GENERATION OF WEB APPLICATIONS FROM UML MODELS USING AN XML PUBLISHING FRAMEWORK

Andreas Kraus, Nora Koch

Institute of Computer Science
Ludwig-Maximilians University of Munich
{krausa,kochn}@informatik.uni-muenchen.de

ABSTRACT

In this paper we present a method for the semiautomatic transition from the design models of a Web application to a running implementation. The design phase consists of constructing a set of UML models such as the conceptual model, the navigation model and the presentation model. We use the UML extension mechanisms, i.e. stereotypes, tagged values and OCL constraints, thereby defining a UML Profile for the Web application domain. We show how these design models can automatically be mapped to XML documents with a structure conforming to their respective XML Schema definitions. Further on we demonstrate techniques how XML documents for the conceptual model are automatically mapped to conceptual DOM objects (Document Object Model). DOM objects corresponding to interactional objects are automatically derived from conceptual DOM objects and/or other interactional DOM objects. The XSLT mechanism serves to transform the logical presentation objects representing the user interface to physical presentation objects, e.g. HTML or WAP pages. Finally we present a production system architecture for Web applications using the XML publishing framework Cocoon which provides a very flexible way to generate documents comprising XSLT and XSP (eXtensible server pages) processors.

INTRODUCTION

Software Engineering for Web applications is already supported by a variety of software tools. The so called Model-Build-Deploy-Platforms such as for example Together Control Center or Rational Rose support the development process of Web applications relying on the UML as modeling language. The term Web application here is based on the J2EE specification for enterprise applications where Web applications have a three or four tier architecture and are deployed and executed within an application server. These tools are capable of deploying a Web application directly to an application server.

Although these tools claim to support the whole development process, they offer no much help for modeling the specialties of Web applications because they only include low level implementation elements like Servlets, Java Server Pages or HTML pages as Web modeling elements. More abstract modeling elements for navigation, presentation and user interaction are missing. Therefore, the user needs a method for Web applications with specific modeling elements and an adequate tool support ranging from the Web application design to the implementation.

Many methodologies for Web applications have been proposed since the middle of the nineties. An excellent overview is presented in Schwabe (2001) where the most relevant methods, such as OO-HMethod (Cachero et al.), WebML (Ceri et al.), OOHDM (Rossi et al.), UWE (Hennicker et al.) and WSDM (De Troyer et. al.) are described on the basis of a same case study. Only some of them support automatic generation of Web applications. Until now these methods mainly focused on the design phase; so does our UML-based Web Engineering approach (UWE).

UWE proposes an UML extension – a so called UML profile – and a systematic design method for Web applications (Koch, 2001). It supports user modeling and adaptivity, i.e. dynamic adaptation of the Web application to the user preferences, knowledge or tasks. Here we limit the use of UWE to non-adaptive applications.

In this paper we extend UWE to include task modeling and an innovative method for the semiautomatic generation of an implementation using XML technologies.

The main aspects of the UWE approach as presented here are:

- the use of a standard notation, i.e. UML through all the models,
- the precise definition of the method, i.e. the description of detailed guidelines to follow in the construction of the models,
- the specification of constraints, i.e. augmenting the precision of the models,
- the definition of a stable production system architecture for Web applications,
- the use of an XML publishing framework for implementing Web applications,

- the semiautomatic implementation generated from UML models.

This paper is organized as follows: Section two presents an overview of the UWE development process for Web applications focusing on generation activities. Section three gives a brief description of the design methodology proposed by UWE. Section four introduces the production system architecture. Section five presents the semi-automatic generation process of Web applications. Finally, in the last section some conclusions and future work are outlined.

UWE PROCESS OVERVIEW

The UML-based Web Engineering (UWE) approach presented by Koch (2001) and extended in this paper supports Web application development with special focus on personalization and systematization. It is an object-oriented, iterative and incremental approach based on the Unified Software Development Process (Jacobson et al., 1999). UWE covers the whole life-cycle of Web applications focusing on design and automatic generation. The notation used for design is a "lightweight" UML profile developed in previous works (Baumeister et al., 1999, Hennicker et al., 2000, and Koch et al., 2001). A UML profile is a UML extension based on the extension mechanisms defined by the UML itself. This profile includes stereotypes defined for the modeling of navigation and presentation aspects of Web applications. The UWE methodology provides guidelines for the systematic and stepwise construction of models which precision can be augmented by the definition of constraints in the Object Constraint Language (OCL).

