Towards Semantic Web Engineering: WEESA - Mapping XML Schema to Ontologies

Gerald Reif, Mehdi Jazayeri
Distributed Systems Group
Technical University of Vienna
{*g.reif, m.jazayeri*}*@infosys.tuwien.ac.at*

Harald Gall
Department of Informatics
University of Zurich
gall@ifi.unizh.ch

Abstract

The existence of semantically tagged Web pages is crucial to bring the Semantic Web to life. But it is still costly to develop and maintain Web applications that offer data and meta-data. Several standard Web engineering methodologies exist for designing and implementing Web applications. In this paper we introduce a technique to extend existing Web engineering techniques to develop semantically tagged Web applications. The novelty of this technique is the definition and implementation of a mapping from XML Schema to ontologies that can be used to automatically generate RDF meta-data from XML content documents.

Keywords: Web engineering, Semantic Web, Ontology

1 Introduction

Web Engineering focuses on the systematic and cost efficient development and evolution of Web applications [8]. The outcome of the Web Engineering process are Web applications that provide Web pages that can be displayed in a Web browser and are human-understandable. For the Semantic Web [3] we need a meta-data representation of the content of a Web page that is also machine-understandable to enable agents to access the semantics of the content. In the Semantic Web this meta-data description is done using the Resource Description Framework (RDF) [12] that references the terms defined in one or more ontologies. Ontologies formally define terms used in a domain and the relationship between these terms. An ontology is defined in an ontology definition language such as RDFS [4], DAML+OIL [6], or OWL [13]. In the remainder of this paper we talk about *Semantic Web applications* when a Web application not only offers the content in HTML format but also meta-data that is machine-understandable.

Looking at Web pages that provide an RDF meta-data description, so called *Semantic Web pages*, we recognize that important parts of the content are stored two times. First, in HTML format that is displayed to the user via the Web browser. Second, in the RDF description that is machine-understandable. This redundancy leads to inconsistency problems when maintaining the content of the Web page. Changes always have to be done consistently for both types of information. Therefore, support is needed for the creation and maintenance of Semantic Web pages.

Most Web engineering methodologies are based on separation-of-concerns to define strict roles in the development process and to enable parallel development [9]. The most frequently used concerns are the *content*, the *graphical appearance*, and the *application logic*. When we plan to design a Semantic Web application we have to introduce a new concern, the *meta-data concern*. In this paper we show how this new concern can be used to extend current Web engineering techniques to develop Semantic Web applications. We call the engineering of semantically tagged Web applications *Semantic Web Engineering*. The contribution of this paper is the definition and implementation of a mapping from XML Schema to ontologies that allows the efficient design of Semantic Web applications based on existing Web engineering artifacts. The mapping can then be used to automatically generate RDF descriptions from the content documents. We call this approach WEESA (WEb Engineering for Semantic web Applications).

The remainder of this paper is structured as follows. Section 2 gives a short introduction to XML based Web publishing. Section 3 introduces the idea of using an XML Schema - ontology mapping to generate RDF descriptions from XML documents. Section 4 shows how this mapping can be implemented. Section 5 discusses

related work, Section 6 gives an outlook on the validation and future steps, and Section 7 concludes the paper.

2 XML based Web publishing

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 again an XML document, HTML, or even plain text. Many Web development frameworks such as Cocoon [1] or MyXML [10] exist that use XML and XSLT for separation-of-concerns.

Based on this technology, editors responsible for the content have only to know the structure of the XML file and the allowed elements to prepare the content pages. Designers, responsible for the layout of the Web application, again have only 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 defines exactly the structure and the allowed 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 [9].

Since XML is widely used in Web engineering, we use the XML content to generate the RDF meta-data description of a Web page. We also use the XML Schema as a contract and map the elements defined in the schema to terms defined in an ontology. Our goal is to use the structure and the content of the XML document to fill the RDF triples with data.

In our 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 introduction.

3 Mapping XML Schema to Ontologies

We use the content of an XML document to derive the RDF meta-data description. In the design phase of the Web application, however, we have no XML documents at hand. But we 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 to terms used in an ontology. Figure 1 shows the definition of this mapping on the design level and the actual generation of RDF meta-data descriptions from the XML document on the instance level. The mapping on the design

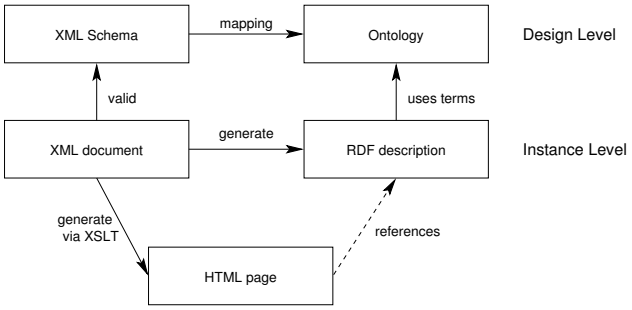

Figure 1. Design and instance level

level has to be done manually. The mapping is then used to automatically generate the RDF descriptions from XML documents.

