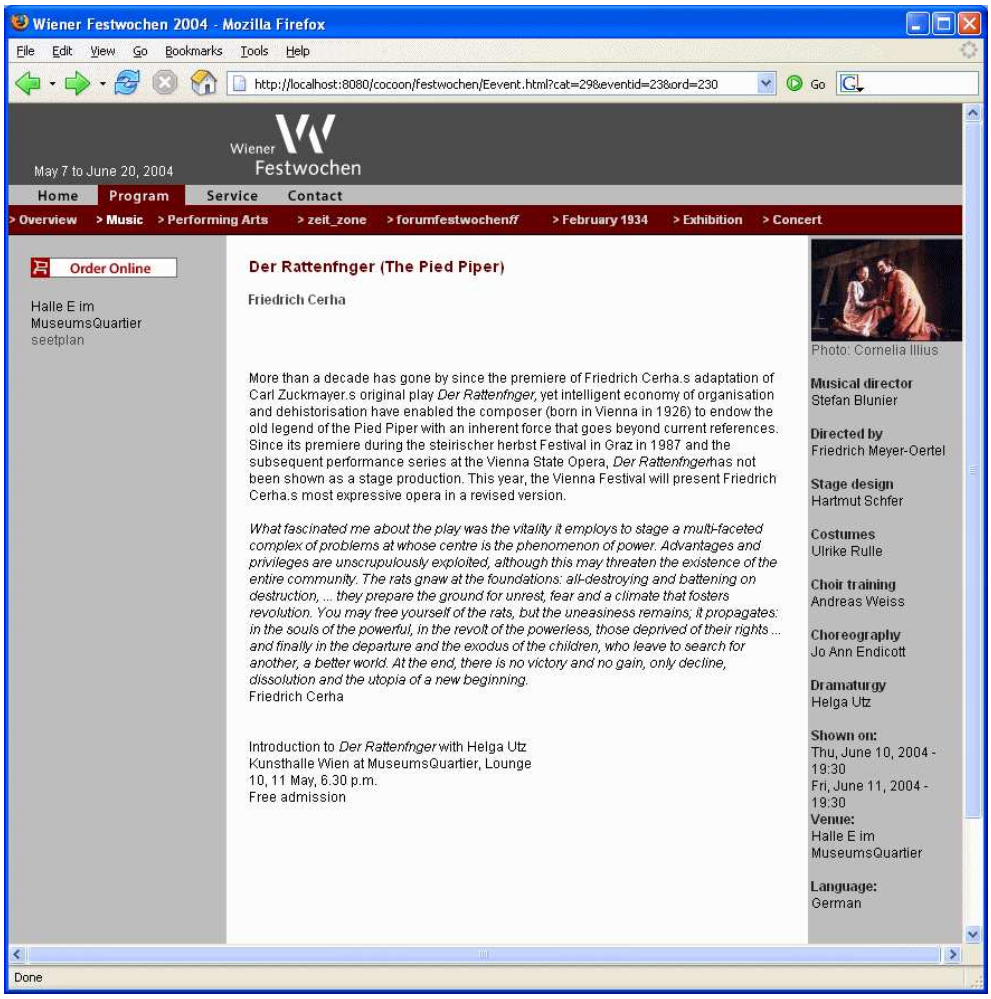

Figure 6.1: Screen-shot of an event description Web page of the VIF Web application.

Schema as contract of the Web application and the XSLT stylesheet to generate the HTML page. The business logic of the application is implemented using *XML server pages* (XSP) [102]. XSP pages are XML files that can contain Java code and database queries that are processed at server side. When a Cocoon `serverpage` generator component reads the XSP page it processes the contained code and replaces it with the computed value. The output of the `serverpage` generator is then an XML document.

In Section 4.2.2 we introduced the `WEESAReadDOMSession` transformer and showed how this transformer is used in a Cocoon pipeline to embed RDF in HTML. This pipeline setup is also used in our case study Web application. The information flow in the pipeline is shown in Figure 4.3. Figure 6.2 depicts the pipeline definition in the Cocoon `sitemap.xmap` configuration file. In the VIF Web application the business logic is processed at the beginning of the pipeline by the `serverpage` generator that is defined in line 9. The output is then a stream of SAX events that represents the XML Schema valid XML document, the XSLT stylesheet and the

```
1  <map:match pattern="Event.html">
2    <!-- create the session -->
3    <map:act type="session">
4      <map:parameter name="action" value="create"/>
5      <map:parameter name="mode" value="immediately"/>
6    </map:act>
7
8    <!-- process the business logic in the xsp file -->
9    <map:generate type="serverpages" src="docs/Event.xsp" />
10
11   <!-- write the XML document to the session -->
12   <map:transform type="writeDOMsession">
13     <map:parameter name="dom-name" value="rdf"/>
14     <map:parameter name="dom-root-element" value="site"/>
15   </map:transform>
16
17   <!-- do the XSLT transformation -->
18   <map:transform type="xslt" src="stylesheets/Event.xsl">
19     <map:parameter name="navi" value="eprog"/>
20     <map:parameter name="id" value="{request-param:cat}"/>
21   </map:transform>
22
23   <!-- configure and start the WEESAReadDOMSession transformer -->
24   <map:transform type="WEESAReadDOMSession">
25     <map:parameter name="dom-name" value="rdf"/>
26     <map:parameter name="trigger-element" value="head"/>
27     <map:parameter name="position" value="in"/>
28     <map:parameter name="weesa-mapping-definition" value="mappings/eventMapping.xml"/>
29     <map:parameter name="weesa-kb-host" value="localhost"/>
30     <map:parameter name="weesa-kb-port" value="9999"/>
31   </map:transform>
32
33   <!-- serialize the output -->
34   <map:serialize type="html" />
35 </map:match>
```

Figure 6.2: Cocoon Pipeline definition for a VIF event Web page in the `sitemap.xmap` configuration file.

WEESA mapping definition were designed for. The following steps in the pipeline have already been explained in Section 4.2.2.

As explained in Chapter 5 the `WEESAReadDOMSession` transformer writes the generated meta-data description in addition via an XML-RPC call to the WEESA KB. To keep the KB up-to-date, the Update Daemon periodically requests for an update of the meta-data description of a Web page. In this case only the RDF/XML representation of the Web page has to be generated. The XSLT transformation for the HTML page does not have to be processed. Therefore we use the pipeline that is based on the `WEESA` transformer as introduced in Section 4.2.3. Again, the business logic at the beginning of the pipeline is processed by the `serverpage` generator.

To enable Cocoon to select the correct pipeline to service an incoming request, each pipeline has a pattern matcher defined at the beginning of the pipeline definition. This pattern matcher matches the path information of the URL and triggers the processing of the pipeline. The matcher for our example pipeline in Figure 6.2 is defined in line 1 and matches the path `"Event.html"`.

As a naming convention in our case study the path in the URL to a pipeline that generates

a HTML+RDF page ends with `".html"` and to a pipeline that generates RDF/XML ends with `".rdf"`. Therefore the URL to the HTML page of the event with the id "23" looks as follows:

```
http://www.festwochen.at/Event.html?eventid=23
```

The corresponding URL to the RDF/XML document is:

```
http://www.festwochen.at/Event.rdf?eventid=23
```

In the following sections we explain the WEESA mapping definitions for the VIF case study and show that the used ontology can be changed by only changing the WEESA mapping definitions.

6.1.1 VIF SEMANTIC WEB APPLICATION USING THE VIF ONTOLOGY

As we did not find an appropriate ontology for cultural festivals on the Web we defined our own VIF ontology to describe the festival details. The ontology provides four main classes: `Festival` to describe the festival itself; `Event` to describe an event; `EventDate` to describe the begin and end date of an event; and `Ticket` to store information about the tickets bought. In addition there is a subclass of the `Event` class for each event category in the festival. The classes and properties of the VIF ontology are shown in Figure 6.3. For reasons of clarity, we do not use the OWL Syntax in the example, but use a simple textual syntax instead. The VIF ontology in OWL syntax can be found in Appendix D.

```
Class: Event                                        Class: EventDate
  -> hasEventName                                     -> beginDate
  -> hasEventDescription                              -> beginTime
  -> hasLocation                                      -> endDate
  -> language                                         -> endTime
  -> eventDate (range: EventDate)
  -> directedBy                                     Class: Festival
  -> ticketShopURL                                    -> hasFestivalName
  -> boughtTicket (range: Ticket)                     -> hasEvent (range: Event)

Class: Concert          (subclass of Event)       Class: Ticket
Class: Exhibition       (subclass of Event)         -> hasPrice
Class: February1934     (subclass of Event)         -> quantity
Class: Music            (subclass of Event)
Class: PerformingArts   (subclass of Event)
Class: Zeit_zone        (subclass of Event)
Class: Forumfestwochen  (subclass of Event)
```

Figure 6.3: VIF ontology for the Vienna International Festival case study Web application.

For a detailed discussion of the WEESA mapping definition we chose the Web page of a VIF event. Following the WEESA approach introduced in Chapter 3, we first define the XML Schema for the Web page. Since the XML Schema is good for defining the structure of XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<site>
  <content>
    <left_frame>
      <tickets href="EShop?id=23"/>
      <location id="22">Hall E at MuseumsQuartier</location>
      <link_new href="images/english/shared/saalplaene/halle_e_Rattenfaenger.gif">
        <desc>seetplan</desc>
      </link_new>
    </left_frame>
    <main>
      <event id="23"/>
      <category id="29"/>
      <title>Der Rattenfaenger (The Pied Piper)</title>
      <description>
        More than a decade has gone by since the premiere of Friedrich Cerha [...]
      </description>
    </main>
    <right_frame>
      <image src="images/english/shared/photos/Rattenfaenger_02_2_small.JPG"></image>
      <desc>Photo: Cornelia Illius</desc>
      <value_pair>
        <sub_title>Directed by</sub_title>
        <description>Friedrich Meyer-Oertel</description>
      </value_pair>
      <value_pair>
        <sub_title>Choir training</sub_title>
        <description>Andreas Weiss</description>
      </value_pair>
      <value_pair>
        <sub_title>Choreography</sub_title>
        <description>Jo Ann Endicott</description>
      </value_pair>
      <value_pair>
        <sub_title>Venue:</sub_title>
        <description>Hall E at MuseumsQuartier</description>
      </value_pair>
      <value_pair>
        <sub_title>Language:</sub_title>
        <description>German</description>
      </value_pair>
      <value_pair>
        <sub_title>Length:</sub_title>
        <description>110 mins, 1 interval</description>
      </value_pair>
      <sub_title>Shown on:</sub_title>
      <dates id="161">
        <edate>Thu, June 10, 2004</edate>
        <date>2004-06-10</date>
        <time>19:30</time>
      </dates>
      <dates id="162">
        <edate>Fri, June 11, 2004</edate>
        <date>2004-06-11</date>
        <time>19:30</time>
      </dates>
    </right_frame>
  </content>
</site>
```

Figure 6.4: XML file of a VIF event Web page.

documents but not good for human readability we show an example instance document that is valid to this schema. The document is shown in Figure 6.4. The XML document splits the Web page into three parts: the `left_frame`, the `main` part, and the `right_frame`. The header containing the logo and the navigation bar is not included in XML document but added by the XSLT stylesheet.

The WEESA mapping definition for a VIF event Web page is shown in Figure 6.5 and 6.6. In Section 3.4.1 we already introduced the available elements/attributes in the mapping definition and their functionality. Therefore, we focus on some specialties in this mapping definition.

In a Semantic Web application, one resource identifier should be used for the same resource on all Web pages throughout the whole Web application. In our example, the same URI should be used as resource identifier for one event. In the VIF Web application, the information about the events is stored in the DB with the event id as key. This DB key is used in the XML document in line 12 as value for the `id` attribute from the `<event>` element. In lines 4-10 of the mapping definition we define the URI that is used as resource identifier for the event we want to make statements about. The URI is made up of the prefix and the value of the `id` attribute. Since we use the same prefix and DB key to generate the resource identifier for the event on all Web pages that make a statement about the event, we can guarantee that the same URI is used for the same event throughout the whole Web application.

In lines 15-25 of the mapping definition we define the triple that defines the class the event is an instance of. In the XML file the event category is defined in line 13 with the `id` attribute of the `<category>` element. The `id="29"` stands for the category `Music` event. For the RDF triple, however, we need the URI of the corresponding category class form the VIF ontology (`http://www.festwochen.at/ontology#Music`) instead of the numerical category id. Therefore we use the `selectEventCategory` Java method that takes the namespace prefix of the VIF ontology and the category id as parameter and returns the URI of the corresponding class. The `resource="true"` attribute of the `<object>` element in line 18 of the mapping definition forces that a resource is generated instead of a literal. For a future version of the WEESA mapping we plan to introduce an if or switch element to cover such cases without using a Java method.

In the right frame of the screen-shot in Figure 6.1 a list of event details is given. These details consist of a heading such as “Directed by” and a value, in this case “Friedrich Meyer-Oertel”. The value pairs can be found in lines 23-46 of the XML document in Figure 6.4. In the mapping definition we use these value pairs to fill the properties describing the event. In lines 36-40 of the mapping definition we use the XPath expression to match the `<sub_title>` element with the `"Directed by"` string to select the corresponding value for the object. The use of value pairs in the design of the XML document has the advantage that content editors have the flexibility to “invent” new headings when writing instance documents. This flexibility, however, has the drawback that the value pairs can only be matched by a string comparison. A different string for the `<sub_title>` element such as `"Direction"` would lead to a mismatch and no triple is generated. As an alternative, if the set of all possible values for the `<sub_title>` element are known at the design time of the XML Schema an element for each possible value can be used (in

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping writeDB="true" updateEvery="120">
  <resources>
    <resource id="event">
      <method>
        <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.addPrefix</name>
        <param const="http://www.festwochen.at/event#" type="java.lang.String"/>
        <param xpath="/site/content/main/event/@id" type="java.lang.String"/>
      </method>
    </resource>
    <resource id="date" anonymous="true" var="date_id" xpath="/site/content/right_frame/dates/@id"/>
    <resource id="festival" const="http://www.festwochen.at"/>
  </resources>
  <triples>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
      <object resource="true">
        <method>
          <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.selectEventCategory</name>
          <param const="http://www.festwochen.at/ontology#" type="java.lang.String"/>
          <param xpath="/site/content/main/category/@id" type="java.lang.String" iter="true"/>
        </method>
      </object>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.festwochen.at/ontology#hasEventName"/>
      <object xpath="/site/content/main/title/text()"/>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.festwochen.at/ontology#hasDescription"/>
      <object xpath="/site/content/main/description/text()"/>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.festwochen.at/ontology#directedBy"/>
      <object xpath="/site/content/right_frame/value_pair[sub_title='Directed by']/description/text()"/>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.festwochen.at/ontology#language"/>
      <object xpath="/site/content/right_frame/value_pair[sub_title='Language:']/description/text()"/>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.festwochen.at/ontology#ticketShopURL"/>
      <object>
        <method>
        <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.addPrefix</name>
        <param const="http://www.festwochen.at/" type="java.lang.String"/>
        <param xpath="/site/content/left_frame/tickets/@href" type="java.lang.String"/>
      </method>
      </object>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.festwochen.at/ontology#hasLocation"/>
      <object xpath="/site/content/description/location/text()"/>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.festwochen.at/ontology#eventDate"/>
      <object ref="date"/>
    </triple>
```

Continued in Figure 6.6.