The modeling activities are the requirements analysis, conceptual, navigation and presentation design. In this work task modeling is included to model the dynamic aspects of the application. Currently, an extension of the ArgoUML tool is being implemented to support the construction of the UWE design models. We focus on the semiautomatic generation of Web applications from models using an XML publishing framework. Figure 1 shows an UML class diagram that represents the UWE process overview in a generic way including all models that are built when developing Web applications with an XML publishing framework. We call this approach UWEXML

Artifacts within the development process are depicted as UML packages. The «trace» dependencies describe which artifacts are historical ancestors of each other. The process starts with analysis and design models created by the user in an editor. The design models are transformed by the UWEXML Preprocessor into XML representations which are fed – together with XML documents containing parameters for the generation process – into the UWEXML Generator. The generator generates on the one hand artifacts which can directly be

deployed, denoted by the «import» dependency. On the other hand some of the generated artifacts have to be adapted before deployment, denoted by the «refine» dependency. In this process we consider deployment to an application server providing a physical component model and to an XML publishing framework
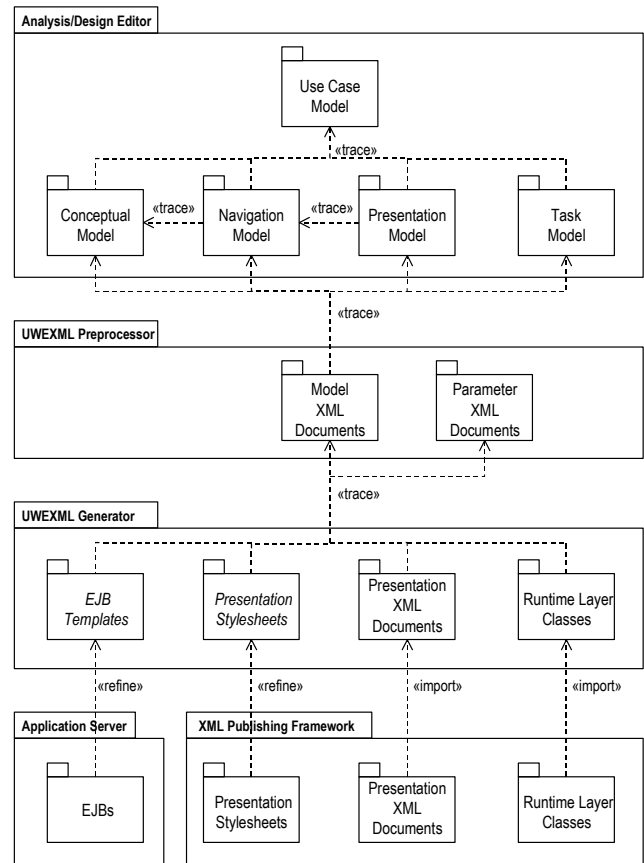

Fig. 1 UWEXML Process Overview

SYSTEMATIC UML-BASED DESIGN OF WEB APPLICATIONS

As a running example to illustrate the generation of a Web application from UML models using an XML publishing framework, the Web site of an online library is used (Koch, 2001). This Online Library application offers users information about journals, books and proceedings. These publications are described by a title, a publisher, a publishing date, a set of articles and authors for each article. In addition, a set of keywords is associated to each article and publication.

In the following we describe the UWE steps for developing a design model consisting of a conceptual model, a navigation model, a presentation model and a task model.