When defining the mapping it might be the case that the content of an XML element can be mapped one-to-one to a term defined in an ontology. But in general this will not be the case and some additional 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.

We take the Web application of the Vienna International Festival[1] (VIF) as our case study and focus on the programme pages in this paper. The VIF application, our group is responsible for, comprises a ticket shop, over 60 event descriptions, reviews, and an archive over the last 52 years. The pages are offered in German and English. The Web application hosts a Web page for each offered event. The XML document for an event page offers an element with the event name that can be mapped one-to-one to the name property in an event class defined in the ontology. The XML document also offers elements for the begin time and the duration of the performance. The ontology, however, uses a different way to express the performance times. It defines properties for the begin and end time of an event in the event class. Therefore the content of the begin time and the duration element have to be processed to match this two properties.

Another possibility to address the mismatch between the XML elements and the ontology terms would be to adjust the XML Schema definition in the design phase of the Web application. The structure of the XML document could be adopted to the kind of information needed by the given ontology. But this might conflict with the information needed for the HTML page. In addition, the XML Schema - ontology map-

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

ping should be flexible enough to allow to change the used ontology also later in the development process. Adopting the XML Schema to the used ontology results in the change of the contract all involved parties have agreed on and would yield to the redesign of the whole Web application. It is also possible that we have to define the mapping for already existing XML documents and do not have the possibility to change the schema. Therefore, a flexible way to map the content of one or more XML elements to the information required by the used ontology is needed. How this mapping can be implemented is shown in the following section.

4 Implementation

The generation of RDF descriptions based on XML content pages is done in two steps. First, in the design phase for each XML Schema (used as contract in the Web application), a mapping to the ontology has to be defined. Second, for each XML content page the rules defined in the previous step are applied to generate the RDF representation.

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 being used. This information can be used to define XPath [5] expressions to select elements or attributes from an XML document. Once an element/attribute is selected, its content is mapped to a position in an RDF triple.

In the mapping definition various ways exist to specify the content for the RDF triples: (1) a constant value, (2) an XPath expression, (3) the return value of a Java method, and (4) a resource reference. In the following we describe each of this ways in more detail.

(1) A constant value can be, for example, the URI reference to a term defined in the ontology. (2) An XPath expression is used do select the content of an element/attribute. (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 has 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.

(4) Unique resource identifiers are needed to fill 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 provides the possibility to define anonymous resources. They are used for resources that never need to be referred directly from outside the RDF description. An anonymous resource consists of the anonymous resource identifier "`_:`" and a unique random string. To define an anonymous resource in the mapping, the resource is labeled to be anonymous.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xs=
                  "http://www.w3.org/2001/XMLSchema">
  <xsd:element name="events" type="EventType"/>
  <xsd:complexType name="EventType">
    <xsd:sequence>
      <xsd:element name="event" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="name"
                         type="xsd:string"/>
            <xsd:element name="begin_time"
                         type="xsd:time"/>
            <xsd:element name="duration"
                         type="xsd:integer"/>
            <xsd:element name="author" minOccurs="0">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="name"
                               type="xsd:string"/>
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="id"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Figure 2. XML Schema for the event example.

Figure 2 shows the XML Schema of a simplified event page from the VIF case study. A valid sample XML document is shown in Figure 3.

The mapping definition consists of two sections. In the first section the resource identifiers are defined that can be referenced later. In the second section the subject, predicate, and object of the actual triples are defined.

Figure 4 shows the WEESA mapping definition for our example. At the beginning of the mapping we define the resources (lines 3-14). For the first resource with the `id="event"` attribute, we define an XPath expression to select the `id` attribute of the `event` element. The according XPath expression looks as follows: `/events/event/@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

```
<?xml version="1.0" encoding="UTF-8"?>
<events xmlns="http://examle.com/event#">
  <event id="event_id3452">
    <name>Cosi fan tutte</name>
    <begin_time>19:30</begin_time>
    <duration>120</duration>
  </event>
  <event id="event_id6754">
    <name>Hamlet</name>
    <begin_time>19:00</begin_time>
    <duration>210</duration>
    <author>
      <name>Shakespeare</name>
    </author>
  </event>
</events>
```

Figure 3. XML document for the event.

method adds a prefix to the attribute value to generate an identifier. The parameters for the Java method are defined in lines 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="event"` is referenced.