Figure 6.5: WEESA mapping definition for a VIF event Web page using the VIF ontology. Part 1/2

Figure 6.5 continued:

```
    <triple>
      <subject ref="date"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
      <object const="http://www.festwochen.at/ontology#EventDate" resource="true"/>
    </triple>
    <triple>
      <subject ref="date"/>
      <predicate const="http://www.festwochen.at/ontology#beginTime"/>
      <object>
        <method>
          <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.extractBeginTime</name>
          <param xpath="/site/content/right_frame/dates[@id='$$date_id$$']/time/text()" type="java.lang.String"/>
        </method>
      </object>
    </triple>
    <triple>
      <subject ref="date"/>
      <predicate const="http://www.festwochen.at/ontology#beginDate"/>
      <object xpath="/site/content/right_frame/dates[@id='$$date_id$$']/date/text()"/>
    </triple>
    <triple>
      <subject ref="date"/>
      <predicate const="http://www.festwochen.at/ontology#endTime"/>
      <object>
        <method>
          <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.extractEndTime</name>
          <param xpath="/site/content/right_frame/dates[@id='$$date_id$$']/time/text()" type="java.lang.String"/>
        </method>
      </object>
    </triple>
    <triple>
      <subject ref="date"/>
      <predicate const="http://www.festwochen.at/ontology#endTime"/>
      <object>
        <method>
          <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.calculateEndTime</name>
          <param xpath="/site/content/right_frame/dates[@id='$$date_id$$']/time/text()" type="java.lang.String"/>
          <param xpath="/site/content/right_frame/value_pair[sub_title='Length:']/description/text()" type="java.lang.String"/>
        </method>
      </object>
    </triple>
    <triple>
      <subject ref="date"/>
      <predicate const="http://www.festwochen.at/ontology#endDate"/>
      <object xpath="/site/content/right_frame/dates[@id='$$date_id$$']/date/text()"/>
    </triple>
    <triple>
      <subject ref="festival"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
      <object const="http://www.festwochen.at/ontology#Festival" resource="true"/>
    </triple>
    <triple>
      <subject ref="festival"/>
      <predicate const="http://www.festwochen.at/ontology#hasFesitvalName"/>
      <object const="Wiener␣Festwochen"/>
    </triple>
    <triple>
      <subject ref="festival"/>
      <predicate const="http://www.festwochen.at/ontology#hasEvent"/>
      <object ref="event"/>
    </triple>
  </triples>
</mapping>
```

Figure 6.6: WEESA mapping definition for a VIF event Web page using the VIF ontology. Part 2/2

our example a <directedBy> element). The use of the correct element for the title can then be enforced by validating the XML document.

As we already explained, we revised an existing Web application for our VIF case study. In the Web application some information is maintained in a well defined format since it is used in the business logic of the Web application. Other information, however, is stored as a user defined string. For example, the date of an event is used in the ticket shop and therefore stored in the well defined "yyyy-mm-dd" format (e.g. "2005-06-21"). But the time when a performance starts is only used for information purpose on the Web page. Our experience with the VIF case study showed that content editors used the <time> element to enter strings in a format such as "hh:mm", "hh:mm to hh:mm", or "hh:mm till hh:mm". For some events also the Length: of the event is defined in a string such as "110 mins, 1 interval" (lines 43-46). To fill the begin and end time property in the meta-data description with data, however, the time information is needed in a well defined format.

When redesigning the Web application this new requirement has to be taken into account and the XML Schema has to be designed in a way that the content editors are forced to provide the information in a well defined format. But still, we can use Java methods in the WEESA mapping definition to extract the needed information from the existing information in the XML document. In line 77 of the mapping definition the extractBeginTime method is used to search a string for the first occurrence of the "hh:mm" pattern. The method then returns the extracted time in XML Schema time format (xsi:time) and fills the beginTime property. In line 92 the extractEndTime method is used to extract the second "hh:mm" pattern from the time string and fills the endTime property. If the pattern does not occur a second time, no such triple is generated. The calculateEndTime method defined in line 102 takes two parameters: the unformatted time string and the <description> of the <value_pair> with the <sub_title> "Length:". The method tries to extract the start time from the time string and the duration in minutes from the length description. It adds the two times and returns the end time to fill the endTime property of the ontology.

The RDF graph of the meta-data generated for the VIF event Web page using the XML document in Figure 6.4 and the WEESA mapping definition from Figure 6.5 and 6.6 is shown in Figure 6.7.

Most of the event details found on the event Web page are static and do not change frequently. The link to the online shop, however, is only shown when tickets for this event are available in the online ticket shop. The appearance of the link is controlled by the <tickets> element in the XML document (line 5 of Figure 6.4). If the <tickets> element is missing, the link is not shown and the triple providing the ticketShopURL (defined in lines 46-56 of Figure 6.5) is not generated for the meta-data description. To keep the information if tickets are available up-to-date in the WEESA KB the Update Daemon has to periodically request an update of the RDF description of an event. The update interval in minutes is defined by the updateEvery attribute in line 2 of the mapping definition.

After the detailed discussion of the meta-data generated for VIF event Web pages we give a brief overview of the meta-data generated for the program overview, the ticket shop, and the homepage of the VIF Web application. The program overview Web pages only provide a list

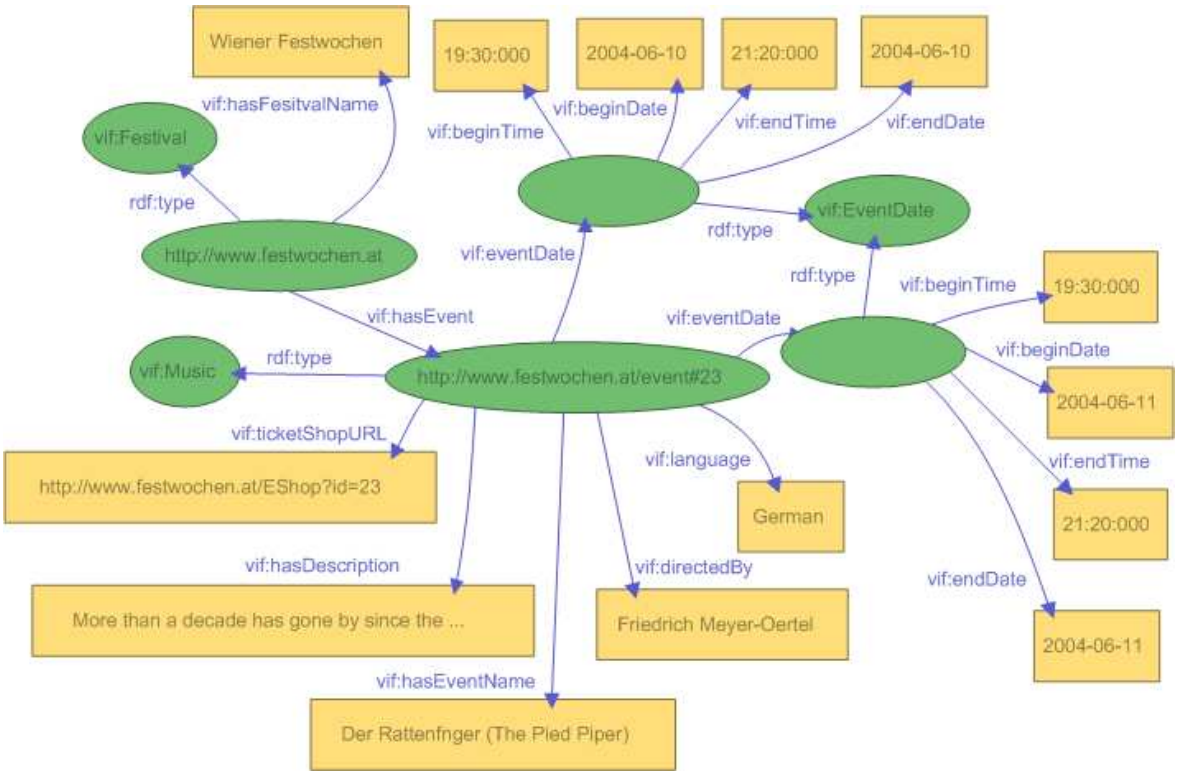

Figure 6.7: RDF graph of the meta-data generated for the event Web page.

of events filtered by the event category. Therefore, the meta-data description contains for each event an instance from the `Event` class or its subclasses described by the event name and the location property. Since this Web page makes statements about events we have to ensure that the same URIs are used as resource identifiers as on the corresponding Web page with the event description. This is done by providing access to the DB key of the event in the XML document, as described above.

The ticket receipt Web page, the user gets as the final acknowledgement of the shopping process in the online ticket shop, contains the event details for the events for which the user bought tickets. The meta-data generated for these events is similar to the one generated for the event Web pages and includes in addition the price and the number of tickets bought for the event. Again, the resource identifier for the event has to be the same as the one on the corresponding event page. Since this information is not of public interest, the meta-data is not added to the WEESA KB. This is indicated by the `writeDB="false"` attribute in the mapping definition.

The VIF homepage itself does not contain much information that can be transformed for the meta-data description. But we use this Web page to provide meta-data about principle information of the VIF festival. This information contains the contact address and the phone number of the festival office. To specify this information we use the vCard ontology [43]. Since this information is static, the corresponding RDF/XML fragment can directly be put into the `<head>`

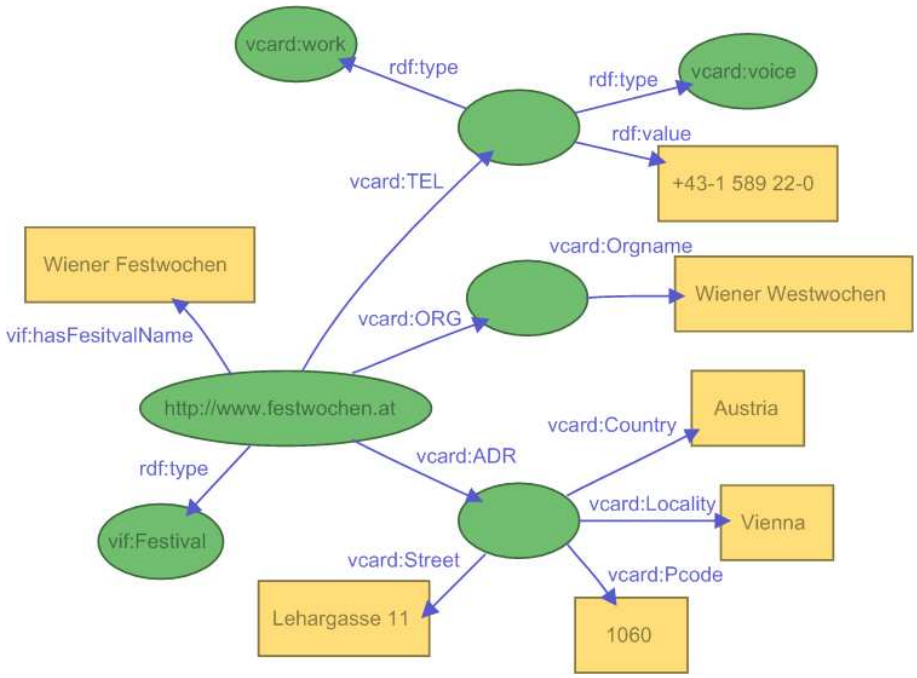

Figure 6.8: RDF graph of the meta-data generated for the VIF homepage.

element of the Web page. In this case study, however, we show that the WEESA mapping can be used to generate static RDF content that is not derived from the content of a Web page. Figure 6.8 shows the RDF graph of the meta-data description for the VIF homepage that is generated using the mapping definition depicted in Figure 6.9. The resource definitions (lines 3-8) do not use any XPath reference to the XML document. Instead, the resources are defined as constants or anonymous. The following triple definitions in Figure 6.9 are like the definitions that we have seen in the examples so far. Since the meta-data generated for the homepage is static, the Update Daemon of the WEESA KB does not have to check for updates. This is indicated with the missing `updateEvery` attribute in line 2 of Figure 6.9.

The use of the meta-data description from the whole Web application in the WEESA KB that is accumulated form the meta-data descriptions of the individual Web pages is discussed in Section 6.2.

