The Expressive Power of UML-based Web Engineering[1]

NORA KOCH AND ANDREAS KRAUS
Ludwig-Maximilians-Universität München. Germany

UML-based Web Engineering (UWE) is a development process for Web applications with focus on systematic design, personalization and semi-automatic generation. UWE describes a systematic design methodology using exclusively UML (Unified Modeling Language) techniques, the UML notation and the UML extension mechanisms. In this article we show the power of this approach. On the one hand, UWE prescribes how to build navigation and presentation models for Web applications defining therefore special UML stereotyped modeling elements and tagged values. On the other hand, we show how these Web specific navigation and presentation models can be supplemented by other views using the variety of UML diagram types and UML modeling elements. Our current research activities focus on the dynamic aspects of the design of Web applications, such as task modeling and modeling of Web scenarios, graphical representation of the distribution of Web components and semi-automatic generation of Web applications based on design models.

Categories and Subject Descriptors: H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia – Architecture; *Navigation; User Issues*; D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Object-oriented Design Methods*; D.2.1 [**Software Engineering**]: Requirements Specification – *Methodologies;* I.6.5 [**Computing Methodologies**]: Simulation and Modeling – *Model Development*

General Terms: Design, Experimentation, Human Factors

Additional Key Words and Phrases: Web Engineering, UML, Visual Modeling, Task Modeling, UML Profile

1. INTRODUCTION

We stress that UML is powerful enough to cover all the requirements that arise when modeling Web applications. For most of these requirements we can use the notation and diagrammatic techniques provided by the UML, i.e. “pure” UML. To cover the special aspects of Web application design, we define in UWE – UML-based Web Engineering – special views graphically represented by UML diagrams, such as the navigation model and the presentation model [Koch, 2001 & Koch et al., 2001]. In addition, a Web designer can also make use of other UML modeling techniques adding other views if necessary, i.e. there is no limitation for the number of views. UML modeling techniques comprise the construction of static and dynamic views of software systems by object and class diagrams, component and deployment diagrams, use case diagrams, state and activity diagrams, sequence and collaboration diagrams. In addition, the UML provides extension mechanisms that we make use of, e.g. we define stereotypes that we utilize in the special views for the Web application modeling. This way we obtain a UML compliant notation – a so called UML “lightweight” extension or UML profile.

[1] This work has been partially supported by the European Union within the IST project AGILE – Architectures for Mobility (IST-2001-32747) and by the German BMBF-project GLOWA-Danube.

Authors' addresses: Nora Koch and Andreas Kraus, Ludwig-Maximilians-Universität München, Germany, www.pst.informatik.uni-muenchen.de, {kochn,krausa}@informatik.uni-muenchen.de

The advantage of using UML diagrams is the known semantics of these diagrams. Furthermore, the notation and the semantics of the modeling elements of "pure" UML, i.e. those modeling elements that comprise the UML metamodel are widely described in the OMG documentation [UML, 2001]. Also it is not difficult for any software designer with UML background to understand a model based on a UML profile, such as the extension that UWE suggests. UML has not only advantages, but also limitations. These limitations are the same for modeling Web applications as for any other kind of software, such as not to support a fine grained component concept and lack of a complete integration of the different modeling techniques. In the new version of UML, that is currently on preparation, some of these limitations will be removed.

All methodologies proposed for Web applications since the middle of the nineties present their own notation and/or diagram types for almost all of their diagrams. An excellent overview are the methods presented at the first International Workshop on Web Oriented Software Technology [Schwabe, 2001] as they are described on the basis of a same case study. Conversely, Conallen [1999] provides a UML extension based on current implementation techniques that is useful as "implementation near" design technique. All these methodologies contribute with important ideas to Web oriented software design. To mention just a few of them, for example the context concept of OOHDM [Rossi et al., 2001], different types of links defined by OO-H [Cachero et al., 2000], graphical and XML representations of concepts by WebML [Ceri et al., 2001] and architecture design of Web applications [Conallen, 1999]. Most of the notations of these methods could easily be translated to UML conform ones as it is shown by Koch [2001] e.g. for the OOHDM contexts.

To show the expressive power of UML this article is structured as follows: First, Section 2 presents an overview of the UWE methodology including analysis, design and generation of Web applications. In Section 3 we give a brief description of the UML techniques that can be used in Web design and focus on the extension mechanisms supported by the UML. Based on an ongoing example Section 4 shows how to use use case models for the requirements specification and Section 5 presents standard class diagrams for conceptual modeling. Section 6 sketches the use of UML stereotyped class diagrams for navigation and presentation modeling. In Section 7 we discuss different ways to represent Web scenarios by UML interaction diagrams and UML statechart diagrams. Section 8 proposes the use of UML activity diagrams for task modeling and Section 9 describes how UML deployment diagrams can be used to document distribution of Web application components. Finally, in the last section some conclusions and future work are outlined.

2. THE UWE METHODOLOGY

The UML-based Web Engineering (UWE) approach presented by Koch [2001] and extended in subsequent papers [Hennicker & Koch, 2001; Kraus & Koch, 2002] supports Web application development with special focus on systematization, personalization and semi-automatic generation. It is an object-oriented, iterative and incremental approach based on the Unified Modeling Language [UML, 2001] and the Unified Software Development Process [Jacobson et al., 1999]. The notation used for design is a "lightweight" UML profile (see Section 3).