In the resource with the `id="author"` we show how an anonymous resource can be defined. This is done using the `anonymous="yes"` attribute. In this case for each XPath match an anonymous resource is generated.

Once we have defined the resources, we can start defining the RDF triples. This is done in the `triples` section (lines 15-57). In the first triple (lines 16-20) the subject uses the `ref="event"` attribute to reference the resource with the `id="event"`. 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/ontology#Event`). Our sample ontology is shown in Figure 5.

The following triples in our example mapping fill the properties of the `#Event` class. The predicate defines the name of the property and the object the value. The `xpath="yes"` attribute of the `object` element defines that the content contains an 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.

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 one. For this purpose we use variables. In line 8 we use the `var` attribute to define the `event_id` variable. This variable can be used in XPath expressions using the `$$` escape

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf=
        "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns="http://example.com/ontology#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://example.com/ontology">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Event"/>
  <owl:Class rdf:ID="Person"/>
  <owl:ObjectProperty rdf:ID="author">
    <rdfs:range rdf:resource="#Person"/>
    <rdfs:domain rdf:resource="#Event"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="endTime">
    <rdfs:range rdf:resource=
          "http://www.w3.org/2001/XMLSchema#time"/>
    <rdfs:domain rdf:resource="#Event"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="beginTime">
    <rdfs:domain rdf:resource="#Event"/>
    <rdfs:range rdf:resource=
          "http://www.w3.org/2001/XMLSchema#time"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="eventName">
    <rdfs:range rdf:resource=
          "http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Event"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="name">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource=
          "http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
</rdf:RDF>
```

Figure 5. OWL ontology for events.

sequence at the beginning and the end of the variable name. The variable is then replaced at runtime with the actual value of the XPath match defined in this element.

At the end of the triples section we show that anonymous resources can be used as any other resource in the triple definition. In the triple defined in lines 42-46 the object uses the reference to the anonymous resource `author`. The following triples define the class and the properties for this resource.

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

4.2 Generating the RDF Description

When a Web page is queried, the corresponding XML document is fetched and the XSLT transformation is used to generate the HTML page. Second, the RDF description has to be generated and referenced from the HTML document. To generate this description all mappings defined for this document have to be executed. Therefore, the XPath expressions have to be evaluated on this document and the result is either

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping xmlns="http://www.infosys.tuwien.ac.at/mapping#">
  <resources>
    <resource id="event">
      <method>
        <name>com.example.mapping.addPrefix</name>
        <param position="1">http://example.com/event#</param>
        <param position="2" xpath="yes" var="event_id">/events/event/@id</param>
      </method>
      <resource id="author" anonymous="yes" xpath="yes">
        /events/event[@id=$$event_id$$]/author
      </resource>
    </resource>
  </resources>
  <triples>
    <triple>
      <subject ref="event"/>
      <predicate>http://www.w3.org/1999/02/22-rdf-syntax-ns#Type</predicate>
      <object>http://example.com/ontology#Event</object>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate>http://example.com/ontology#eventName</predicate>
      <object xpath="yes">/events/event[@id=$$event_id$$]/name</object>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate>http://example.com/ontology#beginTime</predicate>
      <object xpath="yes">/events/event[@id=$$event_id$$]/begin_time</object>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate>http://example.com/ontology#endTime</predicate>
      <object>
        <method>
          <name>com.example.mapping.calcEndTime</name>
          <param position="1" xpath="yes">/events/event[@id=$$event_id$$]/begin_time</param>
          <param position="2" xpath="yes">/events/event[@id=$$event_id$$]/duration</param>
        </method>
      </object>
    </triple>
    <triple>
      <subject ref="event"/>
      <predicate>http://example.com/ontology#author</predicate>
      <object ref="author"/>
    </triple>
    <triple>
      <subject ref="author"/>
      <predicate>http://www.w3.org/1999/02/22-rdf-syntax-ns#Type</predicate>
      <object>http://example.com/ontology#People</object>
    </triple>
    <triple>
      <subject ref="author"/>
      <predicate>http://example.com/ontology#name</predicate>
      <object xpath="yes">/events/event[@id=$$event_id$$]/author/name</object>
    </triple>
  </triples>
</mapping>
```