6.1.2 VIF Semantic Web Application using the iCalendar Ontology

In this section we take the Semantic Web application developed in the previous section and change the ontology that is used to semantically annotate the Web pages. The goal of this case study implementation is to show that the ontology being used can be changed without modifying the structure of the XML pages or the business logic of the Web application. Only the WEESA

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping writeDB="true">
  <resources>
    <resource id="festival" const="http://www.festwochen.at"/>
    <resource id="address" anonymous="true"/>
    <resource id="organization" anonymous="true"/>
    <resource id="phone" anonymous="true"/>
  </resources>
  <triples>
    <triple>
      <subject ref="festival"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
      <object const="http://www.festwochen.at/ontology#Festival" resource="true"/>
    </triple>
    <triple>
      <subject ref="festival"/>
      <predicate const="http://www.festwochen.at/ontology#hasFesitvalName"/>
      <object const="Wiener␣Festwochen"/>
    </triple>
    <triple>
      <subject ref="festival"/>
      <predicate const="http://www.festwochen.at/ontology#hasFesitvalName"/>
      <object const="Wiener␣Festwochen"/>
    </triple>
    <triple>
      <subject ref="festival"/>
      <predicate const="http://www.w3.org/2001/vcard-rdf/3.0#ORG"/>
      <object ref="organization"/>
    </triple>
    <triple>
      <subject ref="organization"/>
      <predicate const="http://www.w3.org/2001/vcard-rdf/3.0#Orgname"/>
      <object const="Wiener␣Westwochen"/>
    </triple>
    <triple>
      <subject ref="festival"/>
      <predicate const="http://www.w3.org/2001/vcard-rdf/3.0#ADR"/>
      <object ref="address"/>
    </triple>
    <triple>
      <subject ref="address"/>
      <predicate const="http://www.w3.org/2001/vcard-rdf/3.0#Street"/>
      <object const="Lehargasse␣11"/>
    </triple>
    <triple>
      <subject ref="address"/>
      <predicate const="http://www.w3.org/2001/vcard-rdf/3.0#Locality"/>
      <object const="Vienna"/>
    </triple>
    <triple>
      <subject ref="address"/>
      <predicate const="http://www.w3.org/2001/vcard-rdf/3.0#Pcode"/>
      <object const="1060"/>
    </triple>
    <triple>
      <subject ref="address"/>
      <predicate const="http://www.w3.org/2001/vcard-rdf/3.0#Country"/>
      <object const="Austria"/>
    </triple>
    <triple>
      <subject ref="festival"/>
      <predicate const="http://www.w3.org/2001/vcard-rdf/3.0#TEL"/>
      <object ref="phone"/>
    </triple>
    <triple>
      <subject ref="phone"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#value"/>
      <object const="+43-1␣589␣22-0"/>
    </triple>
    <triple>
      <subject ref="phone"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
      <object const="http://www.w3.org/2001/vcard-rdf/3.0#work" resource="true"/>
    </triple>
    <triple>
      <subject ref="phone"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
      <object const="http://www.w3.org/2001/vcard-rdf/3.0#voice" resource="true"/>
    </triple>
  </triples>
</mapping>
```

Figure 6.9: WEESA mapping definition for the VIF homepage.

mapping definitions have to be adopted.

In this case study we use the iCalendar ontology [45, 77] instead of the self-defined VIF ontology. The iCalendar ontology is designed for the domain of calendar events and is based on the iCalendar RFC 2445 [84]. This RFC specifies the text based Internet calendaring and scheduling file format that is used in applications such as Apple's iCal [44] and Mozilla's Sunbird [96].

Since we reuse the Web application developed in the previous section, the Web pages provided and the structure of the XML documents remain the same. Again, we take the event Web page for a detailed discussion of the WEESA mapping definition and the generated meta-data. Figures 6.10 and 6.11 show the mapping definition that is designed for the iCalendar ontology which is applied on the XML document in Figure 6.4 of the previous section.

Opposite to the VIF ontology, in the iCalendar ontology, an instance of the `Vevent` class, representing an event of the festival, can have only one begin and end date. Therefore, for each performance day of an event an instance of the `Vevent` class is generated. The mapping definition is straight forward, except for the use of three new Java methods. The `extractStartDate` method used in line 61 of Figure 6.10 takes the time and date as parameters and returns the time and date in XML Schema date-time format (`xsi:dateTime`). The method uses the same pattern to extract the time from the time string as described in the previous section. The `extractEndDate` method used in line 82 of Figure 6.11 tries to extract the end time from the time string and returns the formatted time and day. The `calculateEndDateTime` method used in line 93 aims to compute the end time from the start time and the `Length:` of the event and returns the formatted time and day.

A part of the generated RDF graph is shown in Figure 6.12. As explained above, an instance of the `Vevent` class is generated for each performance day of the event. The graph in the figure, however, only shows the meta-data of the performance on 2004-06-10.

After the discussion of the meta-data generated for VIF event Web pages, we briefly describe the meta-data generated for the program overview, ticket shop, and homepage of the VIF Web application. The meta-data of the program overview Web pages provides an instance of the `Vevent` class for each event, that is described with the event name, location, and the URL to the Event Web page. The meta-data for the ticket shop receipt looks similar to the one for an event page. It contains an instance of the `Vevent` class for all tickets bought. The homepage, however, does not provide any information that can be represented using the iCalendar ontology and therefore does not offer any meta-data.

The last two sections showed the semantic annotation of the Web pages of the VIF case study. Usage scenarios of the RDF meta-data added are discussed in the future work section of Chapter 9. In the following section we present the WEESA KB generated from the accumulated meta-data from the annotated Web pages.

6.2 Using the WEESA Knowledge Base

The VIF case study implementation also provides the WEESA KB we introduced in Chapter 5 that offers the meta-data of the whole Semantic Web application for download and querying. In

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping>
  <resources>
    <resource id="event">
      <method>
        <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.addPrefix</name>
        <param const="http://www.festwochen.at/event#" type="java.lang.String"/>
        <param xpath="/site/content/right_frame/dates/@id" var="date_id" type="java.lang.String" iter="true"/>
      </method>
    </resource>
    <resource id="start_date" anonymous="true" xpath="/site/content/right_frame/dates[@id='$$date_id$$']/@id"/>
    <resource id="end_date" anonymous="true" xpath="/site/content/right_frame/dates[@id='$$date_id$$']/@id"/>
  </resources>
  <triples>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
      <object const="http://www.w3.org/2002/12/cal/ical#Vevent" resource="true"/>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.w3.org/2002/12/cal/ical#summary"/>
      <object xpath="/site/content/main/title/text()"/>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.w3.org/2002/12/cal/ical#desciption"/>
      <object xpath="/site/content/main/description/text()"/>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.w3.org/2002/12/cal/ical#url"/>
      <object>
        <method>
          <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.addPrefix</name>
          <param const="http://www.festwochen.at/Event.html?eventid=" type="java.lang.String"/>
          <param xpath="/site/content/main/event/@id" type="java.lang.String"/>
        </method>
      </object>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.w3.org/2002/12/cal/ical#location"/>
      <object xpath="/site/content/left_frame/location/text()"/>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.w3.org/2002/12/cal/ical#dtstart"/>
      <object ref="start_date"/>
    </triple>
    <triple>
      <subject ref="start_date"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
      <object const="http://www.w3.org/2002/12/cal/ical#Date-Time" resource="true"/>
    </triple>
    <triple>
      <subject ref="start_date"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#value"/>
      <object>
        <method>
          <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.extractStartDate</name>
          <param xpath="/site/content/right_frame/dates[@id='$$date_id$$']/time/text()" type="java.lang.String"/>
          <param xpath="/site/content/right_frame/dates[@id='$$date_id$$']/date/text()" type="java.lang.String"/>
        </method>
      </object>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate const="http://www.w3.org/2002/12/cal/ical#dtend"/>
      <object ref="end_date"/>
    </triple>
    <triple>
      <subject ref="end_date"/>
      <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
      <object const="http://www.w3.org/2002/12/cal/ical#Date-Time" resource="true"/>
    </triple>
```

Continued in Figure 6.11.

Figure 6.10: WEESA mapping definition for a VIF event Web page using the iCalendar ontology. Part 1/2

Figure 6.10 continued:

```
77      <triple>
78        <subject ref="end_date"/>
79        <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#value"/>
80        <object>
81          <method>
82            <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.extractEndDate</name>
83            <param xpath="/site/content/right_frame/dates[@id='$$date_id$$']/time/text()" type="java.lang.String"/>
84            <param xpath="/site/content/right_frame/dates[@id='$$date_id$$']/date/text()" type="java.lang.String"/>
85          </method>
86        </object>
87      </triple>
88      <triple>
89        <subject ref="end_date"/>
90        <predicate const="http://www.w3.org/1999/02/22-rdf-syntax-ns#value"/>
91        <object>
92          <method>
93            <name>at.ac.tuwien.infosys.weesa.wf.MappingLib.calculateEndDateTime</name>
94            <param xpath="/site/content/right_frame/dates[@id='$$date_id$$']/time/text()" type="java.lang.String"/>
95            <param xpath="/site/content/right_frame/dates[@id='$$date_id$$']/date/text()" type="java.lang.String"/>
96            <param xpath="/site/content/right_frame/value_pair[sub_title='Length:']/description/text()" type="java.lang.String"/>
97          </method>
98        </object>
99      </triple>
100   </triples>
101 </mapping>
```

Figure 6.11: WEESA mapping definition for a VIF event Web page using the iCalendar ontology. Part 2/2

this section we discuss the WEESA KB for the case study that uses the self-defined VIF ontology, introduced in Section 6.1.1. The `writeDB` and `updateEvery` attributes of the `<mapping>` element at the beginning of the WEESA mapping definition control the accumulation and maintenance of the meta-data in the KB.

After initializing the WEESA KB, the RDF DB contains over 10,000 RDF statements. This number is made up of the number of statements generated for the Web pages using the WEESA mapping and the number of statements added for maintenance purpose via reification. As discussed in Section 5.2.3 RDF reification increases the number of RDF statements significantly. The actual number of generated RDF statements for the Web pages is about 1,600 RDF statements.

The number of actually accumulated RDF statements originates mainly from the the description of the event Web pages. Seven statements are generates to describe an event, and six statements are added for each performance date of the event. The event overview page does not add new statements to the WEESA KB, since the statements generated for overview pages are also generated for the event description page. As an optimization measure we could therefore set the `writeDB="false"` attribute without loosing data in the WEESA KB.

In the remainder of this section we show sample scenarios that illustrate the added value the WEESA KB of a Semantic Web application offers to software agents. The scenarios are based on the Web application using the VIF ontology introduced in Section 6.1.1.

SCENARIO 1: REQUESTING THE WEB PAGE OF AN EVENT

A business woman has scheduled a business trip to Vienna at the beginning of June. During her stay in Vienna she is interested in visiting an opera. She queries a future Semantic Web search

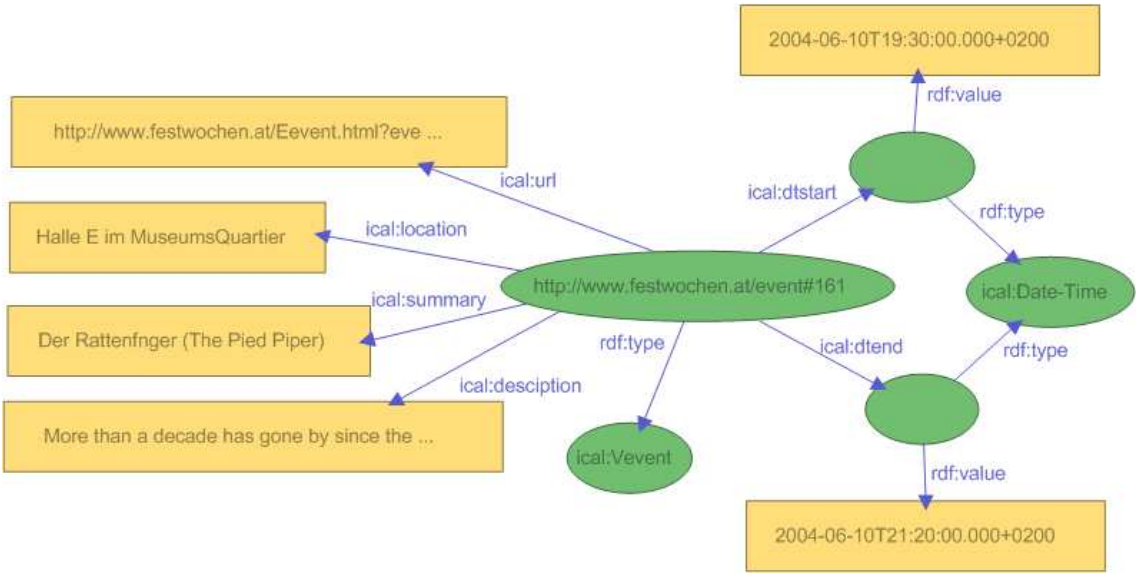

Figure 6.12: RDF graph of the meta-data generated for and event page using the iCalendar ontology.

engine for operas in Vienna at the beginning of June. Among the search hits, the opera "The Pied Piper" raises her interest and she wants to learn more about the this opera from the event Web page. Her software agent can then use the `getURL` method of the Query Interface of the WEESA KB to request the URL of the Web page describing the opera event. The parameter of the method call is the URI representing the selected event (in this example `http://www.festwochen.at/event#23`).

SCENARIO 2: CHECKING IF TICKETS FOR AN EVENT ARE AVAILABLE

A user is enthusiastic about the VIF festival and instructs his software agent to download a snapshot of the WEESA KB for local querying. While browsing the festival events he comes across the opera "The Pied Piper" and wants to know if there are tickets available. His software agent uses the `getUpToDateObject` method of the Query Interface of the WEESA KB to get the link to the ticket shop for this event. The parameters in our example are the URI of the event id (`http://www.festwochen.at/event#23`) and the predicate URI for the ticket shop property (`http://www.festwochen.at/ontology#ticketShopURL`). If there are tickets available, the return value of the method call is the link to the ticket shop. If no more tickets are available, the query result is empty. After buying a ticket in the online ticket shop, he instructs his software agent to add the performance date to his personal calendar.

SCENARIO 3: SEARCHING FOR AN EVENT ON A GIVEN DAY

A business woman has a meeting in Vienna that ends at 6:00 pm and has no plans for the evening. Since she is interested in cultural events she is looking for tickets of a festival event on the 2004-06-10 starting after 7:30 pm. She takes her software agent and specifies her query in a user-friendly interface. The agent transforms the query into a SeRQL query [92] and sends the request to the `query` method of the Query Interface of the WEESA KB. The method then searches the WEESA KB and returns the requested information in RDF/XML format.

Figure 6.13 shows the SeRQL query [92] for the scenario described above. In the `FROM` part of the query (lines 7-11) the RDF triples are specified that should be matched in the RDF DB. Variables in the query are written in curly braces. In the `WHERE` part (lines 13-15) additional conditions the variables have to match are defined. The `CONSTRUCT` part (lines 2-5) defines the triples that should be generated for the query result. The query result contains an RDF graph for each matching event like the one shown in Figure 6.14.

```
1  CONSTRUCT
2    {event_id} vif:hasEventName {eventName},
3    {event_id} vif:ticketShopURL {ticketShopURL},
4    {event_id} vif:eventDate {date_id},
5    {date_id} vif:beginTime {beginTime}
6  FROM
7    {event_id} vif:eventDate {date_id},
8    {event_id} vif:hasEventName {eventName},
9    {event_id} vif:ticketShopURL {ticketShopURL},
10   {date_id} vif:beginDate {beginDate},
11   {date_id} vif:beginTime {beginTime}
12 WHERE
13   event_id = <http://www.festwochen.at/event#23> AND
14   beginDate = "2004-06-10" AND
15   beginTime >= "19:30:000"
16 USING NAMESPACE
17   vif = <http://www.festwochen.at/ontology#>
```

Figure 6.13: Sample SeRQL query to the WEESA KB of the VIF Semantic Web application.

