WEESA - Web Engineering for Semantic Web Applications

Gerald Reif
Distributed Systems Group
Vienna University of Technology
Argentinierstrasse 8
1040 Vienna, Austria
reif@infosys.tuwien.ac.at

Harald Gall
Department of Informatics
University Zurich
Winterthurstrasse 190
8057 Zurich, Switzerland
gall@ifi.unizh.ch

Mehdi Jazayeri
Distributed Systems Group
Vienna University of Technology
Argentinierstrasse 8
1040 Vienna, Austria
jazayeri@infosys.tuwien.ac.at

ABSTRACT

The success of the Semantic Web crucially depends on the existence of Web pages that provide machine-understandable meta-data. This meta-data is typically added in the semantic annotation process which is currently not part of the Web engineering process. Web engineering, however, proposes methodologies to design, implement and maintain Web applications but lack the generation of meta-data. In this paper we introduce a technique to extend existing 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 show how we integrated the WEESA mapping into an Apache Cocoon transformer to easily extend XML based Web applications to semantically annotated Web application.

Categories and Subject Descriptors

H.3.5 [**Information Systems**]: Information Storage and Retrieval; D.2 [**Software**]: Software Engineering

General Terms

Design

Keywords

Web Engineering, Semantic Web, Semantic Annotation, Ontology

1. INTRODUCTION

The existence of semantically annotated Web pages is crucial to bring the Semantic Web to life. 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-understandable meta-data that can be processed by computers.

Semantic annotation addresses this problem and aims to turn human-understandable content into a machine-understandable form by adding semantic markup [7]. 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 development and evolution of Web applications [6]. 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. In this paper we show how existing XML-based Web engineering methodologies can be extended to engineer semantically annotated Web pages. In the remainder of this paper we call Web applications that not only offer human-understandable content but also semantic meta-data *Semantic Web applications*.

The contribution of this paper is the conceptual definition and prototype 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 XML content documents. We call this approach WEESA (WEb Engineering for Semantic web Applications). To our knowledge, WEESA is the first approach that integrates semantic annotation in the Web engineering process. In this paper we show the integration of WEESA into an Apache Cocoon transformer [4] and the use of this transformer to develop Semantic Web applications.

The remainder of this paper is structured as follows. Section 2 briefly introduces semantic annotation. Section 3 introduces XML-based Web engineering. Section 4 explains the idea of using an XML Schema – ontology mapping to generate RDF descriptions from XML documents. Section 5 shows the implementation, and Section 6 the integration of the mapping into a Cocoon transformer. Section 7 discusses related work, Section 8 presents the case study and tool support, and Section 9 gives an outlook on future work and concludes the paper.

2. SEMANTIC ANNOTATION

The aim of semantic annotation is to transform documents into machine-understandable 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) that references the concepts defined in an ontology. 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, DAML+OIL, or OWL.

When adding semantic meta-data to documents, one faces several problems [13]:

- 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 ones 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, 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. 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.
- 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 [8] mainly concentrated on avoiding syntactic mistakes and typos when referencing ontologies. Current tools such a CREAM/OntoMat [7] 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. XML-BASED WEB PUBLISHING

Web Engineering focuses on the systematic and cost efficient design, development, maintenance, and evolution of Web applications [6]. 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 Web applications that in addition offer semantic markup 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 again an XML document, HTML, or even plain text. Many Web development frameworks such as Cocoon [4] or MyXML [10] exist that use XML and XSLT for separation-of-concerns.

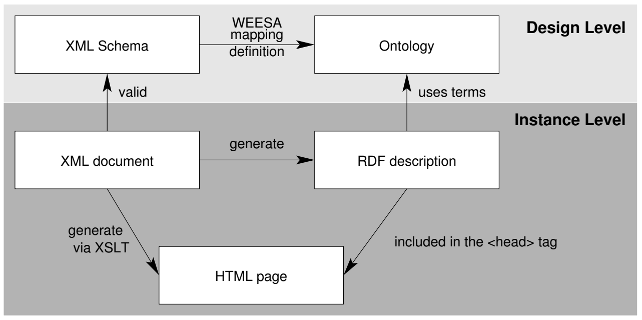

Figure 1: Design and instance level

Based on this technology, editors responsible for the content only have 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 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 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, our approach to engineer Semantic Web applications 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 fill 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 Section before.

4. 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. 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 concepts used in an ontology. Figure 1 shows the definition of the WEESA mapping on the design level and how this mapping is used at instance level to automatically generate RDF meta-data from XML documents.