Figure 4. Mapping from XML Schema to RDF for the event example.

```
process_mapping(resource_list) {
 for all resource_identifier in resource_list {
   if XPath contains unresolvable variable then
     update dependent_resource_list
   else
     resourceStack.push(all XPath matches)
     stackHash.put(resource_identifier, stack)
 }
 do {
   for all resource_identifier in stackHash {
     stack = stackHash.get(resource_identifier)
     xpath_match = stack.pull()
     content = process_content(xpath_match)
     resourceHash.put(resource_identifier, content)
     if resource defines variable
        variableHash.put(variabel_name, xpath_match)
     if stack was not empty
        process_mapping(dependent_resource_list)
   }
   generate_triples()
 } until all stacks are empty
}
```

Figure 6. Pseudo-code for the mapping.

directly used to fill 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 has to be repeated for each match.

The pseudo-code in Figure 6 shows the principle steps that have to be applied to process the mapping definition. The initial parameter for the `process_mapping` method is a list of all `resource_identifiers` defined in the mapping. In our example we can find two `resource_identifiers`: `event` and `author`. In the first loop we check if the XPath expression contains variables that have not jet been defined. If so, the resource and its dependencies are stored in the global `dependent_resource_list`. Otherwise the XPath expression is executed on the XML document and the result is pushed on the stack.

In our example the XPath for `event` can be executed and the two matches (`event_id3452` and `event_id6754`) are pushed on the stack. The stack is then stored in a global hash-table with the `resource_identifier` as key. Since the XPath expression for `author` contains the `$$event_id$$` variable that has not jet been defined, this resource is added to the `dependent_resource_list`.

In the following two nested loops the RDF triples are generated. In the inner loop the XPath matches are taken from the stack, the `process_content` method is called, the variables are assigned their values, and a recursive method call is done to process all resources that depend on the current resource/variable environment. The `process_content` method takes the XPath match 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. The outer loop calls the `generate_triples()` method and terminates when the inner loop has processed all elements from the stack. The `generate_triples()` 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 global `variableHash` and `resourceHash`.

Returning to our example, in the inner loop we take the first match (`event_id3452`) from the stack, call the Java method defined in line 6 and get the content for the resource (`http://example.com/event#event_id3452`), set the `$$event_id$$` variable to `event_id3452` and do the recursive method call. The `resource_list` for this call contains the `author` resource since it depends on the definition of the `$$event_id$$` variable. In the recursive method call the execution of the XPath expression returns with no result since the event with the `event_id3452` does not have an `author` element in our XML document. The processing of the inner loop is finished and the outer loop causes the generation of the RDF triples. This time only the triples defined in lines 16-41 can be generated since the `author` resource is not defined in the `resourceHash`.

The second time the inner loop is processed the `event_id6754` is taken from the stack. The same steps are applied as before. But this time the recursive call finds a match for the XPath expression in the `author` resource, generates an anonymous resource and writes it to the `resourceHash` with `author` as key. In this case, all required resources/variables are defined and an instance for all triples defined in the mapping can be generated.

The output RDF document for our example can be found in Figure 7.

5 Related Work

To our knowledge, no approach uses a mapping from XML Schema to ontologies to develop semantically tagged Web applications and, therefore further evolve current Web engineering methodologies. However, there exists some work that analyzes the structure of an XML document to access the semantic of the content.

The Meaning Definition Language (MDL) defines what an XML document may mean in terms of a UML class model, and defines how that meaning is encoded in the nodes of the XML document [15]. It enables tools and users to access XML at the level of its meaning rather than its structure. A different approach is