6.3 LINKING TO AN EXTERNAL RDF/XML META-DATA DESCRIPTION

We also implemented the VIF case study as a Web application that uses the `<link>` element in the `<head>` of the HTML page to associate HTML and RDF. This case study is based on the implementation that embeds RDF in the HTML page discussed in the previous section. Since the previous case study used the same naming convention to select the pipeline for servicing a request as the one introduced in Section 4.2.3 we are able to to use the `AddRDFLink` transformer to add the `<link>` to the HTML page. Therefore, this case study could be implemented by simply reconfiguring the pipelines in the Cocoon `sitemap.xmap` configuration file. The XML

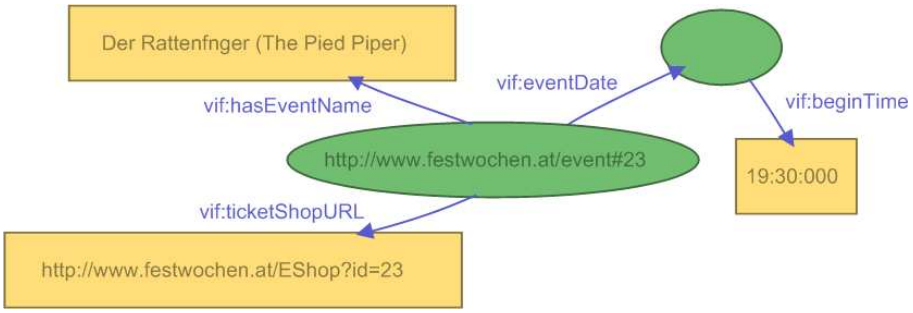

Figure 6.14: RDF graph of the Query result.

Schemas, the WEESA mapping definitions, the business logic, and the XSLT stylesheets remain the same.

For the case study, we again need two types of pipelines: one for the HTML pages that also includes the `<link>` to the external RDF description and one to generate the RDF/XML meta-data description using the WEESA mapping. The structure of the pipeline for HTML pages is shown in Figure 4.6. In this case study we use the `serverpage` generator that processes the business logic, as described in the previous section. The XML Schema valid XML document is then transformed by the XSLT transformer into the HTML page. The `AddRDFLink` transformer adds the `<link>` to the RDF description into the HTML page and the serializer serializes the HTML page for client consumption. The pipeline for the RDF/XML generation is the same as the one defined for the previous case study.

As explained in Section 5.2.1 we extended only the `WEESAReadDOMSession` transformer to write the generated meta-data to the WEESA KB. Therefore, this case study implementation does not provide a KB for querying and download.

We also implemented this case study for both ontologies: the self defined VIF ontology and the iCalendar ontology. Again, the change of the ontology used in the Web application required only in the change of the WEESA mapping definition.

A discussion of the experiences gained in the case study and hints developers should follow when designing WEESA Semantic Web applications can be found in the following chapter.

CHAPTER 7

USING WEESA IN SEMANTIC WEB APPLICATIONS

> Looking at the proliferation of personal web pages on the net,
> it looks like very soon everyone on earth will have 15 Megabytes of fame.
>
> MG Siriam

In the previous chapter we showed the use of WEESA in the VIF Semantic Web application. In this chapter we discuss the experiences gained and the lessons learned while implementing the case study, we list the qualifications a developer should have to semantically annotate a Web application, and give guidelines to develop a Semantic Web application using WEESA.

7.1 LESSONS LEARNED

In this section we summarize the experiences gained while implementing the VIF case study realizations based on the different requirements described in the previous Chapter and give hints to developers who use WEESA to develop Semantic Web applications.

To implement the case study we started developing a traditional Web application. We followed the steps introduced in Chapter 3 and defined the XML Schemas for the Web pages as the contract of the Web application. In the next step we authored the XML content documents that are valid to the corresponding XML Schema. For the dynamic Web pages we wrote XSP pages implementing the business logic that contains SQL DB queries and Java code for the online ticket shop. The XSLT stylesheets finally define the graphical appearance of the Web page. After implementing the pipelines for the HTML page generation shown in Figure 4.1, the traditional Web application without semantic annotations was finished.

To add semantic annotations to the Web application developed so far, in order to realize a Semantic Web application, we defined the ontology to be used and designed the WEESA mapping definition for each Web page that should be annotated. In the next step, we modified the `sitemap.xmap` configuration file to add the WEESA enabled transformers into the Cocoon pipelines. Depending on the association style between HTML and RDF we used the `WEESAReadDOMSession` or the `WEESA` transformer. To set up the WEESA KB, we defined the RDF DB filename, the port number of the XML-RPC service, the time intervals for the Snapshot and Update Daemon, and the snapshot directory in the KB configuration file and added the error handling pipeline shown in Figure 5.11 into the `sitemap.xmap` file to handle "404 not found" errors.

With these additional steps we turned the traditional Web application developed first into a Semantic Web application. In the following we discuss our experiences and the lessons learned while implementing the VIF case study.

ANNOTATION BY CONFIGURATION

The change from a traditional Web application to a WEESA Semantic Web application requires basically two steps: the WEESA mappings have to be defined and the Cocoon pipelines have to be modified. Both of these steps can be done by either writing new XML files (mapping definition) or modifying existing ones (pipeline configuration). No Java programming is involved in these steps.

In addition, the change between the two association styles of HTML and RDF, also could be done by only modifying the pipeline configuration. The WEESA mapping definitions remains the same.

CHANGE OF ONTOLOGIES

We implemented the VIF case study Semantic Web application for different ontologies. For the change from one ontology to the other, only the WEESA mapping definition used had to be changed. The rest of the implementation remained untouched and no additional programming effort was needed.

JAVA MAPPING LIBRARY

In the two paragraphs above we argued, that no Java programming is involved to semantically annotate a Cocoon Web application using WEESA. This is true regarding the Web application. To handle the granularity problem in the WEESA mapping definition, Java methods are used. WEESA provides a library of Java methods for common tasks. In some cases, however, the developer has to implement a Java method for a user-specific task. This user specific method can be added to the library for later reuse.

FREE-TEXT AND MIXED CONTENT

Since WEESA uses the structure of the XML document to identify the concepts that are mapped to the ontologies, free-text and mixed content can not be annotated. Natural language understanding would be needed to do so. However, in our experience this is not a problematic limitation since the concepts that can be found in many ontologies available today can also be found in the structure of an XML document.

DATABASE KEYS FOR RDF RESOURCE IDENTIFIER

The VIF case study further showed that database keys should be accessible in the XML documents to be able to generate unique resource identifiers for the RDF representation. The database keys help to ensure that the same identifier is used for the same resource throughout the whole Web application.

WELL FORMATTED DATA IN THE XML DOCUMENT

Since the RDF meta-data description of the Web pages is intended to be machine-processable the literals used should store information in a well defined format. For example to represent time and date information the XML Schema `xsi:time` respectively, `xsi:date` format should be used. This requirement should also be kept in mind when defining the XML Schema as contract for a Web page. The schema should force content editors to provide information split up into logical units that are stored in a well defined format. For example, to specify the begin and end time of an event, it is better to have a specific XML element for the begin and end time instead of an arbitrary string such as `"form 11:00 to 12:00"`. Using arbitrary strings results in pattern matching, when defining the WEESA mapping.

FORMAT CONFLICT: HUMAN VERSUS MACHINE

In WEESA, the information stored in the XML document is used twice: to generate the HTML page, and to generate the RDF meta-data description. The information on the HTML page is intended for human consumption and should therefore be stored in human-friendly format. The information in the RDF description is intended for machine processing and should therefore be stored in a well defined format. These two intensions can lead to a conflict. For example, the ontology uses the XML Schema `xsi:dateTime` format to represent the end time of an event (e.g. `"2005-05-24T12:00:000+01:00"`). This is not the day and time format a user would expect to find on an HTML Web page. A more human-friendly version should be used e.g. `"Tuesday, May 24 2005, 12:00 am"`.

Since the XSLT transformation cannot do the conversion from the `xsi:dateTime` format to the human-friendly format, the information in the XML document has to be provided human-friendly. As explained in the paragraph above, the human-friendly information should be still stored in a well defined format. Then, Java methods can be used in the WEESA mapping definition to put together the string in the correct format for the RDF description. A solution for the

example above is, to store the end time split up into three parts: the day of the week in plain text ("Tuesday"), the date in xsi:date format ("2005-05-24"), and the time in xsi:time format ("12:00").

The use of the split up elements further increases the flexibility to future changes of the Web application. If the layout of the web page changes the XSLT stylesheet can be used to rearrange the way the date is presented on the screen. In the case used the ontology changes later in the life cycle of the Web application, the separated elements can be mapped to the concepts in the new ontology without parsing a string to find the required information.

SEPARATION-OF-CONCERNS FOR PARALLEL DEVELOPMENT

WEESA has been designed to follow the concept of separation-of-concerns. This concept helps to define strict roles in the development process of the Web application and enables parallel development. Once the XML Schemas are defined as contract for each Web page, the content editor, the graphic designer, the business logic programmer, and the developer responsible for defining the WEESA mappings can work in parallel.

In the paragraphs above we gave hints, what kind of information should be provided in the XML document and how the information should be formated. This hints must be taken into account when defining the XML Schema as contract of a Web page. Therefore, the developer that is responsible for defining the WEESA mappings, has to be involved in the definition of the contracts of the Web application.

COCOON CACHING MECHANISM

The frequency the RDF meta-data description is generated by a WEESA enabled transformer component, depends on the association style between HTML and RDF. When the HTML page links to an external RDF description, the WEESA enabled transformer has to run on request only. When the RDF description is embedded into the HTML page, the meta-data description has to be generated for every page request. To keep the application scalable, the Cocoon caching mechanism can be used. When the XML document that is read by the generator component has not been changed since the last request, the subsequent requests are served by the Cocoon cache. This way the WEESA enabled transformer and the XSLT transformation are not processed.

For dynamic Web pages, such as pages that are based on XSP documents, the Cocoon caching mechanism cannot tell if the dynamic content has changed since the last request (e.g. if the information in the DB has changed). Therefore the caching mechanism does not work for this type of Web pages. To be still able to cache Web pages that are based on XSP, developers can specify within the XSP page that the result of one processing of the XSP page can be cached a given amount of time [17]. A request to a pipeline that is based on such an XSP document is only processed once in the given time period. The other requests are served from the Cocoon cache.

REDUNDANCY IN THE WEESA KB

Many Web applications offer overview pages that provide a list of items which are linked to a detailed item description. In this case, the RDF description of the overview page typically provides only a subset of RDF statements that is available for all detailed item descriptions. Writing the RDF statements of the overview page to the WEESA KB is therefore a redundant step and the `writeDB` attribute in the WEESA mapping definition can be set to `false` for the overview page.

7.2 REQUIRED SKILLS TO SPECIFY THE WEESA MAPPING DEFINITION

WEESA, the technique to design and develop Semantic Web applications introduced in this thesis, follows strictly the concept of separation-of-concerns. Following this concept enables the definition of roles in the development process and parallel development. The roles that can be identified when using WEESA to develop a Semantic Web application are: the content editor, the graphical designer, the programmer developing the business logic, and the developer responsible for specifying the WEESA mapping definitions. Following the term *knowledge engineer* defined in the expert system area, we call developer that defines the WEESA mappings WEESA *engineer*.

In the design phase of the Semantic Web application, the WEESA engineer, as well as the other involved parties, is responsible for the specification of the contract. Once the contract is specified, the parties start working in parallel. The WEESA engineer has to define the used ontology, define the WEESA mapping, and configure the WEESA knowledge base. Detailed guidelines for developing WEESA Semantic Web applications can be found in the following section. The skills a WEESA engineer needs to accomplish his tasks are listed below:

- Knowledge about the domain of the Web application. The engineer has to know the types of Web pages, the intended content, and the involved data sources in the background, such as databases.
- The WEESA engineer has to be familiar with Semantic Web technologies. The engineer has to know the standard ontologies defined in the domain. If no suitable ontology is available, a new ontology has to be developed or an existing one has to be extended. Therefore the WEESA engineer needs to know the ontology definition languages RDF Schema and OWL, knowledge about modeling an ontology for a domain, and the use of ontology editors. The engineer further has to know the RDF data model and the RDF query language SeRQL [92].
- The WEESA engineer has to be familiar with XML technologies such as XML, XML Schema, XML namespaces, and XPath. This knowledge is needed to define and interpret the contracts and to specify the WEESA mapping.

- Of course, the WEESA engineer has to know WEESA. The engineer has to know the WEESA approach, how to specify the WEESA mapping definition, the WEESA enabled Cocoon transformers, and the architecture and configuration of the WEESA KB.
- To address the granularity problem of the mapping definition, the WEESA engineer needs Java programming knowledge to extend the library of Java methods.
- For the Cocoon pipeline configuration, the WEESA engineer has to know the Web application framework Cocoon. To increase the scalability of the Web application, the engineer further needs to know the Cocoon caching mechanism.

Despite the list of required qualifications is long, a graduate student with Java, XML, and Semantic Web experience should be able to develop a Semantic Web application with Cocoon quite fast.

7.3 Guidelines for developing WEESA Semantic Web Applications

In this section we present guidelines for developers that use WEESA to develop Semantic Web applications. As explained in Chapter 3, WEESA can be integrated in XML-based Web engineering methodologies to develop semantically annotated Web applications. Many Web engineering methodologies such as MyXML [51, 53] or wView [20] exist that are based on XML content documents. Therefore some of the steps listed below can vary depending on the Web engineering methodology used.

In the guidelines listed below all steps are described following the same schema. The attributes to describe the steps are: the name of the step, the description, hints the developers should take into account, the involved parties into this step, whether the step is requested by WEESA or the used Web engineering methodology, and references for further reading. To describe the involved parties we use the three parties that can be found in most Web engineering methodologies: the content editor, the graphic designer, and the logic programmer. In addition we have the WEESA engineer as party responsible for the semantic annotation. Some Web engineering methodologies may also identify other parties.

To develop semantically annotated Web application with WEESA we recommend to follow the steps listed below:

1. **Requirements analysis and design of the Web application**

   Description: The requirement analysis of the Web application and the definition of the structure of the Web application depends on the Web engineering methodology used.

   Hints: n/a

   Involved parties: Depending on the used Web engineering methodology.

Methodology: Web engineering methodology

References: n/a

2. **Specification of the contract of the Web application.**

   Description: For each Web page the XML Schema is defined as contract for the Web application. The XML document that is used to generate a Web page has to be valid against the corresponding XML Schema.

   The parties should also agree on a naming convention of the URLs. The path to HTML pages should end with the suffix `".html"` to RDF meta-data descriptions with `".rdf"`.