Conceptual modeling. UWE (as a UML-based approach) proposes use cases for capturing the requirements. Conceptual modeling is based on these use cases; a conceptual model includes the objects involved in the typical activities users will perform with the application. The conceptual design aims to build a conceptual model, which attempts to ignore as many of the navigation paths, presentation and interaction aspects as possible. These aspects are postponed to the steps of the navigation and presentation modeling. The main UML modeling elements used in the conceptual model are: class, association and package. These are represented graphically using the UML notation (Jacobson et al., 1999). Figure 2 shows the conceptual model for the Online Library example; for corresponding use cases see (Koch 2001).
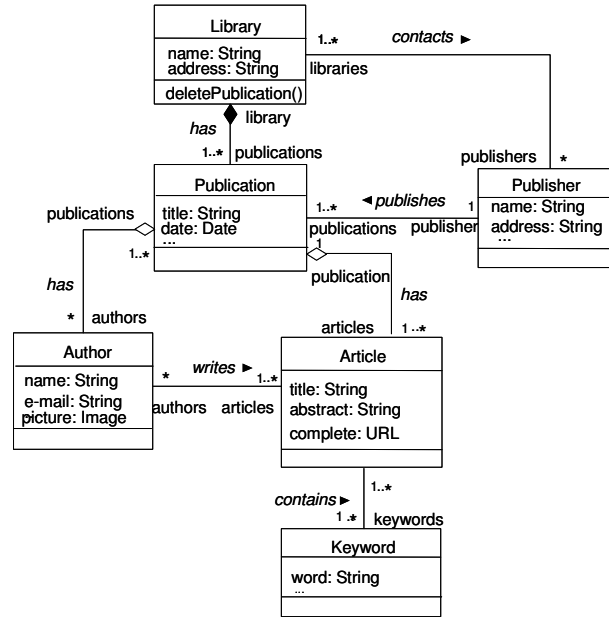
Navigation modeling. Navigation modeling activities comprise the specification of which objects can be visited by navigation through the Web application and how these objects can be reached through access structures. UWE proposes a set of guidelines and semi-automatic mechanisms for modeling the navigation of an application (Hennicker et al., 2000). Figure 3 shows the navigation model for the Online Library application.
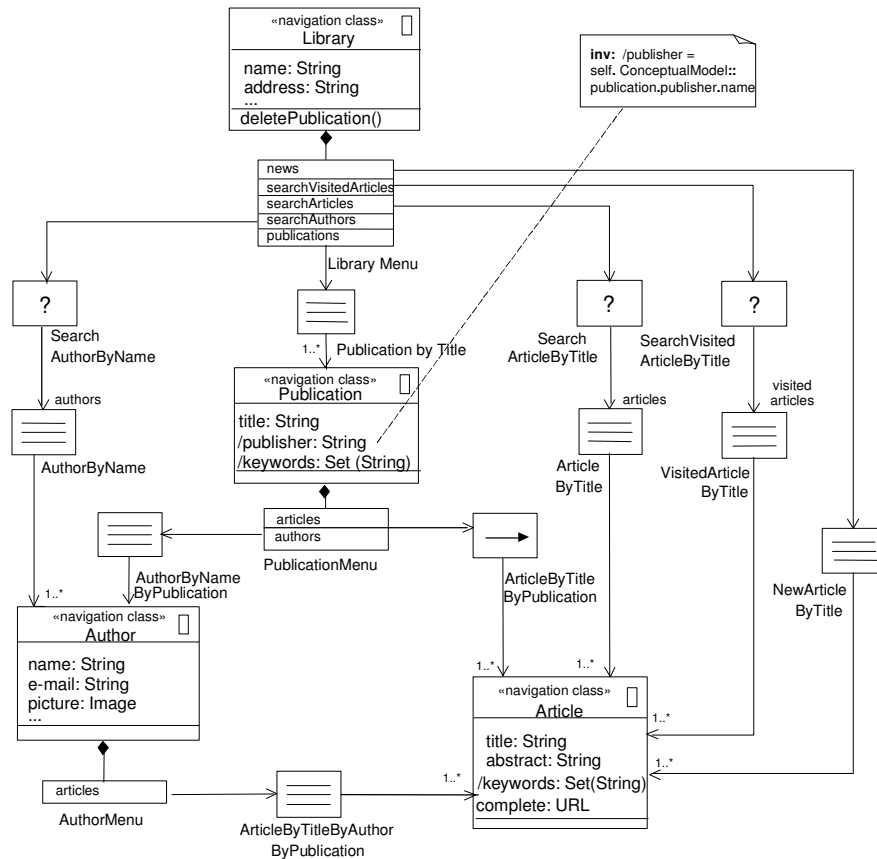
Fig. 2 Conceptual Model of the Online Library Application

Fig. 3 Navigation Model of the Online Library Application

The main modeling elements are the stereotyped class «navigation class» and the stereotyped association «direct navigability». These are the pendant to page (node) and link in the Web terminology.

The access elements defined by UWE are indexes, guided tours, queries and menus. The stereotyped classes for the access elements are «index», «guided tour», «query» and «menu. All modeling elements and their corresponding stereotypes and associated icons are defined in Baumeister et al. (1999).

Note that only those classes of the conceptual model that are relevant for navigation, are included in the navigation model. Although information of the omitted classes may be kept as attributes of other navigation classes (e.g. the newly introduced attribute publisher of the navigation class Publication), OCL Constraints are used to express the relationship between conceptual classes and navigation classes or attributes of navigation classes.

Presentation modeling. The presentation modeling describes where and how navigation objects and access primitives will be presented to the user. Presentation design supports the transformation of the navigation structure model in a set of models that show the static location of the objects visible to the user, i.e. a schematic representation of these objects (sketches of the pages). The production of sketches of this kind is often helpful in early discussions with the customer.

UWE proposes a set of stereotyped modeling elements to describe the abstract user interface, such as «text», «form», «button», «image», «audio», «anchor», «collection» and «anchored collection». The classes «collection» and «anchored collection» provide a convenient representation of frequently used composites. Anchor and form are the basic interactive elements. An anchor is always associated with a link for navigation. Through a form a user interacts with the Web application supplying information and triggering a submission event. (Baumeister et al., 1999). Figure 4 depicts the presentation sketch of a publication.
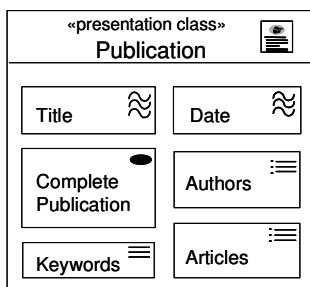