The UWE methodology provides guidelines for the systematic and stepwise construction of models which is detailed by Hennicker and Koch [2000], by Koch [2001] with locus on personalization and in the case study presented at the first IWWOST workshop [Koch et al., 2001]. The core modeling activities are the requirements analysis, conceptual, navigation and presentation design supplemented with task and deployment modeling and visualization of Web scenarios. Task models and statecharts of Web scenarios are included to model the dynamic aspects of the application. We also achieve semi-automatic generation of Web applications from design models. An extension of the ArgoUML tool is being implemented to support the construction of the UWE design models. These design models in XMI format will be used for the semi-automatic generation using an XML publishing framework [Kraus & Koch, 2002]. 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.

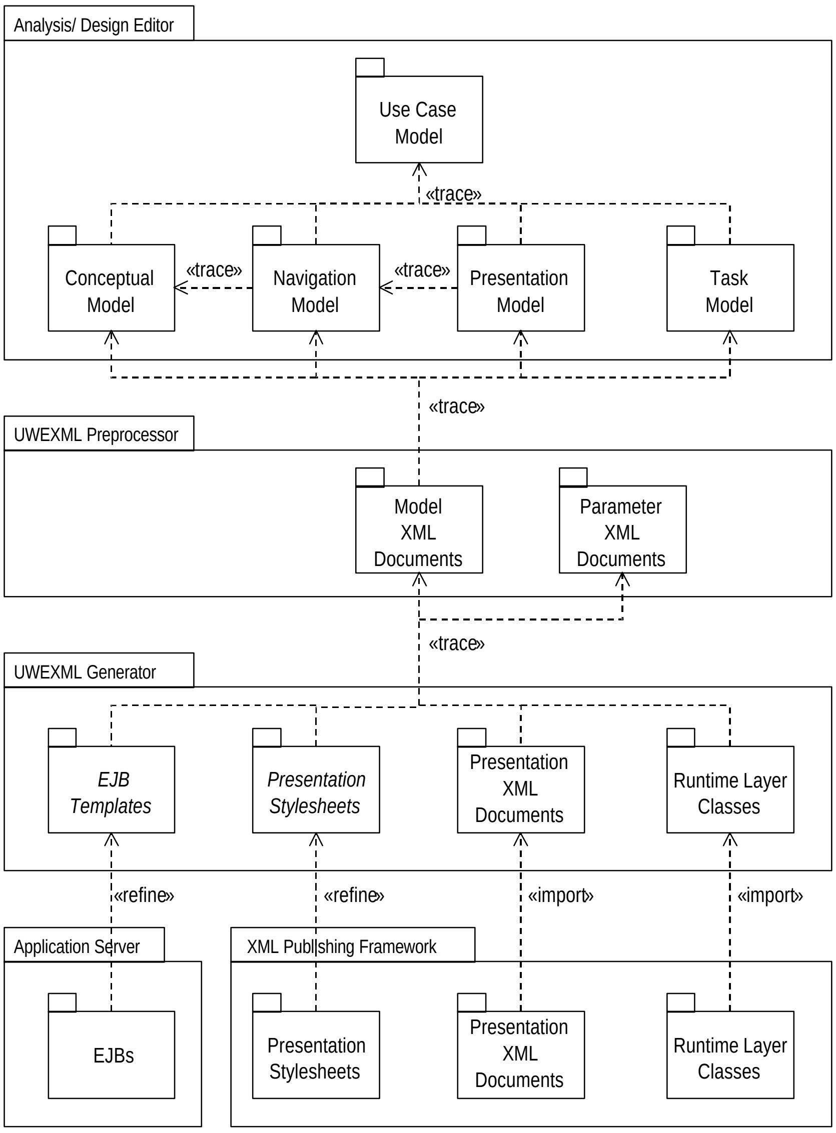


Fig. 1: UWEXML Process Overview

Artifacts within the development process are depicted as UML packages. The «trace» dependencies describe which artifacts are historical ancestors of other artifacts. 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 (Figure 1). In this process we consider deployment to an application server providing a physical component model and to an XML publishing framework.

In this work we show the power and flexibility of UML as diagrammatic technique for Web design and we present the latest development of our current research, i.e. task modeling, dynamic models for Web scenarios and the use of deployment models in the Web application design. Details about the UWEXML approach can be found in [Kraus & Koch, 2002].

3. UML “PURE” AND UML PROFILE

We can solve most of the Web design problems, if we stick to “pure” UML notation and UML diagram types, i.e. without extensions of any type. Special aspects of a domain can instead be addressed by the extension mechanisms provided by the UML itself – a so called UML profile. The notation we use for Web design is a “lightweight” UML profile developed in previous works [Baumeister et al., 1999; Hennicker & Koch, 2000; Koch, 2001]. This profile includes stereotypes defined for the modeling of navigation and presentation aspects of Web applications.

UML provides nine types of diagrams and a set of modeling elements for the diagram types. Syntax and semantics of these modeling elements are defined by the UML meta-model and the well-formedness rules [UML, 2001]. The intended semantics of those diagrams that are relevant for the development of Web applications will be explained in the following sections. In this section a brief introduction to the extension mechanism of UML is given. Stereotypes, tagged values and constraints represent the built-in extension mechanisms of UML. UML can be seen as a family of modeling languages, rather than a single language [Cook, 2000] if we consider these “lightweight” extensions. Lightweight means that it can be easily supported by tools and it does not impact the interchange formats.