   Hints: For database driven Web applications, the database keys should be provided in the XML document to be able to generate unique RDF resource identifier.

   The datatype of XML Schema simple types should be restricted to a well defined format.

   Information items that describe concepts found on the Web page should be provided in separate XML elements/attributes. These items are mapped to (present and future) ontologies.

   It is difficult to semantically annotate free-text and mixed content.

   Involved parties: All partied involved in the Web engineering process.

   Methodology: WEESA, Web engineering methodology

   References: Section 3.2: role of contracts in XML-based Web engineering; Section 7.1: hints that should be taken into account when defining the contract; Section 5.3 discussion of the naming convention.

Once all involved parties have agreed on the contract of the Web application, they can start to develop in parallel. In the following descriptions we list only the steps that are WEESA specific. The other involved parties have to follow the steps proposed by the used Web engineering methodology.

3. **Definition of the used association type between HTML and RDF.**

   Description: The WEESA engineer has to decide which association style between HTML and RDF should be used in the Semantic Web application. The RDF description can be embedded into the HTML page or the HTML page can link to an external RDF description.

   Hints: n/a

   Involved parties: WEESA engineer

   Methodology: WEESA

   References: Section 4.2.1: discussion of the association types between HTML and RDF.

4. **Configuration of the Cocoon pipelines.**

   Description: The Cocoon pipelines have to be defined in the `sitemap.xmap` configuration file. In a first step the pipelines are defined as for a traditional Web Application by the parties defined in the Web engineering methodology. In the next step the WEESA engineer adds the WEESA enabled transformers. Which WEESA enabled transformer is used, depends on the association style selected in the previous step.

   Hints: At the beginning of each pipeline is a pattern matcher that has to follow the naming convention.

   To inform the WEESA KB about Web pages that are no longer available we have to add the `weesa-not-found` action into the pipeline responsible for handling "404 not found" errors.

   The pipelines for downloading the WEESA KB snapshots have to be defined.

   Involved parties: WEESA engineer, graphic designer, logic programmer

   Methodology: WEESA, Web engineering methodology

   References: Chapter 4 introduction of the WEESA enabled Cocoon transformers and the pipeline structure they are used in; Section 5.3.5 use of the `weesa-not-found` action; Section 5.3.7 information about the WEESA KB snapshots.

5. **Definition of the ontologies used.**

   Description: The WEESA engineer has to decide which ontologies are used to semantically annotate the Web application. If no standard ontology exists for the domain, a new ontology has to be designed or an existing one is extended.

   Hints: For easy information exchange standard ontologies should be used whenever possible.

   Involved parties: WEESA engineer

   Methodology: WEESA

   References: Ontology development [66]; ontology library [68]

6. **Specification of the WEESA mapping definitions.**

   Description: For each Web page that should be semantically annotated the WEESA mapping definition has to be specified.

   Hints: For pages that are not of public interest, the `writeDB` attribute has to be set to `false` to prevent the meta-data to be written to the WEESA KB.

   For pages with frequently changing content, the `updateEvery` attribute has to be set to define the update interval the update daemon of the WEESA KB updates the meta-data in the KB.

   Involved parties: WEESA engineer

   Methodology: WEESA

References: Section 3.3 introduction of the WEESA mapping; Section 6.1.1 and 6.1.2 discussion of the mapping definitions used in the VIF case study; Section 5.2.2 WEESA KB specific attributes in the WEESA mapping definition.

7. **Configuration of the WEESA KB**

   Description: The WEESA KB configuration file has to be set up.

   Hints: The time interval defined for the WEESA KB Update Daemon must be coordinated with the update intervals specified in the WEESA mapping definitions. The time interval for the Update Daemon must be less than the smallest number specified in the `updateEvery` attribute of each mapping definition.

   Involved parties: WEESA engineer

   Methodology: WEESA

   References: Section 5.3 explanation of the WEESA KB configuration file; Section 5.2.2 WEESA KB specific attributes in the WEESA mapping definition.

8. **Testing the Semantic Web application**

   Description: Once all involved parties have finished their tasks, the Semantic Web application is ready for testing. The steps to test the Web application are defined in the used Web engineering methodology. In addition, the WEESA engineer has to check the RDF descriptions of the annotated Web pages. Also, the meta-data written to the WEESA KB has to be checked. This can be done by manually checking the content of the RDF database or by issuing test queries to the Query Interface of the WEESA KB.

   Hints: n/a

   Involved parties: Depending on the used Web engineering methodology, WEESA engineer

   Methodology: Web engineering methodology, WEESA

   References: n/a

CHAPTER 8

RELATED RESEARCH AREAS

> Reviewing has one advantage over suicide:
> in suicide you take it out on yourself;
> in reviewing you take it out on other people.
>
> George Bernard Shaw

To our knowledge not much work has been done in developing methodologies to design and implement Semantic Web applications. Work that inspired the development of WEESA, however, can be found in the related research areas *Web engineering* and *semantic annotation*. In this chapter we introduce the related work in these areas and discuss their influence on WEESA.

8.1 WEB ENGINEERING

Web engineering is a subdiscipline of software engineering that aims to provide methodologies to design, develop, maintain, and evolve Web applications. Several methodologies have been proposed in literature, such as OOHDM [88], OO-H [34], and WebML [14]. Most of the Web engineering techniques are based on the concept of separation-of-concerns to define strict roles in the development process. During the Web application development process the involved parties produce a set of engineering artifacts that document the process and enables retracing the design decisions. The output of the Web engineering process are Web applications that provide Web pages in HTML format. Some of the methodologies further support Web pages in other formats such as WML for mobile devices. These Web applications, however, lack semantic annotations and are therefore not suited to engineer Semantic Web applications.

In this section we first discuss a Web engineering methodology that was designed to develop semantically annotated Web applications, followed by a discussion of methodologies that use

Semantic Web technologies in the artifacts during the design process, but do not provide semantic meta-data in the engineered Web application. At the end of this section we introduce existing Web engineering methodologies that can be extended by WEESA to engineer Semantic Web applications.

8.1.1 Semantic Web Engineering

To our knowledge beside WEESA only one approach exists that aims to engineer semantically annotated Web applications. In [73,74] Plessers and De Troyer introduce an extension of the Web Site Design Model (WSDM). WSDM is based on "object chunks", which are elementary design artifacts in engineering process. These object chunks are information items that are assigned to components which in turn are used to build Web pages. Examples for object chunks are the title of a CD, the name of an artist, or the play-time of a track.

To semantically annotate the Web application a mapping from the object chunks to the concepts in one or more ontologies are defined during the design of the Web application. This process is called *conceptual annotation*. The mapping is then used in the actual implementation to annotate the Web pages.

The WSDM extension can be used to annotate static and dynamic Web pages. Dynamic pages are build by object chunks that retrieve their content form a column in a relational database. Defining the mapping from such a dynamic object chunk to a concept in an ontology can be seen as adding semantics to the database column.

The proposed extension of WSDM mainly supported one-to-one mappings. Mismatches in granularity are tackled with the help of intermediate ontologies which can only be used to concatenate object chunks. The intermediate ontology has to be defined by hand. This seems to be a rather complicated way to concatenate two strings. The WSDM extension does not allow any further flexibility to address the granularity problem.

8.1.2 Web Engineering based on Semantic Web Technology Artifacts

Many Web engineering methodologies have been proposed that use ontologies and RDF to formalize the engineering artifacts. These Semantic Web technologies are used to conceptual model the Web application for the given domain. The actual Web application, however, does not contain any semantic annotations.

The Extensible Web Modeling Framework (XWMF) [54,55] aims to use a machine-processable format for the Web engineering artifacts to make the artifacts exchangeable between the multitude of tools that are involved in the Web application life cycle. Therefore, XWMF defines a set of RDF Schema ontologies to define the vocabulary (1) for the structure and the content of the Web application, (2) to support Web engineering tasks, and (3) to specify high-level application-specific concepts for the design of a Web application. XWMF provides tools that

take the RDF description of the Web application that is based on these vocabularies to automatically generate the implementation of the Web application. The generated application does not provide semantically annotated Web pages. The tools, however, implement a query system that supports querying the meta-data but also the data of the Web application.

The Semantic Hypermedia Design Method (SHDM) [58, 89] heavily uses OWL ontologies for domain and methodology specific issues. Ontologies are used for the conceptual model and the navigational model of the application domain. The SHDM further defines method specific ontologies for the abstract and concrete widget interface to model the user interface of the Web application. At runtime a SHDM based Web application uses instances of the navigational model and the requested view, that is obtained from the abstract interface definition, to build the actual Web page in a template engine. The generated Web page, however, does not explicitly contain any of the semantic available in the design artifacts of the Web engineering methodology.

The Semantic Web HTML Generator (SWeHG) [65] follows the opposite goal as WEESA. SWeHG is a tool for generating static HTML Web pages from RDF graphs. SWeHG specifies the RDF to HTML transformation on two levels: On the HTML level, the layout of the Web pages are described using templates; On the RDF level, logical Prolog rules are used to define queries to nodes from the RDF graph. The selected nodes are then mapped to HTML tags in the templates. The SWeHG implementation takes the HTML templates and generates XSLT stylesheets. To generate a HTML Web page SWeHG extracts the page content from the RDF graph and stores it an XML document. The HTML page is then generated using the XSLT stylesheet.

OntoWebber [47,48] proposes a system for managing data on the Web with formally encoded semantics. It aims to enable the reusability of software components, the flexibility in personalization, and the ease of maintainability of data intensive Web applications. OntoWebber uses a domain ontology and a site modeling ontology as basis for the construction of site models as views on the underlying data. Instances of these models are used to create a browsable Web application. To personalize the Web application only the site model has to be modified. Although OntoWebber models the domain in an ontology this model is not accessible in the generated Web application.

8.1.3 WEESA COMPLIANT WEB ENGINEERING METHODOLOGIES

As discussed in Chapter 3 WEESA was designed as a technique that can be used to extend existing XML-based Web engineering methodologies to engineer semantically annotated Web applications. In Section 3.3 we introduces WEESA as mapping from XML Schema to ontologies. WEESA, however, can also be used in Web engineering methodologies that do not define an XML Schema for their XML content documents. The XPath expressions for the WEESA mapping definition can also be specified if we have a representative XML document at hand. In this section we introduce Web engineering methodologies that use XML content documents as basis for the Web pages and can therefor be extended by WEESA to engineer Semantic Web applications.

The XGuide Web development method [51, 53] inspired the development of WEESA. The

central idea of XGuide is to bring the well-established software engineering concepts of interfaces and contracts to the domain of Web engineering. The contracts state the requirements of the Web pages and act as specification of the implementation of the Web application. They further enable parallel development of all involved parties, as discussed in Section 3.2. XGuide uses XML Schema to specify the contracts and XSLT stylesheets for the separation of the content and the layout. Therefore WEESA integrates seamlessly into the XGuide Web development method.

wView [21] is a system for generating Web applications that is based on the declarative specification of the hypermedia design. It supports the separation of the content, navigation, and presentation concern. Each of these aspects in the design process is controlled by a separate specification. Only the specification of the content structure, which is described using UML, must be provided. If not specified, the default options for the navigation and the presentation are used. wView uses a series of XSLT transformations to generate the Web application from the specification. The prototype implementation of wVies generates a Cocoon Web application where WEESA can be integrated as described in Chapter 4.

In [32] the authors propose a method to generate a Web application based on the conceptual model of the application domain. The method used the EER/GRAL-approach [25] to represent the conceptual model as a graph. Once the model is defined, an actual instance of the graph model is created by populating the model with content. The Web page generation process generates XML documents that are transformed into HTML via an XSLT transformation. These XML documents can be used for the WEESA meta-data generation to semantically annotate the Web pages.

The AMACONT approach [5,30] aims to engineer adaptive Web applications that can be adjusted to varying client devices and user preferences. AMACONT uses a component-based XML document format that enables the aggregation an linkage of reusable document components. The document components encapsulate adaptive content, behavior, and layout on different abstraction levels. The approach uses a pipeline based component generator for dynamically transforming adaptable component structures to different Web output formats. To generate the different output formats XML/XSLT is used. To generate the output documents Cocoon pipelines are used. Therefore, WEESA can be integrated in the methodology as outlined in Chapter 4.

8.2 Semantic Annotation

The Semantic Web is based on the assumption that the semantics of the documents available on the Web is accessible by machines. Therefore, semantic annotation is one of the core challenges for building the Semantic Web. Since the Web offers access to documents in various media formats such as text, pictures, audio, and video the research community proposed a wide range of tools that support the user during the annotation process. WEESA is designed to annotate semistructured XML documents, therefore we focus the discussion on text annotation tools. In this section we give an overview of semantic annotation tools and discuss their applicability to engineer Semantic Web applications. We categorize the selected annotation techniques based on

their underlying concept in the following categories: Manuel annotation, semantic interpretation of XML structures, and mapping based techniques.

8.2.1 MANUAL ANNOTATION

Annotating documents by hand is a time consuming an error-borne task. To do so, the users have to type the long and unhandy URIs as resource identifiers to formalize the intended statements in RDF/XML syntax. This process is susceptible to typos an syntactic errors. To eliminate this error sources manual annotation tools offer a graphical user interface to add semantic markup to documents. Most tools use an ontology browser and drag & drop to associate a selected phrase in the document with the selected concept from the ontology. Most manual annotation tools only support the annotation of static documents.

The SHOE Knowledge Annotator [38, 94] is an early annotation tool that does not use the W3C recommendations RDF, RDFS, or OWL but SHOE. SHOE (Simple HTML Ontology Extensions) [59] first published in 1997 is an early ontology and knowledge representation language that can be seen as a predecessor of the current W3C recommendations. The SHOE Knowledge Annotator provides a form based graphical user interface to markup existing Web pages using SHOE ontologies. The annotator only supports the annotation of static Web pages. Annotating dynamic documents leads to performing the same task over and over for a specific pattern of documents.

SMORE [49] (Semantic Markup, Ontology and RDF Editor) of the mindswap project [63] follows the same idea as the SHOE Knowledge Annotator, but supports RDF and OWL. SMORE has an more advanced user interface that even allows to select regions of images for annotation. It provides an embedded HTML editor, a Web and an ontology browser that allows users by means of drag & drop to create instances of ontology concepts from marked phrases in the Web page.

CREAM and its implementation, the OntoMat-Annotizer, provides an semantic annotation and authoring framework [36]. CREAM supports several annotation methods such as manual annotation, authoring of annotated documents, semiautomatic annotation, and the annotation of dynamic pages. When annotating dynamic documents, the database is annotated instead of the HTML page. In the dynamic annotation process predefined database queries are mapped to ontology concepts. This mapping is then used to translate ontology queries, that are entered in a separate query GUI, into database queries.