Fig. 4 Sketch of a Publication of the Online Library

Task modeling. To allow automatic generation of a Web application out of a set of models, task design is needed in addition to the already presented design activities. Task modeling builds on the use case model. Different UML notations are proposed for task modeling. Wisdom is an UML extension that proposes the use of a set of stereotyped classes that make the notation not very intuitive (Nunes et al., 2000). Markopoulus (2000, 2002) makes two different proposals: an UML extension of use cases and another one based on statecharts and activity diagrams. Based on the latter, we use the stereotyped UML dependency «refine» between activities and activity diagrams to indicate a finer degree of abstraction. We also choose a vertical distribution from coarse to fine grained activities to represent a task hierarchy similar to the ConcurTaskTrees of Paternó (2000). Figure 5 shows a task model for the *Delete publication* task. The directed dashed lines express the flow of conceptual and presentation objects during task execution. As demonstrated at the end of section five the operation *deletePublication()* of the class *Library* (see Figure 2) is called during task execution.
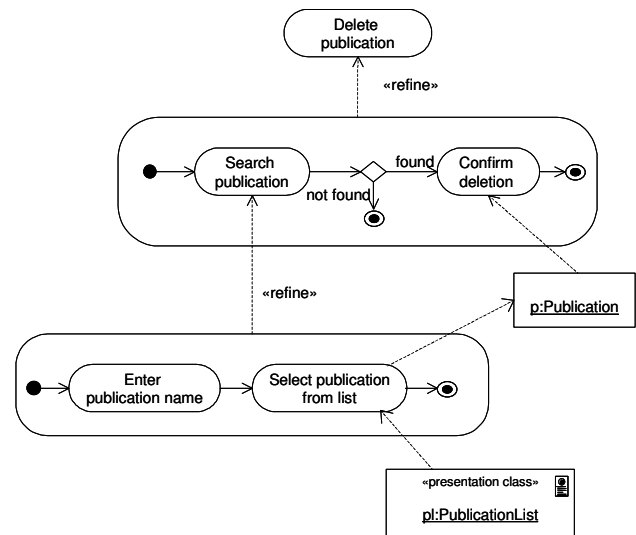

Fig. 5 Task Modeling in the Online Library Application

PRODUCTION SYSTEM ARCHITECTURE

In this section we describe the production system architecture into which the generated Web applications will be deployed. We start by choosing a standard base architecture which we extend by a Web framework. Further we discuss the benefits of using an XML publishing framework. Finally we show how to extend the particular XML publishing framework Apache Cocoon.

Base architecture. Our production system architecture for Web applications is based on an application architecture for the Java 2 Platform Enterprise Edition (J2EE). Figure 6 shows this architecture, where we have only included the details relevant for Web applications.

J2EE has a four tier architecture: Client, Web, Business and Enterprise tier. Web and Business tier together build the J2EE Server tier provided by an application server. The Client tier contains the client side presentation components like a Web browser. Compare to the Client tier in the Thin Web Client architecture as described in Conallen (1999). The Client tier is connected to the Web tier via the HTTP protocol. The Web Container in the Web tier is a container for Java Servlets and Java Server Pages, the answer within the Java Technology to Server Pages. Static pages like HTML pages can be delivered, too, but if you rely on many static pages the combination with a conventional high performance HTTP server would be the better alternative. So the Web tier performs the server side presentation functionality. Business logic exclusively resides within Enterprise Java Beans (EJBs). EJBs are server side components living in an EJB Container within an Application Server. Various complexities inherent in enterprise applications such as transaction management, life-cycle management and resource pooling are handled by the EJB Container, thus reducing the complexity of component development. The Web tier has access to these components which encapsulate business logic, database access and access to legacy systems. For more information about J2EE and it's application architecture see J2EE Architecture.

The J2EE application architecture is a well established standard for enterprise application development, providing us with a powerful component model to which the conceptual model will be mapped to. This also enhances software reusability.

Building on the J2EE architecture there is – besides of technology constraints – no standard method for modeling and building Web user interfaces. To a certain degree separation of concerns is encouraged: business logic should reside in EJBs and Java Server Pages can be used to separate presentation from logic.

This is the extension point for appropriate Web frameworks that improve this situation and establish new standards for building Web user interfaces.

A very promising approach is the Apache Struts framework which helps to build applications with Java Servlets and Java Server Pages based on the Model-View-Controller (MVC) (Gamma, 1995) design paradigm, colloquially known as Model 2. The entry point of Struts within the J2EE architecture is a controller Servlet that dispatches requests to appropriate handler classes. These handlers act as adapters between Controller and Model.

Requests are then forwarded to another handler or directly to a View, i.e. a JSP page.

While Java Server Pages (JSP) technology fulfills the separation of output and logic, separation of content and presentation does not hold. Content and presentation elements are mingled in the same way as in pure HTML, even when using Cascading Stylesheets (CSS).
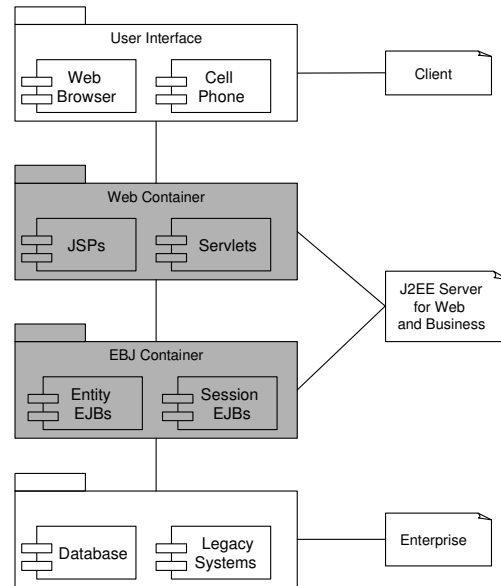

Fig. 6 J2EE Web Application Architecture

Another approach are XML publishing frameworks such as Cocoon (McLaughlin, 2001) which are primarily designed for publishing XML content. Similar to Struts the entry point is a Servlet. The publishing process is done by applying eXtensible stylesheets (XSL) to the XML content thereby transforming it for the presentation for different output media. An XML publishing framework has the inherent quality of strict separation of content and presentation. This is an important requirement when choosing the appropriate Web framework. A Web artist thus can handle presentation independently from the content. Further, for one content unit distinct presentations for different output channels like HTML, WAP pages or PDF must be applicable.