In Section 2 we introduced the granularity problem 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. Therefore we propose to dynamically compute the missing information from the information available in the XML document. In some cases 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, in this paper we take the fictive MyTunes online CD store as illustrative example. MyTunes offers for each artist a Web page with 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 can not be found in the XML document but calculated from the play-times of each single track. In this case some additional processing is needed to generate the information required by the ontology from the data provided by the XML document. Our MyTunes application offers in addition 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. It defines properties for the begin 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 possibility to 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 adopted to the kind of information needed by the given ontology. But this would lead to several problems: (1) Some information needed for the Web page might be lost. (2) Over time a new ontology can 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 which represents the contract all parties agreed on. This would yield in the redesign of the whole Web application. (3) It is 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.

5. IMPLEMENTATION

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

5.1 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 [3] 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 a RDF triple.

```
<?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 2: XML document for an album.

The goal of the mapping definition is to fill the subject, predicate and object of RDF triples with data. In the mapping definition various ways exist to specify the content of 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 these ways in more detail.

(1) A constant value can be, for example, the URI reference to a concept defined in the ontology. (2) An XPath expression is used do select the content of an element/attribute. In this case, a 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 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 a RDF triple. In the mapping definition we can define that the Java method has to be called for each XPath match and a triple for each match is generated, or that all XPath matches are handed over as a `Vector` to the Java method and only one RDF triple is generated.

(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 to directly from outside the RDF description. To define an anonymous resource in the mapping, the resource is labeled to be anonymous.

Figure 2 shows an example XML document for an album in our MyTunes 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.

A WEESA mapping definition (see Figure 3) consists of two sections. The first section defines the resource identifiers that can later be used. The second section defines the subject, predicate, and object of the actual RDF triples.

```xml
<?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="yes" 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="yes"/>
    </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="yes"/>
        </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="yes"/>
    </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: 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 4: Ontology used for our MyTunes example.

Figure 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 2. The respective 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 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="album"` is referenced.

In the resource with the `id="track"` (line 11) 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 13-59). 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="yes"` attribute is used to indicate that the value of the object should be interpreted as a RDF resource. The default interpretation would be a literal. Our sample ontology is shown in Figure 4. (For reasons of clarity, we do not use the OWL Syntax in the example, but use a trivial textual syntax instead.)

The following triples in our example mapping 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 28-31 show the use of a Java method where all XPath matches are handed over as a `Vector` to the method indicated by the `xresultAsVector="yes"` attribute.

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 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.

```
recurisveMapping(resourceSet)
 if resourceSet is empty
  generateTriples()
  return

 forall resource in resourceSet
  if there are XPath matches
    resourceStack.push(all XPath matches)
    stackHash.put(resource, resourceStack)

    while (stack is not empty)
      xpath_match = stack.pop()
      content = processContent(xpath_match)
      globaleResourceHash.put(resource, content)
      if resource defines variable
        globaleVariableHash.put(variable, xpath_match)
      recursiveMapping(resDependencyHash.get(resource))
  else
    recursiveMapping(empty Set)
```

Figure 5: Pseudo-code for processing the WEESA mapping.

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 34-38 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.

5.2 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 included into the `<head>` tag of the HTML document. To generate this description all mappings defined for the XML document have to be executed. Therefore, the XPath expressions have to be evaluated on this document and the result is either directly used to fill a position in a 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 or the matches are handed over to the Java method as a `Vector`.

The pseudo-code in Figure 5 shows the principle steps that have to be applied to process the 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 (such as defined in lines 34-38) can only be generated if the values for both resources have been processed. The initial parameter

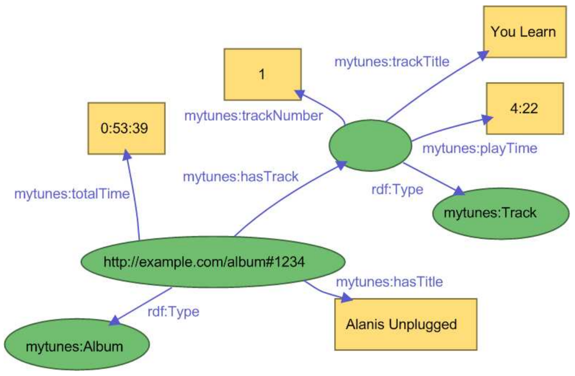

Figure 6: Snippet from the generated RDF graph.

for the `recursiveMaping()` method is the `Set` of resources that do not depend on any other resources and can therefore be processed directly.

For better understanding we use the MyTunes example mapping from Figure 3 to illustrate the execution of the pseudo-code. We start with analyzing the resource dependencies and find out 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 done 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. 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`.

Coming back 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 and empty `Set` and the `generateTriples()` method is called. A part of the generated RDF graph for our example can be found in Figure 6.