Manual annotation tools provide support when semantically annotating existing Web pages. But still, the annotation process remains an additional task and has to be performed after the Web page is finished. The annotation process is not integrated in the engineering process of the Web application.

The manual annotation tools introduced above all uses phrases found on a Web page to instantiate a concept defined in an ontology. This one-to-one mapping between HTML and ontologies, however leads to the granularity problem discussed in Section 3.1 and 3.3. For example, the Web page displays a date in a human readable format such as `"Tuesday, May 24`

2005, 12:00am" but the ontology defines the range of the date property to use the XML Schema xsi:dateTime format. In the ontology instance the date should be formated to "2005-05-24T12:00:000+01:00". Therefore, additional processing is needed to meet the requirements of the ontology.

8.2.2 Semantic Interpretation of XML Structures

Many Web engineering methodologies use XML/XSLT to separate the content from the layout of the Web pages. To automatically annotate these Web pages with semantic markup machines needs access to the semantics of the semistructured XML content. As discussed in Section 2.2.1, there is no inherent meaning associated with the nesting of the XML elements. To overcome this problem, several approaches have been proposed in literature to interpret XML unambiguously as RDF statements.

In [56] Klein proposes a procedure to interpret ambiguous general XML documents as unambiguous RDF statements with the help of an RDF Schema ontology. In this approach the structure of the XML document and the RDF Schema are used to make the implicit meaning of the XML document explicit and therefore accessible for machines. The base-line of the procedure is that the ontology specifies which elements/attributes in the XML document are relevant and what role in the ontology they have, i.e. whether the specify a class or a property. The relationship between the ontology and the XML document is established through the use of the same names for the XML element/attribute and the classes/properties in the ontology. The RDF Schema ontology, which specifies the interpretation of the XML document, has to be defined manually. It defines whether an XML label should be interpreted as class or property and how the XML element – attribute relation and the child element relation should be interpreted. Klein's algorithm then take these rules to populate the subject, predicate, and object of the RDF triples with data.

The round-tripping tool between XML and RDF [6] allows to directly interpret XML documents with an RDF Schema using the XML Schema as basis for describing how XML is mapped into RDF and back. The modeling primitives of the XML Schema are therefore interpreted as the class, property, and relationship definition of the ontology. For example, XML Schema simple types are mapped onto a relation between a resource and literal. For each XML Schema complex type a new resource is created that represents an instance of the class. Once the ontology is generated out of the XML Schema using the interpretation rules defined for the round-tripping approach, XML Schema valid XML documents can be transformed into an RDF graph, that uses the vocabulary of the generated ontology. The round-tripping approach also supports the transformation in the other direction, an RDF graph can be written as XML document by following the same interpretation rules.

The approach proposed in "Lifting XML Schema to OWL" [29] is similar to the round-tripping approach introduced above. It aims to lift XML Schemas to OWL ontologies and to transform XML instance documents into RDF graphs. To do so, the approach defines a set of interpretation and transformation rules.

The approaches introduced in this section aims to make the semantics of XML documents accessible to machines by interpreting the structure of the XML document. The content of the XML elements/attributes is mapped to nodes in the RDF graph. Since non of the approaches do allow any further processing of the mapped content, we face the granularity problem discussed in Section 3.1 and 3.3.

The approaches above all use their own ontology to semantically interpret XML documents. The ontology has either to be defined by hand or is generated form the XML Schema. In both cases, however, the ontology is based on the structure of the XML document and the equivalent names have to be used for the XML elements/attributes and the corresponding classes/properties in the ontology. This leads to problems when the XML document should be interpreted according to a standard ontology defined by a third party, as discussed in Section 3.3.

8.2.3 Mapping based Annotation

In this thesis we presented WEESA as mapping based approach to generate RDF meta-data from XML content documents. Elements and attributes from the XML document are mapped to concepts that are defined in an ontology. In this section we introduce other mapping based approaches proposed in literature.

The Meaning Definition Language (MDL) [1,98] defines what an XML document may mean in terms of an ontology, and defines how that meaning is encoded in the elements and attributes of the XML document. Once the meaning of an XML document is defined in MDL, MDL-based tools allow users and developers to interface the XML document at the level of its semantics. Tools developed for MDL enable the automatic conversion of meaning-based requests into structure-based requests, provide a Java API to the semantics to the XML content, and translate XML documents from one XML Schema into another. The MDL translation tool takes the the XML Schemas from the source and target XML format, the ontology used to define the meaning, and the MDL for the source and target XML format and produces an XSLT stylesheet for the transformation of XML documents from the source to the target format. Since for the RDF/XML syntax no XML Schema can be defined the translation tool cannot be used to generate RDF graphs.

In "Mapping XML Fragments to Community Web Ontologies" [3] an approach is presented that aims to provide an unique query interface to XML documents that are based on different document type definitions (DTD). This unique query interface is based on a common domain ontology. The approach uses the DTD and XPath to establish a mapping between XML fragments and ontology concepts. When a user formulates a query, the mapping is used to reformulate the query to the DTD of the corresponding XML document. This way the heterogeneity of the XML formats are hidden from the end-users. The mapping, however, is not used to generate an RDF description from the XML documents.

Both mapping based annotation techniques introduced in this section directly map the content of XML elements or attributes to concepts in the ontology. Since no further processing of the content can be specified we encounter the granularity problem, discussed in Section 3.1 and 3.3.

CHAPTER 9

CONCLUSION AND FUTURE WORK

As an adolescent I aspired to lasting fame,
I craved factual certainty,
and I thirsted for a meaningful vision of human life -
so I became a scientist.
This is like becoming an archbishop so you can meet girls.

M. Cartmill

9.1 CONCLUSION

In the current WWW, Web applications provide their Web pages in HTML format only. This HTML pages are intended to be displayed by a Web browser. The content of the Web pages is expressed in natural language, weakly structured with HTML tags, and its semantics is not accessible to machines. To enable information sharing and reuse across application, enterprise, and community boundaries, however, computers need access to the semantics of the content. To overcome this limitation, Tim Berners-Lee proposed the Semantic Web as an extension of the current Web, in which the information is given a well defined meaning. The meaning of the information on a Web page is formalized using semantic meta-data that is based on concepts defined in ontologies. Therefore, the existence of semantically annotated Web pages is crucial to bring the Semantic Web into being.

The Web engineering community proposed several methodologies to design, develop, maintain, and evolve Web applications. Following this methodologies, the outcome of the Web development process are Web applications that provide Web pages in HTML format that lack semantic annotations.

Researchers in the semantic annotation field developed tools to augment documents with semantic markup, describing the meaning of the document. These tools are based on different techniques such as manual annotation, semantic interpretation of XML structures, and mapping based techniques. The annotation process, however, remains an additional task and is not integrated in the engineering process of the Web application as proposed by the Web engineering community.

In this thesis we introduced WEESA (WEb Engineering for Semantic web Applications) that aims to integrate semantic annotation into the engineering process of the Web application by reusing existing Web engineering artifacts. At the design level, WEESA defines a mapping from elements and attributes defined in an XML Schema to concepts that are defined in one or more ontologies. At the instances level, the WEESA mapping is used in the Web application to automatically generate RDF meta-data from XML content documents. Therefore, the same XML documents are used as source for the HTML page and the RDF meta-data representation. WEESA does not define a new Web engineering methodology, but can be used to extend existing XML-based Web engineering methodologies to develop Semantic Web applications.

To support developers when implementing Semantic Web applications we integrated the WEESA meta-data generator into the Apache Cocoon Web application development framework. Different approaches have been proposed to associate the HTML Web page with its RDF meta-data description. To support these different association styles, we integrated WEESA into two new Cocoon transformer components. The `WEESAReadDOMSession` transformer can be used to embed the generated RDF meta-data into the HTML page. The `WEESA` transformer can be used to generate a stand-alone RDF description that is linked by the corresponding HTML Web page. To add this link, we developed the `AddRDFLink` transformer.

Since WEESA follows the principle of separation-of-concerns and the flexible component architecture of Cocoon no additional programming is needed for developing a Semantic Web application compared to the development of a traditional Web application. To semantically annotate the Cocoon Web application only two new steps are needed: (1) The WEESA mapping from the XML Schema to the ontology has to be defined. (2) Depending on the association style between HTML and RDF, the developer has to modify the Cocoon pipeline and add the corresponding WEESA enabled transformer component.

So for we have discussed WEESA as technique to semantically annotate single Web pages. For reasoning and querying purpose, however, it is better to have the full meta-data model of the Web application. Therefore we proposed the accumulation of the meta-data form individual Web pages to build the meta-data model of the whole Web application. This meta-data model is then offered for querying as the knowledge base (KB) of the Web application. In addition a snapshot of the WEESA KB is taken and offered for download by software agents. Opposite to Web crawlers proposed in literature, the WEESA approach generates the KB at server side. This has the advantage, that the KB management component can take the responsibility to keep the KB consistent with the content of the Web application.

After the introduction of the WEESA mapping and the WEESA KB we presented the evaluation of the suggested techniques in the Vienna International Festival (VIF) industry case study. We implemented the case study based on different ontologies and for different associations styles

between HTML and RDF. We further discussed the configuration of the WEESA KB and showed sample scenarios how software agents can benefit from the existence of the KB.

The description of the VIF case study followed a discussion of the lessons learned when implementing the WEESA Semantic Web application. We analyzed that a Cocoon Web application can be semantically annotated by specifying the WEESA definitions and adding the WEESA enabled Cocoon transformers to the Cocoon pipeline. No additional programming afford is needed to do so. Also, a change of the used ontologies requires only in the adaptation of the WEESA mapping definitions. Based on the lessons learned we discussed the qualifications a developer should have to use WEESA to semantically annotate Cocoon Web applications and presented the steps in the development process.

In the last chapter we presented work in the related research areas Web engineering and semantic annotation.

9.2 FUTURE WORK

In this thesis we presented WEESA, an approach to engineer semantically annotated Web applications. These Web applications also provide their accumulated meta-data in a knowledge base (KB) for download and querying. Now that we are able to engineer Semantic Web applications that make the semantics of the content accessible to machines, applications and tools are needed that take advantage of the existence of this meta-data. In literature, such applications are referred to as *software agents*. In this section we present examples, how the users can benefit from the existence of such software agents.

Figure 9.1 shows examples of the use of semantic meta-data at client and sever side. The top left of the figure shows the Semantic Web application with its WEESA KB as introduced in this thesis. At server side, we suggest to harvest the KB of individual Semantic Web applications to build the global KB of a *Semantic Web Search Engine*. At client side, we show how a *Semantic Clipboard* can help the user to reuse the information on a Web page in desktop applications. These applications are only two examples to sketch the potential power of the future Semantic Web.

9.2.1 SEMANTIC SEARCH ENGINE

Based on the idea that Semantic Web applications offer their KB for download we suggest the *Semantic Harvester* architecture. Web applications that provide their KB for download can register at the WEESA Semantic Harvester. The Harvester then collects periodically the KBs from the registered applications and integrates them in its global KB. This is shown at the bottom left of Figure 9.1.

The suggested architecture follows the idea proposed by the Harvester search engine architecture [10] where the index of Web applications is created locally at server side and is then offered to brokers for download. With this architecture, in contrast to Web crawlers, the KB of

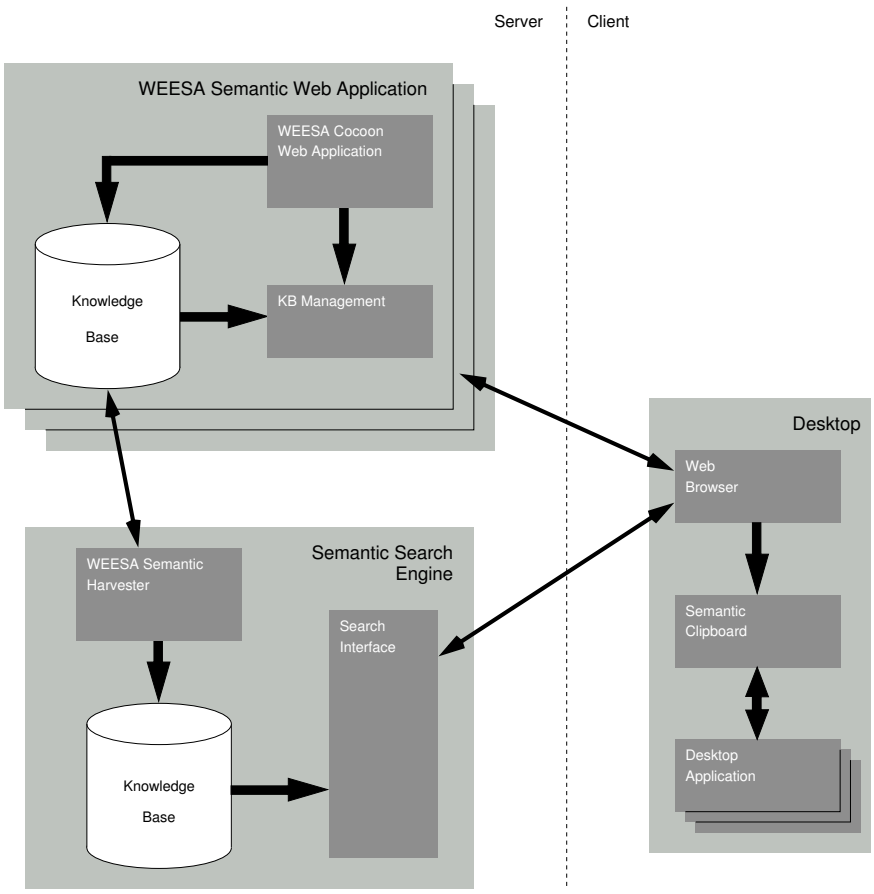

Figure 9.1: Overview of the architecture of the SWEET project.

a Web application can be downloaded as a single stream rather than requiring separate requests for every single Web page.

The global KB built by the Harvester infrastructure is then used in a Semantic Search Engine. Many research initiatives focus on semantic querying [40,95,103]. Offering the user an interface where the user directly has to enter queries in an RDF query language such as SPARQL [75], SeRQL [92], RDQL [80], and RQL [50] requires to much background knowledge of an arbitrary Web user. Therefore, we suggest a wizard like interface for the search engine which uses the knowledge about the domain given in the ontology and the knowledge stored in the global KB to help a user to refine his search query.