In this paper we propose the use of the XML publishing framework Cocoon within the J2EE architecture for the semiautomatic generation of Web application. For the dynamic aspects of a Web application we have to extend the content production process making use of the extension facilities of Cocoon.

Extension of Cocoon. The original Cocoon publishing system engine works as follows (Apache Cocoon): a Web request is first passed to a *Producer* component which produces a DOM (Document Object Model, see W3C) object from the request parameters that

is then passed to the *Reactor*. Within the Reactor the DOM object is processed by *Processor* components. The order and type of Processors is determined by the processing instructions in the DOM object. Finally the *Formatter* component is formatting the processed DOM object to a physical format, optionally making use of formatting objects (FO).

The framework is to a high degree customizable, own Producer, Processor or Formatter components can just be plugged in. The standard shipped Producer component *ProducerFromFile* is producing DOM objects from XML files mapping the request URL to file names. This is a static process not allowing us to realize our needs for translating dynamic aspects.

So we plug in our own *ControllerProducer* which is an equivalent to the Controller component (a Servlet) in the Struts framework, extending this way the Cocoon framework. This Producer extends the mere Publishing Framework by a runtime layer. It is controlling the presentation flow, i.e. in terms of the MVC pattern producing the View. Figure 7 depicts the flow within the customized Cocoon publishing engine. This View is actually an XML document which is transformed by a nested ProducerFromFile component into a DOM object. Further on this DOM object is processed by the XSP (eXtensible Server Pages) Processor, thereby communicating with the runtime layer (i.e. the Model in terms of the MVC pattern) to fill the View with content. Then the XSLT (eXtensible Stylesheets Language Transformations) Processor is transforming the logical View into a physical View. Finally the Formatter component is formatting the physical View into the physical output format.

SEMIAUTOMATIC GENERATION OF WEB APPLICATIONS

As illustrated in Figure 1 (Process Overview) the design models for a Web application are directly fed into a generator which automatically generates an implementation for further deployment in Cocoon. Implementation and deployment of EJB components is not automatically performed, this has to be done by the component developers and system integrators. Also the final physical presentation of pages for various output channels still has to be performed by the web artist. Nevertheless the generator is generating templates for these activities.

The generator program is expecting a set of XML documents as input describing the design models, and a set of XML documents containing parameters for the generation process. As interface between UML modeling tools and the generator we use the XMI-Format, a standardized XML format to interchange UML models. In a preprocessing step the models described in the XMI format are extracted and transformed into input XML

documents for the main generation process. The advantages of using our own description format for the models instead of using the XMI format directly are

- the generator does not depend on the XMI format, other (even non UML) modeling tools which don't produce XMI format may as well be used,
- the complexity of the model description format is reduced to a complexity supported by the generator thereby easing the following processing steps.
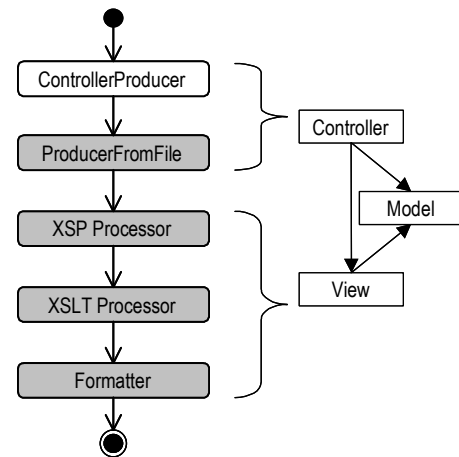

Fig. 7 Flow within the customized Cocoon engine. (Standard Cocoon components have a gray background)

The preprocessing step may as well be plugged into the modeling tool directly, skipping transformations of the model into intermediate formats as XMI.

In the following paragraphs we show for the Online Library example how a XML document of the conceptual model is transformed first to a XML document of the navigation model and then to a XML document of the presentation model. We thereby restrict the XML documents to the *Publication* element. Further we neglect XML namespaces and XML Schema specifications. Each transformation step corresponds to a «trace» relation between the design models.