3.1 Stereotypes

A UML stereotype is a new kind of model element defined within the model based on an existing kind of model element. Stereotypes may extend the semantics and have additional constraints but they do not provide access to the metamodel of the language. The intent is that a generic modeling tool, such as a model editor should treat a stereotyped element for most purposes as an ordinary modeling element, while differentiating it for certain semantic operations, such as well-formedness checking and code generation. The definition of a stereotype also includes a notation icon.

The use of this powerful mechanism has both advantages and risks. The advantage is to easily create modeling languages for specific application domains with more expressive and precisely defined modeling elements. Examples of domains for which such extensions have already been defined are the real-time [Selic, 2000], the business

[UML, 2001] and the Web domain [Conallen, 1999 & Baumeister et al., 2000]. The risk is the excessive use of stereotypes that can make a language both difficult to handle and to understand.

Some stereotypes only change the notation of a modeling element; they just serve as a kind of comment. Powerful stereotypes, instead, add or redefine semantic restrictions on the metamodel element. UML stereotypes are classified according to their expressiveness into decorative, descriptive, restrictive and redefining stereotypes [Berner et al., 1999). Decorative stereotypes change the concrete syntax and/or visual representation of a language element. Descriptive stereotypes extend the syntax of a language element such that additional information can be expressed. Restrictive stereotypes extend the syntax and impose semantic restrictions. Redefining stereotypes redefine a language element modifying the original semantics of the metamodel element.

In UWE for example, the stereotyped association «direct navigability» has the restriction that both association ends must be connected to stereotyped classes «navigation class» or access primitives (see Figure 4).

3.2 Tagged Values

A UML tagged value is a (tag, value) pair that permits arbitrary information to be attached to any model element. The information is expressed in text form and is commonly used to store non-functional requirements or project management information. The interpretation of the value is a convention between the modeler and the modeling tool.

UWE uses tagged values to model adaptive navigation of personalized Web applications. Adaptive navigation consists in changes to the navigation structure or how this structure is presented to the user. Adaptation is based on the knowledge the applications has about the user. This knowledge is captured in a user profile. Hence, in the UML diagrams to model the adaptive navigation we specify properties to the «direct navigability» association, such as {sorted} or {annotated} to show that the objects of an index will be sorted by preferences of the user for example (see Figure 5).

3.3 Constraints

A UML constraint is a condition or restriction that allows new semantics to be specified linguistically for a model element. The specification is written as an expression in a constraint language. This constraint language may be a formal language, such as in the case of the Object Constraint Language (OCL) provided by the UML, or it may be a natural language. In the latter case, it can not be used for automatic model checking.
As example, we use the following OCL constraint included in the navigation model of the ongoing example to indicate how an derived attribute of the navigation model is related to an attribute of another class of the conceptual model (see Figure 4).

context *Publication*
inv *derived attribute" /publisher" from class publication of the conceptual model*
*/publisher = self**.** ConceptualModel **::** publication.publisher**.**name*

4. USE CASES FOR REQUIREMENTS SPECIFICATION

A use case model can be used to describe the functional requirements of an application in terms of use cases. A UML *use case* is a coherent unit of functionality provided by the application that interacts with one or more outside *actors* of the application. It describes a piece of behavior of the application without revealing the internal structure. We show in this section how requirements for a Web application can be specified with a use case model. UML use case diagrams are built with two main UML modeling elements, namely *use cases* and *actors* and use case relationships between these elements, such as *associations* between an actor and a use case and dependencies «includes» and «extends» between use cases.

As a running example to illustrate the UML techniques used in UWE, the Web site of a personalized online library is presented. This online library application offers information about publications to registered and anonymous users. A publication captures information about journals, books and proceedings. These publications are described by a title, a number, a publisher, a publishing date, a set of articles and authors for each article. Books consists of exactly one article whose title is the same as the book title. In addition, a set of keywords is associated to each article and publication. We distinguish the following types of users of this online library application: unregistered and registered readers and a library administrator. The registered reader is modeled by tracking his interests in articles and registering the articles he visits. This registered reader can also mark articles (bookmarks) as being of special interest. A list of personal keywords for each reader is administrated by the application. The system performs the update of the user model in accordance with the observations on the reader's behavior (in this case limited to the articles he marks or visits frequently). The list can include positive as well as negative keywords. Negative keywords are used to hide irrelevant publications and articles from the reader.