There are two known limitations to the current WEESA implementation: (1) If a method call is defined that takes several XPath expressions as parameter, we do not generate all permutations of the matches of the XPath expressions. Instead we use the `iter="yes"` attribute to indicate over which XPath matches should be iterated. For the other parameters we use the first XPath match. (2) Because of the dependency rule between resources WEESA cannot generate RDF graphs that include circles. In this case no independent resource can be found to start the recursion.

6. WEESA AS COCOON TRANSFORMER

In the previous section we introduced the WEESA mapping and illustrated how this mapping can be used to generate RDF meta-data from XML documents. In this section we show how we integrated WEESA into Cocoon transformers and how to use these transformers in a Cocoon Web application.

Apache Cocoon [4] is a component based Web development framework that uses XML/XSLT for separation-of-concerns. Cocoon uses component pipelines to build Web applications where each component on the pipeline is specialized on a particular operation.

Figure 7 shows the pipeline of a typical Cocoon Web application. Each pipeline consists of a *generator*, an arbitrary number of *transformers*, and a *serializer*. A user request causes the generator to read XML from a data source and produces as output a stream of SAX events representing the XML document. This output is the input of a transformer. A transformer takes the SAX events, does some transformation (e.g. XSLT transformation), and the results are again SAX events. This 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 an XML Schema valid XML document which fulfills the contract defined in the Web engineering process. This document is then taken by the XSLT transformer which uses the XSL stylesheet to produce the HTML page. The serializer finally takes the SAX events and processes them into a character stream for client consumption. The steps of a conventional Cocoon Web application are shown in Figure 7 as white pipeline components.

In the case of a Semantic Web application we introduce two new steps to the pipeline. Since we need the Schema valid XML document for the XSLT transformation and for the WEESA meta-data generation, we have two choices. We can either integrate WEESA in a modified XSLT transformer that generates RDF and HTML or we can split up the pipeline. In our approach we choose to split up the pipeline using the `WriteDOMSession` transformer. This transformer takes the input document and writes it first as DOM 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. The meta-data is then added in the `<head>` tag of the HTML page. The serializer finally delivers the HTML+RDF page to the client. The additional steps are shown in Figure 7 as light gray pipeline components.

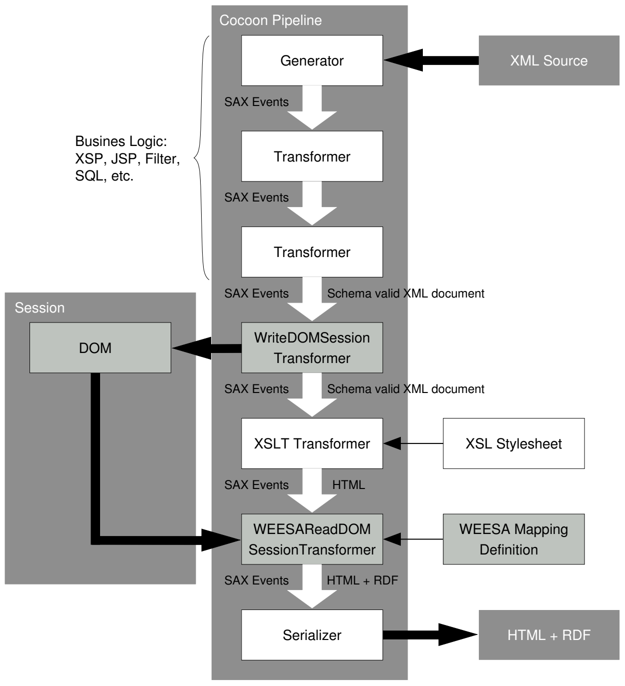

Figure 7: WEESA **meta-data generation in the Cocoon pipeline.**

Embedding the RDF/XML meta-data in the `<head>` tag of a HTML document, however, breaks HTML 4.01 and XHTML validity [14]. The recommended approach is to not embed RDF/XML in HTML/XHTML but rather to use the `<link>` element in the `<head>` element of the HTML/XHTML to point at a separate RDF/XML document [15]. The problem of embedding RDF/XML in HTML is extensively discussed in [12]. Using the `<link>` element, the reference in our example looks as follows:

```
<link rel="meta" type="application/rdf+xml"
                              href="album1234.rdf"/>
```

We also support this way of associating RDF with HTML using the `WEESA` transformer. This second type of transformer, the `WEESA` transformer, takes the XML document as input from the pipeline and produces RDF/XML as output. This output is then taken by a serializer and sent to the client. In this case the pipeline for the HTML page is unchanged, only the `<link>` element has to be added to the Web page. The pipeline for the RDF generation looks similar to that for HTML. Using the `<link>` tag 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.