Conceptual model The conceptual model in our example is described by the following XML document *conceptual-model.xml*:

```
<?xml version="1.0"?>
<conceptual-model ...>
  ...
  <conceptual-class name="Publication">
    <attribute name="title"
               type="java.lang.String"/>
    <attribute name="date"
               type="java.lang.String"/>
    <association name="publisher"
               to="Publisher" mult="1"/>
    <association name="articles"
               to="Article" mult="1..*"/>
```

```
    ...
  </conceptual-class>
  ...
</conceptual-model>
```

Every element of the conceptual model is represented as *conceptual-class* node within the *conceptual-model* root node. It may contain *attribute* and *association* definitions. Attributes have a *name* and a Java *type* parameter, associations are always directed, the *name* parameter is corresponding to the role name of the corresponding UML association end (Parameter *to*). The parameter *mult* is equivalent to the UML multiplicity construct. Attributes and associations always have public visibility and instance scope. Read only behavior can be archived by setting the *read-only* parameter to "true" and key field behavior by setting the *key* parameter to "true".

Now we take a look at a typical conceptual model instance document:

```
<?xml version="1.0"?>
<conceptual-model-instance ...>
  ...
  <Publication cm-id="cm-Publication´-1">
    <title>Java and XML</title>
    <date>2001</date>
    <publisher>
      <link cm-ref="cm-Publisher-1"/>
    </publisher>
    <articles>
      <link cm-ref="cm-Article-1"/>
      <link cm-ref="cm-Article-2"/>
    </articles>
  </Publication>
  ...
</conceptual-model-instance>
```

The structure of the model instance should be self explanatory. Attribute and association *name* parameters in the model description are translated to tag names. These tags are now filled with the content, i.e. with the attribute values of the EJB components. Former associations now have embedded *link* tags, corresponding to the link UML construct in object diagrams. The *cm-ref* parameter references another conceptual model instance element with the same value of the corresponding parameter *cm-id*.

Navigation model. The navigation model in our example is described by the following XML document *navigation-model.xml*:

```
<?xml version="1.0"?>
<navigation-model ...>
  ...
  <navigation-class name="Publication"
       conceptual-class="Publication">
    <attribute name="title" expr="title"/>
    <attribute name="date" expr="date"/>
    <attribute name="publisher"
             expr="publisher.name"/>
    <attribute name="keywords"
        expr="Set(articles.keywords.word)"/>
```

```
    <access-primitive
            name="PublicationsByTitle">
      <index>
        <discriminator expr="title"/>
      </index>
    </access-primitive>
    ...
  </navigation-class>
  ...
</navigation-model>
```

The coarse structure of this document is similar to the conceptual model document. Every *navigation-class* element is related to one (or none) *conceptual-class* element in the conceptual model, corresponding to the «trace» relation between these two design models. The *expr* parameter of the *attribute* tag contains an OCL expression fragment to derive an attribute value from the conceptual instance values (compare with the OCL constraint in Figure 3). Access primitives for navigation elements are specified by the *access-primitive* tag containing the concrete access primitive. In this example an "index" access primitive on the conceptual instances "title" attribute is specified.

Transforming the conceptual model instance produces the following navigation model instance:

```
<?xml version="1.0"?>
<navigation-model-instance ...>
  ...
  <Publication nm-id="nm-Publication-1">
    <title>Java and XML</title>
    <date>2001</date>
    <publisher>O'Reilly</publisher>
    <keywords>
      <word>Java</word>
      <word>XML</word>
      <word>DOM</word>
    </keywords>
  </Publication>
  ...
</navigation-model-instance>
```

Note how links are transformed into derived attributes.

Logical presentation model. The presentation model XML document *presentation-model.xml* is similar to the other two documents:

```
<?xml version="1.0"?>
<presentation-model ...>
  ...
  <presentation-class name="Publication"
        navigation-class="Publication">
    <text name="title" expr="title"/>
    <text name="date" expr="date"/>
    <text name="publisher" expr="publisher"/>
    <collection name="keywords"
              expr="keywords"/>
  </presentation-class>
  ...
</presentation-model>
```

Here the stereotype names utilized in the corresponding presentation design model are used as tag names to specify presentation elements. Again the relation to the navigation model document is established by a navigation in the OCL expression fragment in the *expr* parameter of the presentation elements.

Finally, by transforming the navigation model instance we get the presentation model instance (the structure should be self explicatory):

```
<?xml version="1.0"?>
<presentation-model-instance ...>
  ...
  <Publication pm-id="pm-Publication-1">
    <text name="title">Java and XML</text>
    <text name="date">2001</text>
    <text name="publisher">O'Reilly</text>
    <collection name="keywords">
      <coll-item>Java</coll-item>
      <coll-item>XML</coll-item>
      <coll-item>DOM</coll-item>
    </collection>
  </Publication>
  ...
</presentation-model-instance>
```

Generating the logical presentation documents. So far we showed how to map the conceptual objects to a *conceptual-model-instance* XML document. This document was first transformed to the *navigation-model-instance* document and then, finally to the *presentation-model-instance* document. Now we can generate a presentation document for each presentation class in the presentation model; see the XML Publishing Framework part in Figure 1. This is demonstrated in the following listing showing the presentation document *Publication.xml*. Although this looks rather complicated the basic idea is very simple and explained below.

```
<?xml version="1.0"?>
<?cocoon-process type="xsp"?>
<?cocoon-process type="xinclude"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="Publication.xsl"
                 type="text/xsl"?>

<xsp:page language="java" ...>
  <page>
    <xsp:logic>...</xsp:logic>
    <include xinclude:parse="xml">
      <xsp:attribute name="xinclude:href">
        presentation-model-instance.xml    \
          #xpointer(//Publication[@pm-id='  \
          <xsp:expr>pm_id</xsp:expr>'])
      </xsp:attribute>
    </include>
  </page>
</xsp:page>
```