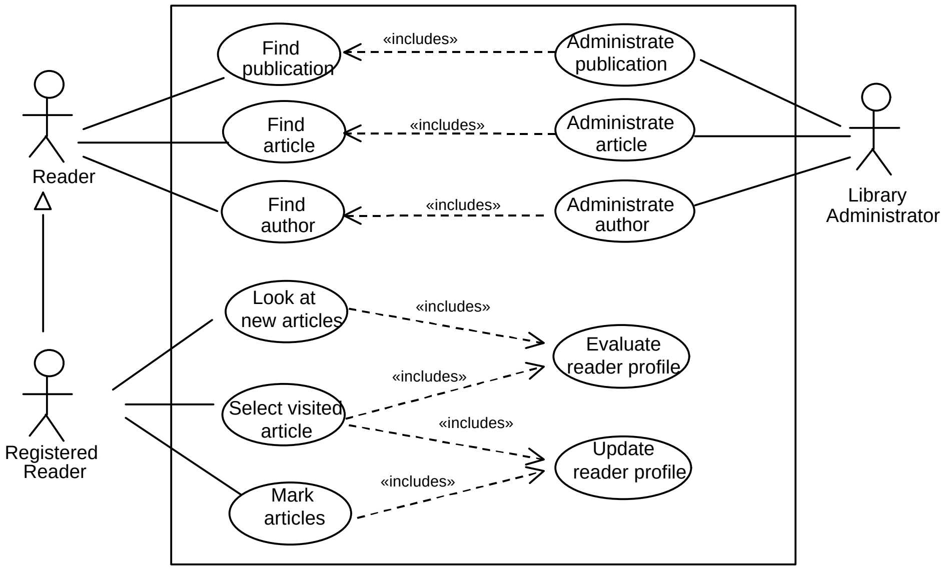


Fig. 2: Use Case Model of the Online Library Application

The use case model of the Online Library Application is partially shown in Figure 2. We can further detail these use cases using a textual form or UML activity diagrams to specify the sequence of actions to be performed by the actors involved in the use cases.

5. CLASS DIAGRAMS FOR CONCEPTUAL MODELING

A UML class diagram is used to graphically represent a conceptual model as a static view that shows a collection of static elements of the domain. Following OOHDM [Rossi et al., 2000] UWE aims to build a conceptual model of a Web application, which attempts to ignore as many of the navigation, presentation and interaction aspects as possible. These aspects are postponed to the navigational and presentational steps of the design.

The main modeling elements used in the conceptual model are: *class* and *association.* However, the power of class diagrams is given by a variety of additional features that can be used to semantically improve these diagrams. Examples of these features are *association* and *role names*, *multiplicities*, different forms of associations supported by the UML like *aggregation*, *inheritance*, *composition* and *association class*, which are represented graphically by the UML notation [2001]. If the conceptual model consists of many classes, it is recommended that they be grouped using the UML *package* modeling element.

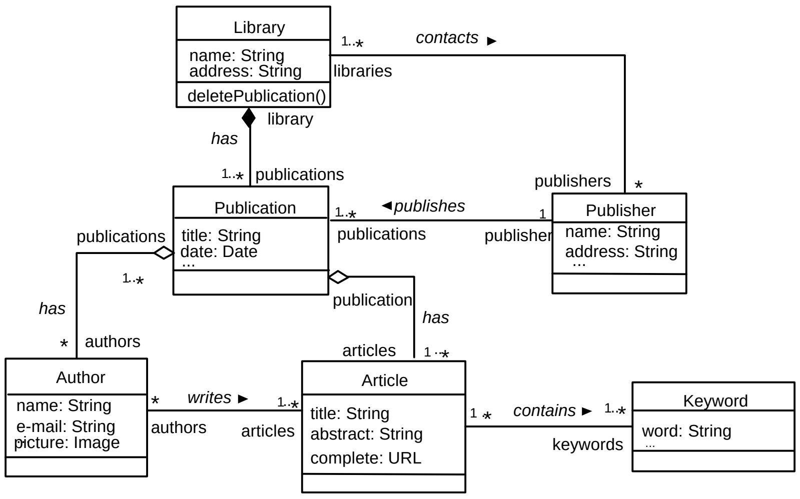


Fig. 3: Conceptual Model of the Online Library Application

Based on the use cases of the previous section and the detailed description of these use cases with activity diagrams a conceptual model is built. The conceptual model includes the objects involved in the typical activities users will perform with the application, i.e. objects that are relevant input for the activity or are the result of the activity. Figure 3 shows the conceptual model of the Online Library example. The example is limited to the core data and functionality, although many other aspects should be included in the Online Library in an incremental and iterative process. These aspects could be additional classes and operations. In addition to the search engines, authoring functions must be incorporated to allow for a visible and changeable user model.

6. STEREOTYPED CLASS DIAGRAMS FOR NAVIGATION AND PRESENTATION MODELING

Navigation and presentation modeling are not exclusive concerns of Web applications, but the navigation and presentation concepts became more important in software development with the advent of the Web. Additionally, separate modeling of conceptual, navigation and presentation aspects of Web applications increases the device independence and the reuse possibilities.

6.1 Navigation Models

Navigation modeling of Web applications comprises the construction of two navigation models, i.e. the navigation space model and the navigation structure model. The former specifies *which* objects can be visited by navigation through the application. It is a model at analysis level. The latter defines *how* these objects are reached. It is a model at design level. The navigation models are represented by stereotyped class diagrams.

The navigation space model includes the classes of those objects which can be visited by navigation through the Web application and the associations which specify which objects can be reached through navigation. UWE provides a set of guidelines and semi-automatic mechanisms for modeling the navigation of an application, which are detailed in previous works [see Hennicker & Koch, 2000; Hennicker & Koch, 2001].