9.2.2 Semantic Clipboard

Currently, data can be transfered between desktop applications from different vendors via copy and paste as (rich) text only. The semantics of the data is lost. To overcome this shortcoming we suggest to realize a *Semantic Clipboard* that uses Semantic Web technologies to allow data to be

shared across application boundaries preserving its semantic.

For example, in our VIF case study form Chapter 6 a user ordered a ticket for an event in the online ticket shop and gets the receipt of the tickets bought. After the user got this final confirmation, the user typically enters the dates into the personal calendar. Currently this has to be done manually. Since in our case study the receipt Web page is semantically annotated with all event details, the user can take advantage of the Semantic Clipboard and copy and paste the Web page to the calendar application. The calendar application has access to the semantics of the Web page via its meta-data description and can add the dates of the events to the personal schedule.

Copy and paste the same meta-data to different applications can have different reactions as a consequence. For example, a user copies the e-banking posting data to the clipboard. Depending on the target application pasting the data has a different semantics. Pasting the data to a calendar application issues that a calendar entry is added for each booking line; pasting the data to a spread sheet means that the data is formatted in a table; pasting the data into MS Money the bookings are performed on the corresponding account.

Different applications might use different ontologies for overlapping domains. Therefore, ontology mediation is needed to map between the two ontologies. The current WEESA prototype maps XML structures to concepts in ontologies. Based on this experience we plan to extend WEESA to map between ontologies and use the WEESA ontology mediator in the Semantic Clipboard.

APPENDIX A

XML SCHEMA FOR WEESA MAPPING DEFINITION

XML Schema for the WEESA mapping definition that was introduced in Chapter 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="mapping">
    <xs:annotation>
      <xs:documentation>WEESA Mapping</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="resources" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Container for resources</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="resource" type="ResourceType" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>Resource definition</xs:documentation>
                </xs:annotation>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="triples">
          <xs:annotation>
            <xs:documentation>Container for triples</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="triple" type="TripleType" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>Statement triple</xs:documentation>
                </xs:annotation>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
```

```
    <xs:attribute name="writeDB" type="xs:boolean"/>
    <xs:attribute name="updateEvery" type="xs:integer"/>
  </xs:complexType>
</xs:element>
<xs:complexType name="TripleType">
  <xs:annotation>
    <xs:documentation>Triple Type</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="subject" type="SubjectType">
      <xs:annotation>
        <xs:documentation>Subject</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="predicate" type="PredicateType">
      <xs:annotation>
        <xs:documentation>Predicate</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="object" type="ObjectType">
      <xs:annotation>
        <xs:documentation>Object</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ResourceMethod">
  <xs:annotation>
    <xs:documentation>Resource Method Type</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="name" type="xs:string">
      <xs:annotation>
        <xs:documentation>Resource method name</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="param" type="ResourceParam" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Resource method parameter</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="TripleMethod">
  <xs:annotation>
    <xs:documentation>Triple Method Type</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="name" type="xs:string">
      <xs:annotation>
        <xs:documentation>Tripple method name</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="param" type="TripleParam" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Tripple method parameter</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="POType" abstract="true">
  <xs:annotation>
    <xs:documentation>Abstract Type for Predicate and Object</xs:documentation>
  </xs:annotation>
  <xs:sequence>
```

```
      <xs:element name="method" type="TripleMethod" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Object Method</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="xpath" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>XPath expression</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="const" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>String constant</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="ref" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>Predicate reference</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="resource" type="xs:boolean" use="optional">
      <xs:annotation>
        <xs:documentation>Is predicate a resource?</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="PredicateType">
    <xs:annotation>
      <xs:documentation>Predicate Type</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="POType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="ObjectType">
    <xs:annotation>
      <xs:documentation>Object Type</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="POType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="ResourceType">
    <xs:annotation>
      <xs:documentation>Resource Type</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="method" type="ResourceMethod" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Resource Method</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="xpath" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>XPath expression</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="const" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>String constant</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="id" type="xs:string" use="required">
```

```
      <xs:annotation>
        <xs:documentation>Resource ID</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="var" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>Resource variable name</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="anonymous" type="xs:boolean" use="optional">
      <xs:annotation>
        <xs:documentation>Is anonymous resource?</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="AbstractParam" abstract="true">
    <xs:annotation>
      <xs:documentation>Abstract Parameter Type</xs:documentation>
    </xs:annotation>
    <xs:attribute name="const" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>String constant</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="type" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>Java data type</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="xpath" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>XPath expression</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="ResourceParam">
    <xs:annotation>
      <xs:documentation>Resource Parameter Type</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="AbstractParam">
        <xs:attribute name="var" type="xs:string" use="optional">
          <xs:annotation>
            <xs:documentation>Variable name</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="TripleParam">
    <xs:annotation>
      <xs:documentation>Triple Parameter Type</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="AbstractParam">
        <xs:attribute name="iter" type="xs:boolean" use="optional">
          <xs:annotation>
            <xs:documentation>Should be iterated through all XPath results?
            Only one parameter must have this attribute set to true.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="xresultAsVector" type="xs:boolean" use="optional">
          <xs:annotation>
            <xs:documentation>Is result a vector?</xs:documentation>
          </xs:annotation>
```

```
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="SubjectType">
    <xs:annotation>
      <xs:documentation>Subject Type</xs:documentation>
    </xs:annotation>
    <xs:attribute name="ref" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>Subject reference</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:schema>
```

APPENDIX B

MYTUNES SAMPLE ONTOLOGY

Ontology for the MyTunes example introduced in Chapter 3.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://example.com/MyTunes#"
  xml:base="http://example.com/MyTunes">
  <owl:Ontology rdf:about="">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Sample ontology for the illustrative example.
    </rdfs:comment>
  </owl:Ontology>
  <owl:Class rdf:ID="Artist"/>
  <owl:Class rdf:ID="Album"/>
  <owl:Class rdf:ID="Track"/>
  <owl:Class rdf:ID="Event"/>
  <owl:ObjectProperty rdf:ID="hasAlbum">
    <rdfs:range rdf:resource="#Album"/>
    <rdfs:domain rdf:resource="#Artist"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasTrack">
    <rdfs:range rdf:resource="#Track"/>
    <rdfs:domain rdf:resource="#Album"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="hasLocation">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Event"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="year">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Album"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasTitle">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Album"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="beginTime">
    <rdfs:domain rdf:resource="#Event"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
```

```
  <owl:DatatypeProperty rdf:ID="trackNumber">
    <rdfs:domain rdf:resource="#Track"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="endTime">
    <rdfs:domain rdf:resource="#Event"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasArtistName">
    <rdfs:domain rdf:resource="#Artist"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="totalTime">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Album"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="trackTitle">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Track"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasEventName">
    <rdfs:domain rdf:resource="#Event"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="playTime">
    <rdfs:domain rdf:resource="#Track"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
</rdf:RDF>
```

APPENDIX C

WEESA KB MANAGEMENT ONTOLOGY

WEESA KB ontology in OWL syntax to maintain the WEESA KB in the RDF DB. The use of the properties is discussed in Section 5.2.3.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.infosys.tuwien.ac.at/weesa/kb#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.infosys.tuwien.ac.at/weesa/kb">
  <owl:Ontology rdf:about="">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Ontology to maintain the WEESA KB.</rdfs:comment>
  </owl:Ontology>
  <owl:ObjectProperty rdf:ID="description"/>
  <owl:DatatypeProperty rdf:ID="updateEvery">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="validUntil">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#unsignedLong"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="lastUpdate">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="url">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="rdf">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
  </owl:DatatypeProperty>
</rdf:RDF>
```

APPENDIX D

VIF ONTOLOGY

VIF ontology used in the case study discussed in Chapter 6.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://www.festwochen.at/ontology#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.festwochen.at/ontology">
  <owl:Ontology rdf:about="">
    <owl:versionInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >1.0</owl:versionInfo>
    <rdfs:label>Wiener Festwoche Ontology</rdfs:label>
  </owl:Ontology>
  <owl:Class rdf:ID="PerformingArts">
    <rdfs:label xml:lang="en">Performing Arts</rdfs:label>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Event"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="February1934">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Event"/>
    </rdfs:subClassOf>
    <rdfs:label xml:lang="en">February 1934</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID="Exhibition">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Event"/>
    </rdfs:subClassOf>
    <rdfs:label xml:lang="en">Exhibition</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID="Ticket"/>
  <owl:Class rdf:ID="Festival">
    <rdfs:label xml:lang="en">Festival</rdfs:label>
  </owl:Class>
  <owl:Class rdf:about="#Event">
    <rdfs:label xml:lang="en">Event</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID="EventDate"/>
  <owl:Class rdf:ID="Zeit_zone">
    <rdfs:subClassOf rdf:resource="#Event"/>
```

```
  <rdfs:label xml:lang="en">zeit_zone</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="Music">
  <rdfs:label xml:lang="en">Music</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Event"/>
</owl:Class>
<owl:Class rdf:ID="Forumfestwochenff">
  <rdfs:label xml:lang="en">forumfestwochen ff</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Event"/>
</owl:Class>
<owl:Class rdf:ID="Concert">
  <rdfs:label xml:lang="en">Concert</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Event"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="boughtTicket">
  <rdfs:range rdf:resource="#Ticket"/>
  <rdfs:domain rdf:resource="#Event"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="eventDate">
  <rdfs:range rdf:resource="#EventDate"/>
  <rdfs:domain rdf:resource="#Event"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasEvent">
  <rdfs:domain rdf:resource="#Festival"/>
  <rdfs:range rdf:resource="#Event"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="hasLocation">
  <rdfs:domain rdf:resource="#Event"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="endTime">
  <rdfs:domain rdf:resource="#EventDate"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#time"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="directedBy">
  <rdfs:domain rdf:resource="#Event"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasDescription">
  <rdfs:domain rdf:resource="#Event"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="quantity">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Ticket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="beginTime">
  <rdfs:domain rdf:resource="#EventDate"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#time"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ticketShopURL">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Event"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="beginDate">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
  <rdfs:domain rdf:resource="#EventDate"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasFestivalName">
  <rdfs:domain rdf:resource="#Festival"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasEventName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Event"/>
```

```
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="endDate">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
    <rdfs:domain rdf:resource="#EventDate"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasLocationName">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="language">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Event"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasPrice">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Ticket"/>
  </owl:DatatypeProperty>
</rdf:RDF>
```

BIBLIOGRAPHY

[1] Kal Ahmed, Danny Ayers, Mark Birbeck, Jay Cousins, David Dodds, Josh Lubell, Miloslav Nic, Daniel Rivers-Moore, Andrew Watt, Robert Worden, and Ann Wrightson. *Professional XML Meta Data*. Wrox Press, 2001.

[2] Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, and Karsten Tolle. The ICS-FORTH RDFSuite: Managing voluminous RDF description bases. In *2nd International Workshop on the Semantic Web (SemWeb'01) at the 10th International World Wide Web Conference*, pages 1–13, Hongkong, May 2001.

[3] Bernd Amann, Irini Fundulaki, Michel Scholl, Catriel Beeri, and Anne-Marie Vercoustre. Mapping XML fragments to community web ontologies. In *Proceedings 4th International Workshop on the Web and Databases*, 2001.

[4] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. The MIT Press, Cambridge, Massachusetts; London, England, 2004.

[5] Peter Barna, Geert-Jan Houben, Flavius Frasincar, and Richard Vdovjak. Semantical description of models for web design. In *Workshop on Application Design, Development and Implementation Issues in the Semantic Web at the 13th International World Wide Web Conference*, New York, USA, May 2004. CEUR Workshop Proceedings. `http://CEUR-WS.org/Vol-105/`.

[6] Steve Battle. Poster: Round-tripping between XML and RDF. In *International Semantic Web Conference (ISWC)*, Hiroshima, Japan, November 2004. Springer-Verlag.

[7] Tim Berners-Lee. A roadmap to the Semantic Web. W3C homepage, September 1998. `http://www.w3.org/DesignIssues/Semantic.html`.

[8] Tim Berners-Lee, R. Fielding, and L. Masinter. RFC 2396 - uniform resource identifiers (URI). IETF RFC, August 1998. `http://www.ietf.org/rfc/rfc2396.txt`.

[9] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific America*, 284(5):34–43, 2001.

[10] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. The harvest information discovery and access system. In *2nd International World Wide Web Conference*, Chicago, Illinois, USA, October 1994.

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

[12] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF Schema. In *Proceedings of the First International Semantic Web Conference on The Semantic Web (ISWC)*, pages 54–68. Springer-Verlag, 2002.

[13] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th International Conference on World Wide Web*, pages 613–622, Chiba, Japan, 2005. ACM Press.

[14] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Language (WebML): a modeling language for design ing Web sites. In *Proceedings of the 9th World Wide Web Conference, Amsterdam, N etherlands*, volume 33 of *Computer Networks*, pages 137–157. Elsevier Science B.V, May 2000.

[15] James Clark and Steve DeRose. *XML Path Language (XPath) Version 1.0*. W3C Recommendation, 16 November 1999. `http://www.w3.org/TR/xpath`.

[16] The Apache Cocoon project homepage, Last visited February 2005. `http://cocoon.apache.org/`.

[17] XSP caching with cocoon head, Last Visited May 2005. http://wiki.apache.org/cocoon/XSPCachingWithCocoonHEAD.

[18] Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. *DAML+OIL (March 2001) Reference Description*. W3C Note, 18 December 2001. `http://www.w3.org/TR/daml+oil-reference`.

[19] DAML-ONT initial release, October 2000. `http://www.daml.org/2000/10/daml-ont.html`.

[20] Andrea R. de Andrade, Ethan V. Munson, and Maria da G. C. Pimentel. Engineering web applications with xml and xslt. In *Proceedings of the WebMedia & LA-Web 2004 Joint Conference 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress*, pages 86–93, Washington, DC, USA, 2004. IEEE Computer Society.

[21] Andrea R. de Andrade, Ethan V. Munson, and Mariada G. Pimentel. A document-based approach to the generation of web applications. In *Proceedings of the 2004 ACM symposium on Document engineering*, pages 45–47, New York, NY, USA, 2004. ACM Press.

[22] Mike Dean and Guus Schreiber eds. *OWL Web Ontology Language Reference*. W3C Recommendation, 10 February 2004. `http://www.w3.org/TR/owl-ref/`.

[23] W3C Document Object Model (DOM) homepage, Last visited March 2005. `http://w3.org/DOM`.

[24] Dublin core metadata initiative homepage, Last visited April 2005. `http://www.dublincore.org/`.