The document is sequentially processed by three Cocoon processors:

First the eXtensible Server Page (XSP) processor is performing some logic (omitted), i.e. communicating with the runtime layer and determining which Publication presentation object has actually to be presented. The *pm_id* expression inside the *xsp:expr* tag evaluates to the *pm-id* attribute of this presentation object. By using the *xsp:attribute* tag the attribute *xinclude:href* for the next processing step is thereby constructed.

Second then xinclude processor is including the corresponding presentation node from the presentation-model-instance document using an XPointer (see W3C) expression.

Third the XSLT processor is transforming the logical presentation to the physical presentation, see the following sections.

Optimizations. Until now our storage layer for the conceptual objects view consists just of a XML document, or: a DOM object, depending on the implementation. We retrieve the presentation DOM object by applying a sequence of transformations. While this works well as proof of concept, for a production system solution we need some optimizations because every time the conceptual objects (i.e. EJB components) are changing the conceptual DOM object has to be changed. Afterwards the transformation sequence has to be performed again. The situation is even worse because we don't know when and which conceptual objects have changed because we don't want to force a Observer pattern (Gamma, 1995) on every conceptual object just for presentation reasons. In conclusion the conceptual DOM object has to be recreated on every request.

The solution is to perform the sequence of transformations in the model layer thereby generating code for the dynamic lookup of the components attributes and associations. As example consider the following XSP page. This page together with the corresponding stylesheet will automatically be precompiled and cached by the Cocoon engine. Note that the resulting content of this presentation document is the same as without optimization so that the stylesheet introduced in the next paragraph will be the same in both cases.

```
<?xml version="1.0"?>
<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="Publication.xsl"
                 type="text/xsl"?>

<xsp:page language="java" ...>
 <page>
  <xsp:logic>...</xsp:logic>
  <Publication ...>
    <text name="title">
      <xsp:expr>title</xsp:expr>
    </text>
    ...
    <collection name="keywords">
     <xsp:logic>
      for( Iterator i = keywords.iterator();
       i.hasNext(); ) {
       <xsp:content>
```

```
      <coll-item>
       <xsp:expr>
        ((Keyword)i.next()).getWord()
       </xsp:expr>
      </coll-item>
     </xsp:content>
    }
   </xsp:logic>
  </collection>
 </Publication>
 </page>
</xsp:page>
```

Generating stylesheets for the physical presentation. As already mentioned the transformation from the logical to the physical presentation is realized through application of an XSL stylesheet by the XSLT processor within Cocoon. The generator generates a basic stylesheet which has to be adjusted to the desired layout by the Web artist; see the XML Publishing Framework part in Figure 1. The corresponding XML Schema for the logical presentation determines the possible transformations. A deeper introduction in XSL which is based on XPath (see W3C for the specification) is out of scope of this paper. In the following listing we give you an example for a very simple stylesheet *Publication.xsl* for the physical presentation of the Publication element.

```
<?xml version="1.0"?>
<xsl:stylesheet ...>
 <xsl:template match="page">
  <xsl:processing-instruction
   name="cocoon-format" type="text/html"/>
  <html>
   <head><title>Publication</title></head>
   <body>
    <h1>Publication</h1>
    <table>
     <tr>
      <td>Title:</td>
      <td><xsl:value-of
select="Publication/text[@name='title']"</td>
     </tr>
     <tr>
      <td>Date:</td>
      <td><xsl:value-of
select="Publication/text[@name='date']"</td>
     </tr>
     <tr>
      <td>Publisher:</td>
      <td><xsl:value-of
select="Publication/text[@name='publisher']"
      </td>
     </tr>
     <tr>
      <td>Keywords:</td>
      <td>
       <xsl:for-each select="Publication/
          collection[@name='keywords']/
          coll-item">
       <xsl:value-of select="text()"/>
       <xsl:text> </xsl:text>
      </xsl:for-each>
      </td>
     </tr>
    </table>
   </body>
```

```
   </html>
  </xsl:template>
</xsl:stylesheet>
```

Note the processing instruction in this stylesheet which selects the HTML Formatter component for the following formatting process.

Supporting different presentation media. When the Web application has to support different presentation media such as Web browsers or WAP devices you profit by using a XML publishing framework instead of server page technologies like Java Server Pages. Because of the mingling of presentation and presentation logic – in the best case – one has to duplicate pages and modify the contained presentation logic for every presentation media. Using Cocoon there is only one set of these server pages but you can assign different stylesheets for different presentation media including distinguishing between browser types, see the listing below.

```
<?xml version="1.0"?>
...
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="Publication-html.xsl"
                type="text/xsl"?>
<?xml-stylesheet href="Publication-wap.xsl"
                type="text/xsl" media=wap?>
...
```

The set of different presentation media to be supported is supplied as parameter to the generation process. A stylesheet is generated for each presentation media.