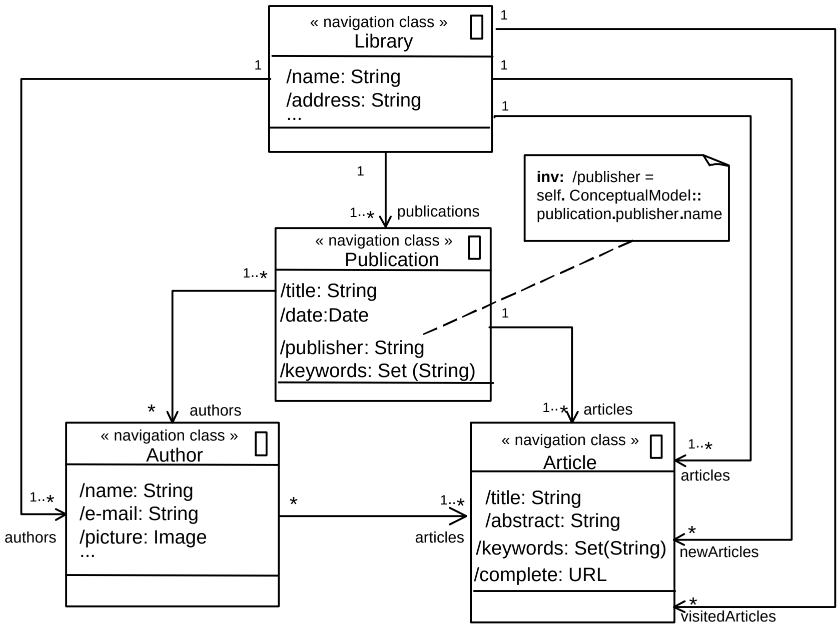


Fig. 4: Navigation Space Model of the Online Library Application

Figure 4 shows the navigation space model for 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. 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.

The navigation structure model is build on the basis of the navigation space model, it can be considered as a refinement step in the UWE design process. UWE provides a set of guidelines and semi-automatic mechanisms for this refinement process [Hennicker & Koch, 2000], which consists in enhancing the navigation space model by indexes, guided tours, queries and menus.

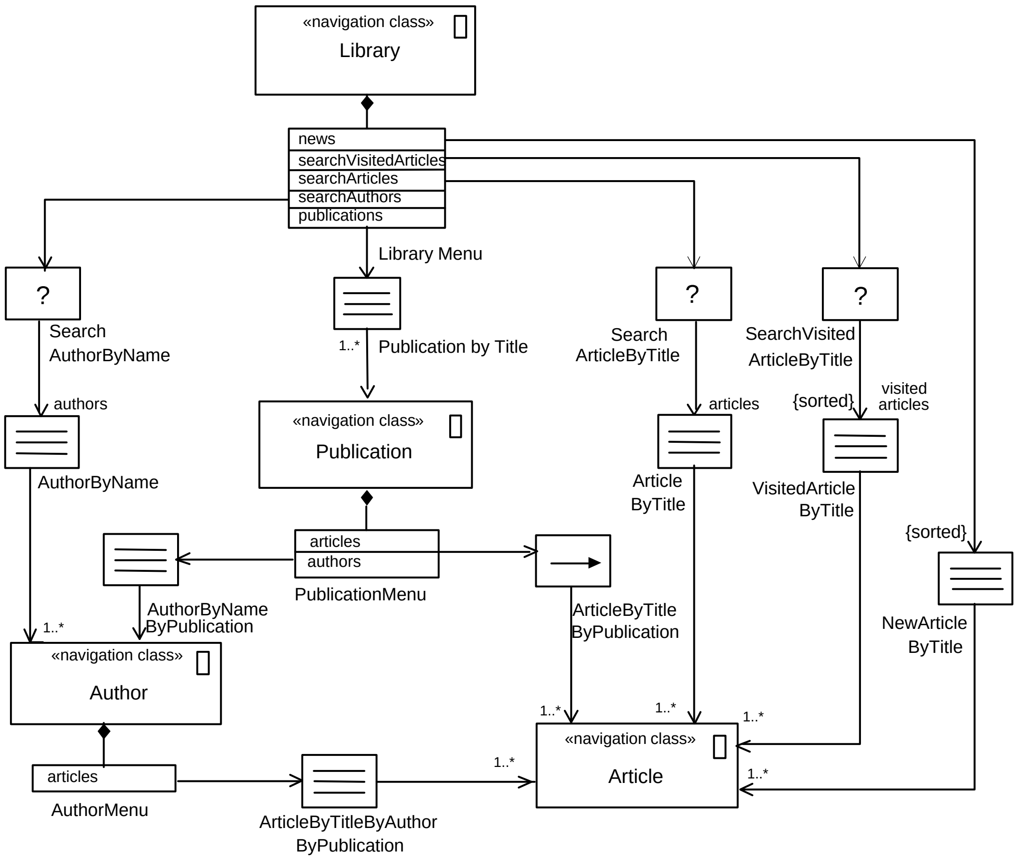


Fig. 5: Navigation Structure Model of the Online Library Application

The stereotyped classes for the access elements are *«index»*, *«guided tour»*, *«query» and «menu»*. The semantics and the icons for these stereotypes stem from Baumeister et al. [1999]. Figure 5 shows the stereotyped class diagram that represent the navigation structure model of the ongoing example. In Figure 5 we can observe the *tagged values {sorted}* that indicate that the corresponding indexes have they index items sorted according to the user's preferences. The system obtains the user's preferences from the current values of the user model. After getting used to the notation this diagram results as very helpful in the static representation of navigation structure.

6.2 Presentation Model

A particular form of a class diagram is used for the presentation model. It is a class diagram using the UML composition notation for classes, i.e. containment represented by graphical nesting of the symbols of the parts within the symbol of the composite. This kind of representation is appropriate for modeling user interfaces as it allows for spatial ordering and relative dimensions although this information cannot be managed by standard case tools.

The presentation model 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). Figure 6 depicts such a presentation sketch for the navigation class of publication. The production of sketches of this kind is often helpful in early discussions with the

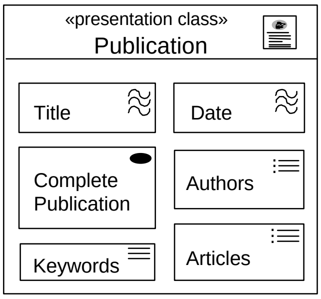


Fig. 6: Publication Sketch of the Online Library Application

customer. These sketches can be linked to produce storyboards. Such a storyboard is not included in this article due to space problems, but is shown for the conference review system example in Koch et al. [2001].