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf=
       "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://example.com/ontology#">
  <Event rdf:ID="envent_id3245">
    <endTime>19:30</endTime>
    <beginTime>21:30</beginTime>
    <eventName>Cosi fan tutte</eventName>
  </Event>
  <Event rdf:ID="event_id6754">
    <author>
      <Person rdf:ID="_:64hdze6fdhz65">
        <name>Shakespeare</name>
      </Person>
    </author>
    <eventName>Hamlet</eventName>
    <endTime>21:30</endTime>
    <beginTime>18:00</beginTime>
  </Event>
</rdf:RFD>
```

Figure 7. RDF representation for the events.

taken in [2]. There the DTD and XPath is used to establish a mapping between XML fragments and ontology concepts. Both approaches do not support variables in the mapping definition, and do not offer the flexibility to further process the XML content in Java methods to better match the ontologies' requirements.

Ontology mapping [14] defines semantic relations between two ontologies on a conceptual level. This mapping is used on the data level to transform an instance of the source ontology into an instance of the target ontology. This idea is closely related to the WEESA mapping presented in this paper. However, in ontology mapping the source and the target are ontologies, that provide natural semantic units (classes, properties and their relationship) to model the domain; but XML Schema defines only the grammar of a valid XML document [7]. Therefore, in the XML Schema to ontology mapping the structure elements of the schema have to be mapped to concepts in the domain model of the ontology.

The XWMF (eXtensible Web Modeling Framework) project uses RDF as a basis for modeling Web applications [11]. RDF is used to specify the structure and the content of a web site and to make statements about the elements of a Web site. But RDF is only used to specify the design documents and not to semantically tag the Web pages.

Based on our experiences in Web engineering of large Web applications such as the VIF, we derived the need to extend existing Web engineering methodologies to establish a link to the Semantic Web. Our WEESA XML Schema to ontologies mapping addresses exactly this link.

6 Validation and Next Steps

Currently we are testing the mapping in our Vienna International Festival case study to see if we can cover the needs of a real world application. At the moment we define the mapping files we use by hand. To get a broader acceptance, tool support is needed to define the mapping. We plan to develop a tool that takes an XML Schema and automatically generates the maximal possible tree structure for this schema. Elements/attributes can then be selected and the XPath expression is generated. On the other side, the class hierarchy and the properties defined in the ontology are graphically displayed. In addition we present a list of available Java methods that can be used to further process the element's/attribute's content. This can then be used to define the mapping in a GUI via drag & drop.

Our current approach focuses on the generation of the RDF representation of individual Web pages. When we accumulate the RDF description of all available Web pages in a Web application, we end up with a meta-data representation of the whole Web application. This can be further used as knowledge base that can be offered as a Web Service.

The RDF representation of a single Web page is generated on demand. This cannot be done for all Web pages when the meta-data representation of the Web application is needed. In this case, the representation has to be pre-generated when the content has changed or a new document is added. In addition, triples have to be removed when a document is deleted or XML elements do not show up in a document anymore. Therefore a daemon is needed to keep track of the changes and keep the knowledge base up to date.

7 Conclusion

The deployment of the Semantic Web requires Web applications that are semantically tagged. On the other hand authoring Web pages that offer data and meta-data is a costly task and has the potential risk of inconsistencies in documents. But inconsistent data weakens the acceptance of the Semantic Web. Therefore, support is needed not only for maintaining but also for designing Semantic Web applications.

This paper presented the WEESA approach to develop Semantic Web applications that is based on established Web Engineering ideas. WEESA uses the same XML documents as source for the HTML page and the RDF representation. In the design phase we define a mapping from XML Schema documents to ontologies. This mapping can then be used to automati-

cally generate RDF descriptions from XML documents. Our approach will enable developers to reuse existing Web engineering artifacts to generate semantically tagged Web applications. Currently, we are evaluating the WEESA approach in the Vienna International Festival Web application to tine-tune our mapping and cope with real-world needs.

Acknowledgements

We thank Clemens Kerer for productive discussions about Web engineering and feedback on earlier drafts of this paper. We thank Pascal Fenkam for proofreading and many suggestions for improving the paper.

References

[1] The Apache Cocoon project homepage. http://cocoon.apache.org/.

[2] B. Amann, I. Fundulaki, M. Scholl, C. Beeri, and A. Vercoustre. Mapping XML fragments to community web ontologies. In *Proceedings Fourth International Workshop on the Web and Databases*, 2001.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific America*, 284(5):34–43, 2001.

[4] D. Brickley and R. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

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

[6] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. *DAML+OIL (March 2001) Reference Description*. W3C Note, 18 December 2001. http://www.w3.org/TR/daml+oil-reference.

[7] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. C. A. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The semantic web: The roles of XML and RDF. *IEEE Internet Computing*, 4(5):63–74, 2000.

[8] M. Gaedke and G. 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.

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

[10] C. Kerer and E. 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.

[11] R. Klapsing, G. Neumann, and W. Conen. Semantics in web engineering: Applying the resource description framework. *IEEE MultiMedia*, 8(4):62–68, April-June 2001.

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

[13] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. *OWL Web Ontology Language Semantics and Abstract Syntax*. W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-owl-semantics-20040210/.

[14] N. Silva and J. ao Rocha. Semantic web complex ontology mapping. In *Web Intelligence 2003 Conference*, Halifax, Canada, October 2003.

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