Mapping task models. The task model is also mapped to an XML document which is not included here. Within the activity diagrams of task models we allow only activity states and no action states, we call them task activities. Task execution is performed within the runtime layer basing on the one hand on the task description in the XML document and on the other hand on user defined classes for performing activities on conceptual objects.

We sketch some ideas of task execution within the runtime layer:

- Task hierarchy composition and decomposition is done automatically.
- For each session the task execution state is stored.
- The task activity which is the lowest one in the active task hierarchy path is the active task activity which will be executed in one user interaction step.
- When a task activity has an incoming presentation object flow associated to it then this presentation object is displayed. If this presentation object does not contain an input form task execution terminates, otherwise task execution is suspended until the user submits the input form.
- When there is any conceptual object flow the *execute* method of a user task class is invoked. A template for

this class is generated which has to be filled by the task developer. The signature of the method *execute* is determined by the ingoing and outgoing object flow for the corresponding activity in the task model. The corresponding objects are passed as parameters. Such a class for the task activity *Confirm Deletion* in Figure 5 may look like this:

```
public class ConfirmDeletion implements
  TaskActivity
{
  public void execute( Publication p );
  {
    ...
    p.getLibrary().deletePublication( p );
    ...
  }
}
```

CONCLUSIONS AND FUTURE WORK

In this work we showed how to semiautomatically generate implementations for Web applications from UML design models using an XML publishing framework. We first presented the design activities of the UML based Web Engineering approach (UWE) thereby extending it by introducing task modeling which plays an important role within the interactional modeling. Then we presented a standardized, stable and scaleable production system architecture including a component model for use in the generated implementation. We showed how to plug the XML publishing framework Cocoon into this architecture and how to extends Cocoon to fit our needs. Then we demonstrated the generation process.

The next step is to complete the ArgoUML/UWE tool that is currently built to support the semiautomatic transition from design models to a running implementation of Web applications as proposed. There are many open issues, which still need to be addressed in Web engineering. This includes for example task modeling combined with user modeling, semi-automatic implementation of adaptive applications, handling complex presentation structures with windows and framesets. These open issues will be topics for future works.

REFERENCES

ArgoUML. http://www.tigris.org.

Apache Cocoon XML Publishing Framework, http://xml.apache.org/cocoon/index.html.

Apache Struts Project, http://jakarta.apache.org/struts

Baumeister H., Koch N., Mandel L., 1999, "Towards a UML Extension for Hypermedia Design", *Proceedings of The Unified Modeling Language Conference: Beyond the Standard* (UML´1999), France R. and Rumpe B. (Eds), LNCS 1723, Springer Verlag, 614-629.

Conallen J., 1999, "Building Web Applications with UML", Addison Wesley.

Gamma E., Helm R., Johnson R., Vlissides J., 1995, "Design Patterns", Addison Wesley.

Hennicker R., Koch N., 2000, "A UML-based Methodology for Hypermedia Design", *Proceedings of the Unified Modeling Language Conference*, UML´2000, Evans A. and Kent S. (Eds.). LNCS 1939, Springer Verlag, 410-424.

Jacobson I., Booch G., Rumbaugh J., 1999, "The Unified Software Development Process", Addison Wesley.

J2EE, Java 2 Enterprise Edition, http://java.sun.com/ j2ee.

McLaughlin B., 2001, "Java & XML", O'Reilly Publishing Company.

Kamm C., Reine F., Wördehoff H., 2001, "Basisarchitektur E-Business", *Workshop E-Business*, Informatik 2001, Wien.

Koch N., 2001, "Software Engineering for Adaptive Hypermedia Applications", PhD. Thesis, Reihe Softwaretechnik 12, Uni-Druck Publishing Company, Munich.

Koch N., Kraus A., Hennicker R., 2001, "The Authoring Process of the UML-based Web Engineering Approach", First International Workshop on Web-Oriented Software Technology, Valencia/Spain.

Markopoulos P., 2000, "Supporting Interaction Design with UML, Task Modelling" , TUPIS'2000 Workshop at the UML'2000.

Markopoulos P., 2002, "Modelling User Tasks with the Unified Modelling Language", to appear.

Nunes J. N., Cunha J. F., 2000, "Towards a UML Profile for Interaction Design: The Wisdom approach", *Proceedings of the Unified Modeling Language Conference*, UML´2000, Evans A. and Kent S. (Eds.). LNCS 1939, Springer Publishing Company, 100-116.

Paternò F., 2000, "ConcurTaskTrees and UML: how to marry them?", TUPIS'2000 Workshop at the UML'2000.

Rational Rose, http://www.rational.com/products/rose

Schwabe D., 2001, "A Conference Review System.", 1st Workshop on Web-oriented Software Technology, Valencia, to appear and http://www.dsic.upv.es/~west 2001/iwwost01.

Together Control Center, http://www.togethersoft.com

W3C, The World Wide Web Consortium, http://www.w3c.org.