7. RELATED WORK

To our knowledge not much work has been done to integrate the Semantic annotation process into Web engineering. In [13], the authors suggest an extension of the Web Site Design Model (WSDM). In this approach object chunk entities which are artifacts in the Web application design process are mapped to concepts in the ontology. The WSDM extension also enables the annotation of dynamic pages. Mismatches in granularity are tackled with the help of intermediate ontologies which can only be used to concatenate object chunks and does not allow any further flexibility to address the granularity problem.

There exists other related work that is not directly related to Web engineering but 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 [16]. It enables tools and users to access XML at the level of its meaning rather than its structure. A different approach is taken in [1]. 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 ontology requirements.

There is further related work in the area of Semantic annotation. CREAM/OntoMat [7] is a Semantic annotation framework that offers several annotation methods such as manual annotation, authoring of annotated documents, semiautomatic annotation, and the annotation of dynamic pages. This flexible approach is, however, not integrated in the Web engineering process.

In the area of interpreting XML as RDF data several approaches exist. In [11], XML documents are interpreted as RDF data via a RDF Schema to enable machines to interpret XML unambiguously as a set of statements in the RDF data model. The round-tripping tool between XML and RDF [2] allows to directly interpret XML documents with a RDF model using the XML Schema as basis for describing how XML is mapped into RDF and back. In [5] the idea is that every element/attribute name maps to a RDF property, viewing the structure of the XML document as relational model between parent nodes and their children. All these approaches rely on the equality of the XML element/ attribute names and those of the class/property names in the ontology. This, however, cannot be guaranteed, since ontologies are often defined by third parties.

8. CASE STUDY AND TOOL SUPPORT

We have evaluated WEESA in the annual Vienna International Festival[1] (VIF) industry case study. VIF is a database supported Web application that comprises a ticket shop, over 60 event descriptions, reviews, and an archive over the last 52 years. The experiences of VIF shows that WEESA is well suited to develop Semantic Web applications. However, since WEESA only uses the structure to identify the concepts in the XML document, free-text and mixed content can not be annotated. Natural language understanding would be needed to do so. To our experience, this is not a problematic limitation since the concepts that can be found in many ontologies can also be found in the structure of an XML document.

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 identi-

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

fier is used for the same resource throughout the whole Web application.

At the moment we define the mapping files by hand. To get a broader acceptance, tool support is needed to define the mapping. Currently we develop a tool that takes an XML Schema and automatically generates the maximal valid tree structure for this schema. Elements/attributes can then be selected and the XPath expression is generated. On the other hand, 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.

The prototype implementation of the WEESA mapper and the WEESA transformer can be downloaded under:
`http://www.infosys.tuwien.ac.at/WEESA`

9. CONCLUSION AND FUTURE WORK

The deployment of the Semantic Web requires Web applications that are semantically annotated. 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 designing but also for maintaining Semantic Web applications.

This paper presented the WEESA approach to develop Semantic Web applications that is based on established Web Engineering methodologies. 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 automatically generate RDF descriptions from XML documents. Our approach enables developers to reuse existing Web engineering artifacts to generate semantically tagged Web applications.

Based on our experiences gained from the first prototype and the case study we plan to revise the WEESA mapping. The next version of WEESA should overcome the limitations listed at the end of Section 5.2. The use of XPath 2 might help to simplify the mapping definition. We will further investigate the possibilities to use WEESA's flexible mapping approach to map between ontologies and to use WEESA for ontology mediation.

Our approach focuses on the generation of the RDF representation of individual Web pages. Looking at the meta-data description of a single Web page, however, gives only a very limited view of the information offered by 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, we plan to accumulate meta-data descriptions from Web pages at server side to obtain the meta-data model of the whole Web application. This model can be offered for querying or for download in one stream in contrast to to the current gatherers that open a new connection for each page. Based on this idea we plan to harvest meta-data models form Semantic Web applications to build the knowledge base of a *Semantic Search Engine*.

Acknowledgements

We thank Pascal Fenkam and Engin Kirda for their valuable comments and suggestions for improving the paper.

10. REFERENCES

[1] B. Amann, I. Fundulaki, M. Scholl, C. Beeri, and A.-M. Vercoustre. Mapping XML fragments to community web ontologies. In *Proceedings 4th Int. Workshop on the Web and Databases*, 2001.

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

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

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

[5] M. Ferdiand, C. Zirpins, and D. Trastour. Lifting xml schema to owl. In *4th International Conference on Web Engineering*, pages 354–358, Munich, Germany, July 2004.

[6] 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.

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

[8] J. Heflin and J. 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.

[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] M. Klein. Using RDF Schema to interpret XML documents meaningfully. In S. Handschuh and S. Staab, editors, *Annotation for the Semantic Web*, volume 96 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, 2003.

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

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

[14] 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.

[15] 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/#faq-html-compliance.

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