The set of stereotyped modeling elements proposed by UWE to describe the presentation model consists of «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].

7. STATECHART AND INTERACTION DIAGRAMS TO MODEL WEB SCENARIOS

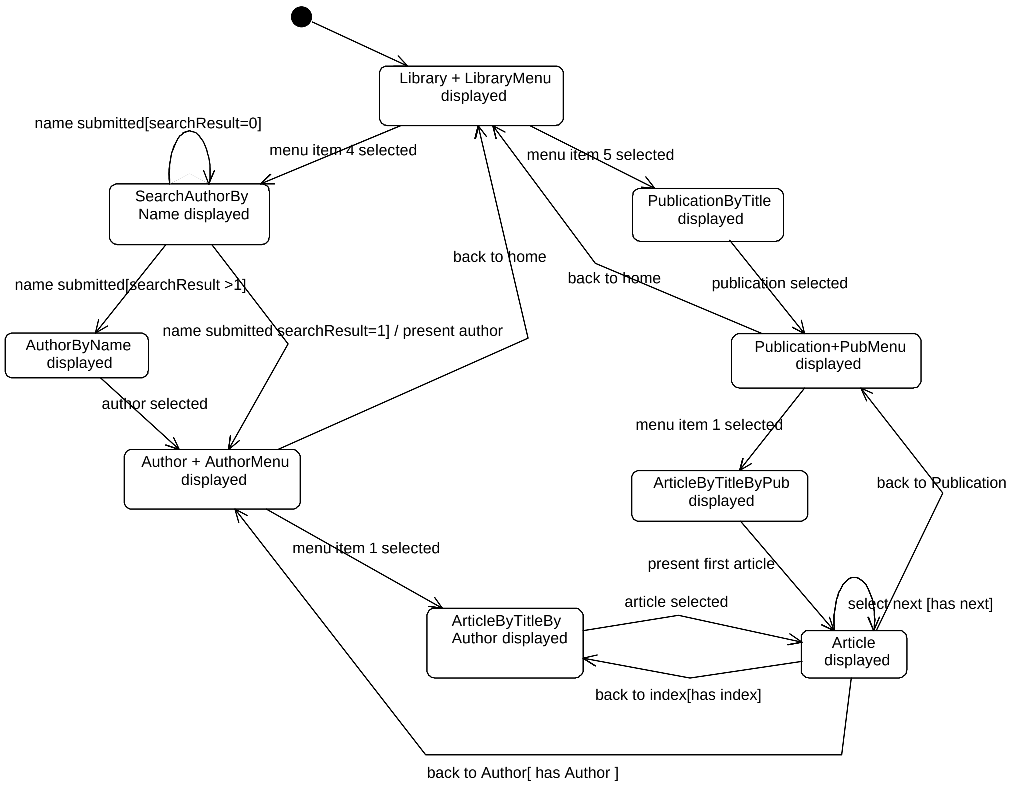
Our current research focuses between others on improving the modeling and visualization of the dynamic aspects of a Web application. We aim to find out which are the more appropriate diagram types for modeling these aspects. UML provides for this purpose statechart diagrams, and interaction diagram, i.e. sequence and collaboration diagrams.

A UML statechart diagram shows a sequence of *states* that an object goes through during its life, together with responsive *actions*, triggering *events* and *guard conditions* associated to *state transitions*. The concept state machine was originally invented by Harel [1987] and is a graph of states, transitions and nested composite states.

In UWE we use statechart diagrams for visualizing navigation scenarios in the same way OO-H [Cachero et al., 2000] does. It allows us to detail parts of the navigation structure model specifying the events that trigger the transitions, defining guard conditions and explicitly including the actions to be performed. Figure 7 depicts a statechart diagram for the user interface of the Online Library application. The states are named after the presentation classes that are actually displayed at the user interface. The fact that we only show a scenario allows us to introduce more specification details, such explicitly determine when back navigation is possible (see Figure 7). In case of a multi-window presentation, UML *synch states* are used to control the synchronization of concurrent regions of the state machines for the different windows.

A UML sequence diagram shows object interaction arranged in temporal order. It presents the objects participating in the interaction and the sequence of messages sent between them. Conallen [1999] uses sequence diagrams for the description of the *use case realization*, i.e. how the use cases are implemented. UWE proposes the use of sequence diagrams the same as other methodologies does to depict presentation flows, i.e. the dynamic aspects of the presentation, such as interactions between windows and frames. Such a sequence diagram is not presented in this article, but is included in the models of the conference review system [Koch et al., 2001].

A UML collaboration diagram shows interaction organized around roles. Collaboration diagrams are equivalent to sequence diagrams, but unlike sequence diagrams, they show the relationship among the roles. Such is the case in the adaptive presentation model presented in Koch [2001].