[25] Jürgen Ebert, Andreas Winter, Peter Dahm, Angelika Franzke, and Roger Süttenbach. Graph based modeling and implementation with EER/GRAL. In *Proceedings of the 15th International Conference on Conceptual Modelling (ER'96)*, Berlin, Germany, 1996. Springer-Veralg.

[26] Dave Beckett ed. *RDF/XML Syntax Specification (Revised)*. W3C Recommendation, 10 February 2004. `http://www.w3.org/TR/rdf-syntax-grammar/`.

[27] Jeff Heflin ed. *OWL Web Ontology Language Use Cases and Requirements*. W3C Recommendation, 10 February 2004. `http://www.w3.org/TR/webont-req/`.

[28] Dieter Fensel, Juergen Angele, Stefan Decker, Michael Erdmann, Hans-Peter Schnurr, Steffen Staab, Rudi Studer, and Andreas Witt. On2broker: Semantic-based access to information sources at the www. In *Workshop on Intelligent Information Integration at the International Joint Conference on Artificial Intelligence*, Stockholm, Schweden, August 1999.

[29] Matthias Ferdiand, Christian Zirpins, and David Trastour. Lifting xml schema to owl. In *4th International Conference on Web Engineering*, pages 354–358, Munich, Germany, July 2004.

[30] Zoltán Fiala, Michael Hinz, Geert-Jan Houben, and Flavius Fransincar. Design and implementation of component-based adaptive web presentations. In *Proceedings of the (SAC'04)*, Nicosia, Cypres, March 2004. ACM.

[31] Martin Gaedke and Guntram Graef. Development and evolution of web-applications using the webcomposition process model. In *International Workshop on Web Engineering at the 9th International WorldWide Web Conference*, Amsterdam, the Netherlands, May 2000.

[32] Torsten Gipp and Jürgen Ebert. Conceptual modeling and web site generation using graph technology. Technical report, Universität Koblenz-Landau, Institut für Informatik, 2001.

[33] GlueX, Last visited April 2005. `http://www.javarealm.com/projects`.

[34] Jaime Gomez, Christina Cachero, and Oscar Pastor. Conceptual Modeling of Device-Independent Web Applications. *IEEE Multimedia*, 8(2):26–39, April-June 2001.

[35] The gzip homepage, Last visited March 2005. `http://www.gzip.org/`.

[36] Siegfried Handschuh and Steffen Staab. Annotation of the shallow and the deep web. In Siegfried Handschuh and Steffen Staab, editors, *Annotation for the Semantic Web*, volume 96 of *Frontiers in Artificial Intelligence and Applications*, pages 25–45. IOS Press, Amsterdam, 2003.

[37] Jens Hartmann and York Sure. An infrastructure for scalable, reliable semantic portals. *IEEE Intelligent Systems*, 19(3):58–65, May 2004.

[38] Jeff Heflin and James Hendler. Searching the web with SHOE. In *Artificial Intelligence for Web Search. Papers from the AAAI Workshop*, pages 35–40, Menlo Park, CA, 2000. AAAI Press.

[39] Jeff Heflin, James Hendler, and Sean Luke. Shoe: A blueprint for the semantic web. In Dieter Fensel, James Hendler, Henry Liebermann, and Wolfgang Wahlster, editors, *Spinning the Semantic Web*, pages 29–63. The MIT Press, 2003.

[40] Ian Horrocks and Sergio Tessaris. Querying the semantic web: A formal approach. In *Proceedings of the 1st International Semantic Web Conference*, Sardinia, Italy, 9-12 June 2002. Springer-Verlag.

[41] HTML 4.01: Definition of the script element, Last visited March 2005. `http://www.w3.org/TR/html401/interact/scripts#edef-SCRIPT`.

[42] Eero Hyvönen, editor. *Semantic Web Kick-Off in Finland - Vision, Technologies, Research, and Applications*. HIIT Publications, Helsinki, Finland, 2002.

[43] Renato Iannella. Representing vCard objects in RDF/XML. W3C Note 22 February 2001, 2001. `http://www.w3.org/TR/vcard-rdf`.

[44] Apple ical homepage, Last visited March 2005. `http://www.apple.com/ical/`.

[45] iCalendar OWL ontology definition, April 7 2004. `http://www.w3.org/2002/12/cal/ical`.

[46] Jena - a semantic web framework for java, Last visited April 2005. `http://jena.sourceforge.net/`.

[47] Yuhui Jin, Stefan Decker, and Gio Wiederhold. Ontowebber: Model-driven ontology-based web site management. In *Semantic Web Working Symposium (SWWS)*, Stanford, California, USA, August 2001.

[48] Yuhui Jin, Sichun Xu, Stefan Decker, and Gio Wiederhold. Managing web sites with ontowebber. In *Proceedings of the 8th International Conference on Extending Database Technology*, pages 766–768, London, UK, 2002. Springer-Verlag.

[49] Aditya Kalyanpur, James Hendler, Bijan Parsia, and Jennifer Golbeck. SMORE - semantic markup, ontology, and RDF editor. Technical report, University of Maryland, 2003. `http://www.mindswap.org/papers/SMORE.pdf`.

[50] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. RQL: a declarative query language for RDF. In *Proceedings of the 11th International World Wide Web Conference*, pages 592–603. ACM Press, 2002.

[51] Clemens Kerer. *XGuide - Concurrent Web Development with Contracts*. PhD thesis, TU Vienna, 2003.

[52] Clemens Kerer and Engin Kirda. Web engineering, software engineering and web application development. In *3rd Workshop on Web Engineering at the 9th World Wide Web Conference*, pages 135 – 147, Amsterdam, the Netherlands, May 2000. Springer-Verlag.

[53] Clemens Kerer and Engin Kirda. XGuide - concurrent web engineering with contracts. In *Proceedings of the 4th International Web Engineering Conference (ICWE 2004)*, pages 88–92, Munich, Germany, 2004. Springer-Verlag.

[54] Reinhold Klapsing. Semantics in web enginieering: Applying the resource description framework. *IEEE Multimedia*, 8(2):62–68, April-June 2001.

[55] Reinhold Klapsing and Gustaf Neumann. Applying the resource description framework to web engineering. In *Proceeding of the 1st International Conference on Electronic Commerce and Web Technologies: EC-Web 2000*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[56] Michael Klein. Using RDF Schema to interpret XML documents meaningfully. In Siegfried Handschuh and Steffen Staab, editors, *Annotation for the Semantic Web*, volume 96 of *Frontiers in Artificial Intelligence and Applications*, pages 79–89. IOS Press, Amsterdam, 2003.

[57] Graham Klyne and Jeremy J. Carroll eds. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation, 10 February 2004. `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

[58] Fernanda Lima and Daniel Schwabe. Application modelling for the semantic web. In *Proceedings of the 3th International Conference on Web Engineering (ICWE 2003)*, pages 417–426, Oviedo, Sapin, July 2003. Springer-Verlag.

[59] Sean Luke and Jeff Heffin. Shoe 1.0 proposed specification, 1997. `http://www.cs.umd.edu/projects/plus/SHOE/sepec.hmtl`.

[60] Alexander Maedche, Steffen Staab, Nenad Stojanovic, Rudi Studer, and York Sure. SEmantic portAL: The SEAL approach. In Dieter Fensel, James Hendler, Henry Lieberman, and Wolfgang Wahlster, editors, *Spinning the Semantic Web*, pages 317–359. The MIT Press, 2003.

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

[62] Deborah L. McGuinness and Frank van Harmelen eds. *OWL Web Ontology Language Overview*. W3C Recommendation, 10 February 2004. http://www.w3.org/TR/webont-features/.

[63] Mindswap project homepage, Last visited April 2005. http://www.mindswap.org/.

[64] mindswap: Why do we call this the first site on the Semantic Web?, Last visited April 2005. http://www.mindswap.org/first.shtml.

[65] Eero Myvönen, Markus Holi, and Kin Viljanen. Designing an creating a web site based on RDF content. In *Workshop on Application Design, Development and Implementation Issues in the Semantic Web at the 13th International World Wide Web Conference*, New York, USA, May 2004. CEUR Workshop Proceedings. http://CEUR-WS.org/Vol-105/.

[66] Natalya F. Noy and Deborah L. McGuinness. *ontology development 101: a guide to creating your first ontology*. Stanford University, Last visited May 2005. http://protege.stanford.edu/publications/ontology_development/ontology101.pdf.

[67] OIL at the on-to-knowledge homepage, 2000. http://www.ontoknowledge.org/oil/.

[68] Protege ontologies library, Last visited May 2005. http://protege.cim3.net/cgi-bin/wiki.pl?ProtegeOntologiesLibrary.

[69] Web-ontology (webont) working group. W3C Recommendation, 10 February 2004. http://www.w3.org/2001/sw/WebOnt/.

[70] PageKit web application framework, Last visited April 2005. http://pagekit.org.

[71] Sean B. Palmer. RDF in HTML: Approaches, June 2002. http://infomesh.net/2002/rdfinhtml/index.html.

[72] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks eds. *OWL Web Ontology Language Semantics and Abstract Syntax*. W3C Recommendation, 10 February 2004. http://www.w3.org/TR/owl-semantics/.

[73] Peter Plessers and Olga De Troyer. Annotation for the semantic web during website development. In *4th International Conference on Web Engineering*, pages 349–353, Munich, Germany, July 2004.

[74] Peter Plessers and Olga De Troyer. Web design for the semantic web. In *Workshop on Application Design, Development and Implementation Issues in the Semantic Web at the 13th International World Wide Web Conference*, New York, USA, May 2004. CEUR Workshop Proceedings. http://CEUR-WS.org/Vol-105/.

[75] Eric Prud'hommeaux and Andy Seaborne eds. Sparql query language for RDF. W3C Working Draft, 19 April 2005. http://www.w3.org/TR/rdf-sparql-query/.

[76] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 specification. W3C recommendation, 24 December 1999. http://www.w3.org/TR/html4/.

[77] RDF calendar workspace, Last visited March 2005. http://www.w3.org/2002/12/cal/.

[78] W3C: Frequently Asked Questions about RDF: How do I put some RDF into my HTML pages?, September 2004. http://www.w3.org/RDF/FAQ/#How.

[79] W3C: RDF issue tracking: Issue faq-html-compliance: The suggested way of including RDF meta data in HTML is not compliant with HTML 4.01 or XHTML, January 2004. http://www.w3.org/2000/03/rdf-tracking/\symbol{35}faq-html-compliance.

[80] RDQL - RDF data query language, Last visited March 2005. http://www.hpl.hp.com/semweb/rdql.htm.

[81] Gerald Reif, Harald Gall, and Mehdi Jazayeri. Towards semantic web engineering: WEESA - Mapping XML Schema to ontologies. In *Workshop on Application Design, Development and Implementation Issues in the Semantic Web at the 13th International World Wide Web Conference*, New York, USA, May 2004. CEUR Workshop Proceedings. http://CEUR-WS.org/Vol-105/.

[82] Gerald Reif, Harald Gall, and Mehdi Jazayeri. Using WEESA - to Semantically Annotate Cocoon Web Applications. Technical Report TUV-1841-2005-31, Distributed Systems Group, Vienna University of Technology, 2005. http://www.infosys.tuwien.ac.at/weesa/TUV-1841-2005-51.pdf.

[83] Gerald Reif, Harald Gall, and Mehdi Jazayeri. WEESA - Web Engineering for Semanitc Web Applications. In *Proceedings of the 14th International World Wide Web Conference*, pages 722–729, Chiba, Japan, May 2005.

[84] RFC 2445: Internet calendaring and scheduling core object specification (icalendar). IETF RFC, November 1998. http://www.ietf.org/rfc/rfc2445.txt.

[85] RFC 3870: Application/rdf+xml media type registration. IETF RFC, September 2004. http://www.ietf.org/rfc/rfc3870.txt.

[86] The web robots pages. http://www.robotstxt.org/wc/robots.html.

[87] SAX, simple API for XML homepage, Last visited March 2005. http://www.saxproject.org/.

[88] Daniel Schwabe, Gustavo Rossi, and Simone D. J. Barbosa. Systematic hypermedia application design with oohdm. In *HYPERTEXT '96: Proceedings of the the seventh ACM conference on Hypertext*, pages 116–128, New York, NY, USA, 1996. ACM Press.

[89] Daniel Schwabe, Guilherme Szundy, and Fernanda Lima Sabrina de Moura. Design and implementation of semantic web applications. In *Workshop on Application Design, Development and Implementation Issues in the Semantic Web at the 13th International World Wide Web Conference*, New York, USA, May 2004. CEUR Workshop Proceedings. http://CEUR-WS.org/Vol-105/.

[90] World Wide Web Consortium (W3C) Semantic Web activity homepage. http://w3c.org/sw.

[91] Seminar: Semantic web kick-off in finland - vision, technologies, research, and applications, November 2001. http://www.cs.helsinki.fi/u/eahyvone/stes/semanticweb/kick-off/proceedings.html.

[92] The SeRQL query language, rev. 1.1. User Guide for Sesame, Release 1.1.1, Last visited March 2005. http://www.openrdf.org/doc/users/ch06.html.

[93] Sesame RDF database homepage at openRDF.org, Last visited March 2005. http://www.openrdf.org/.

[94] The SHOE knowledge annotator, Last visited April 2005. http://www.cs.umd.deu/projects/plus/SHOE/nowledgeAnnotator.html.

[95] Michael Sintek and Stefan Decker. Triple - a query, inference, and transformation language for the semantic web. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, Sardinia, Italy, 9-12 June 2002. Springer-Verlag.

[96] The mozilla sunbird project, Last visited March 2005. http://www.mozilla.org/projects/calendar/sunbird.html.

[97] World Wide Web Consortium (W3C) homepage. http://w3c.org.

[98] Robert Worden. Meaning Definition Language (MDL), Version 2.06, July 2002. http://www.charteris.com/XMLToolkit/Downloads/MDL206.pdf.

[99] XHTML 1.0 the extensible hypertext markup language (second edition). W3C recommendation, 1 August 2002. http://www.w3.org/TR/xhtml1/.

[100] Apache XML-RPC project homepage, Last visited March 2005. http://ws.apache.org/xmlrpc/.

[101] XML-RPC home page, Last visited March 2005. http://www.xmlrpc.com/.

[102] XML server pages, Last visited April 2005. http://cocoon.apache.org/2.1/userdocs/xsp/.

[103] Lei Zhang, Yong Yu, Jian Zhou, ChenXi Lin, and Yin Yang. An enhanced model for searching in semantic portals. In *Proceedings of the 14th international conference on World Wide Web*, pages 453–462, Chiba, Japan, 2005. ACM Press.