Library + LibraryMenu displayed
name submitted[searchResult=0]
menu item 4 selected
menu item 5 selected
SearchAuthorBy Name displayed
PublicationByTitle displayed
back to home
back to home
name submitted[searchResult >1]
publication selected
name submitted searchResult=1] / present author
AuthorByName displayed
Publication+PubMenu displayed
author selected
menu item 1 selected
Author + AuthorMenu displayed
ArticleByTitleByPub displayed
back to Publication
menu item 1 selected
present first article
article selected
select next [has next]
ArticleByTitleBy Author displayed
Article displayed
back to index[has index]
back to Author[has Author]

Fig. 7: Statechart Diagram of the Web Presentation

Which type of diagram a designer chooses for the representation of Web scenarios depends on the desired level of granularity, whether it is at analysis or design level and on the objective of the graphical representation, i.e. states and transitions, role-centered or time-centered visualization.

8. ACTIVITY DIAGRAMS FOR TASK MODELING

The concept *task* stems from the Human Computer Interaction (HCI) field [van Harmelen, 2001]: a *task* is composed of one or more subtasks and/or *actions* that a user may perform to achieve a *goal*; a goal represents a desired change in the state of the system and may be realized by formulating a *plan* composed of tasks and then performing those tasks; actions are primitive tasks that have no structure. Here we want to use the concept task in a broader sense by considering tasks performed by the user (user tasks) or by the system (system tasks).

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 & Cunha, 2000]. Markopoulus [2000, 2002] makes two different proposals: an UML extension of use cases and another one based on statecharts and activity diagrams. The use cases of the user can already be considered as tasks at analysis level. Because UML activity diagrams are normally used to further refine use cases we

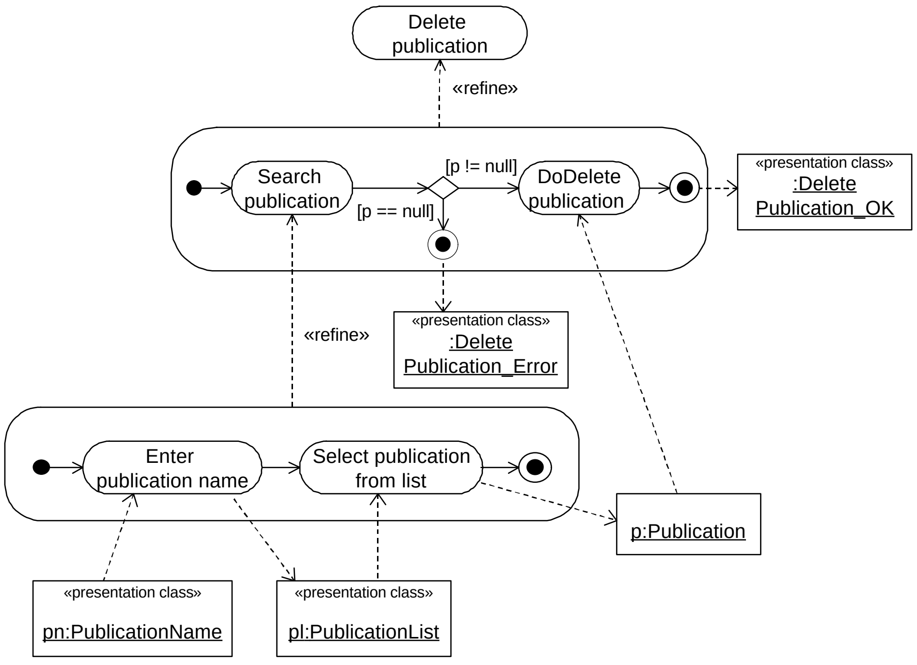

Delete publication
«refine»
Search publication
[p != null]
[p == null]
DoDelete publication
«presentation class» :Delete Publication_OK
«presentation class» :Delete Publication_Error
«refine»
Enter publication name
Select publication from list
p:Publication
«presentation class» pn:PublicationName
«presentation class» pl:PublicationList

Fig. 8: Task Modeling in the Online Library Application

use activity diagrams for task modeling and the modeling elements for task modeling therefor are those for activity diagrams, i.e. *activities, transitions, branche*s, etc. Activity diagrams in general can be considered as "roadmaps" of system functional behavior [Lieberman, 2001]. With our extension of the concept task we may speak of "roadmaps" of user interaction with the system. These "roadmaps" ease the automatic generation of Web applications out of a set of models [Kraus & Koch, 2002].

Within the task model 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]. The temporal order (with branches) between tasks is expressed by transitions between activities. Task modeling in the HCI field also comprises the description of the objects – so called *referents* – the user will perceive. These objects actually are the presentation and conceptual objects which are included in the task model. The relationship between tasks and these objects is expressed as object flow visualized as directed dashed lines. We use ingoing presentation objects to express user input through these presentation objects and outgoing presentation objects to express output to the user through these presentation objects. In addition, we use conceptual objects to express input and output of tasks. The imposed semantic on branch expressions within the task model is that they are boolean (Java) expressions over the named objects of the object flow. Figure 8 shows a task model for the "Delete publication" task.

9. DEPLOYMENT DIAGRAMS TO DOCUMENT THE DISTRIBUTION OF WEB APPLICATION COMPONENTS

We use deployment diagrams to document the distribution of Web application components. The main elements in UML deployment diagrams are *nodes* which are rendered graphically as cubes. A node is a physical element that exists at run time and represents a computational resource [Booch et al., 1999]. A node may contain objects and *components* that reside within this computational resource. A (UML) component is a

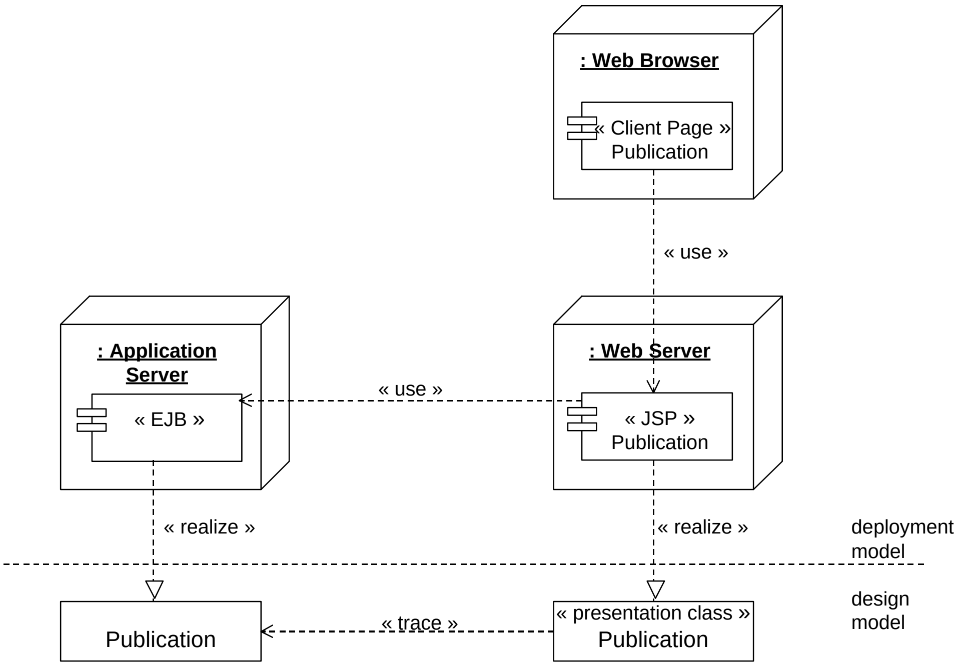


Fig. 9: Deployment Diagram of the Online Library Application with focus on the Publication element

physical and replaceable part of a system that performs to and provides the realization of a set of interfaces and is rendered as rectangle with tabs. Additionally, physical connections between nodes such as for example TCP/IP connections can be modeled.

Figure 9 shows a part of the deployment diagram for our example application. We focus on the Publication element and dependency relations between components. The upper part shows how the components of our example application could be deployed; in the lower part of the figure we added classes of the design model of the application to show by which physical components these classes actually will be realized. The Publication element from the conceptual model will be realized by an Enterprise Java Beans (EJB) component that resides within the Application Server node. The Publication element from the presentation model is realized by a Java Server Pages (JSP) component residing in the Web Server node. It uses the Publication EJB component which is depicted as «use» dependency. Finally within the Web Browser node the client page component for the Publication element is displayed to the user. The term client page stems from Conallen [1999].

10. CONCLUSIONS AND FUTURE WORK

In this paper we showed that UML is powerful enough to cover the requirements that arise when modeling Web applications. For most of these requirements we can use the notation and diagrammatic techniques provided by the UML without extensions. We illustrated how to use the static diagram types to model the static aspects of Web application as well as how to use the dynamic diagram types for the dynamic aspects, such as for example activity diagrams for task modeling. To cover the special aspects of Web application design, we defined special views graphically represented by UML class diagrams, such as the navigation model and the presentation model using a UML “lightweight” extension – a so called UML profile. We use UML interaction and statechart diagrams to visualize Web scenarios. Additionally, we showed how to use the implementation diagram types such as deployment diagrams for the documentation of the deployment of Web application components.

Our future work focuses on the one hand on further refinement of the modeling of the dynamic aspects of Web applications using state diagrams, sequence diagrams and activity diagrams; on the other hand we aim to develop a tool that supports systematic design modeling and allows semi-automatic generation of Web applications. We are working on an extension of the ArgoUML case tool to support the systematic building of the here proposed design models of Web applications. At the same time we are examining the use of an XML publishing framework for the semi-automatic generation of Web applications from design models [Kraus & Koch, 2002].

ACKNODLEDGEGMENTS

Many thanks to Gustavo Rossi and Daniel Schwabe for their valuable feedback and their requests for additional explanations.

REFERENCES

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

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

BERNER S., GLINZ M. and JOOS S. 1999. A Classification of Stereotypes for Object-Oriented Modeling Languages. In *Proceedings of the Unified Modeling Language Conference UML´99*, France R. & Rumpe B., Eds., LNCS 1723, Spriner Verlag, 249-264.

BOOCH G., RUMBAUGH J. and JACOBSON I. 1999, *The Unified Modeling Language User Guide,* Addison Wesley.

CACHERO C., GOMEZ J. and PASTOR O. 2000. Extending an Object-Oriented Conceptual Modelling Approach to Web Application Design. In *Proceedings of CaiSE, LNCS 1789,* Springer Verlag, 79-93.

COOK S. 2000. The UML family: Profiles, Prefaces and Packages (Invited Talk). In *Proceedings of the Unified Modeling Language Conference, UML´2000,* Evans A. and Kent S., Eds.. LNCS 1939, Springer Verlag.

CERI S., FRATERNALI P., MATERA M. and MAURINIO A. 2001. Designing multi-role, collaborative Web sites with WebML: a Conference Management System Case Study. In *1st Workshop on Web-oriented Software Technology,* Valencia, to appear, http://www.dsic.upv.es/~west 2001/iwwost01.

CONALLEN J. 1999. *Building Web Applications with UML,* Addison Wesley.

HAREL D. 1987. *Statecharts: A Visual Formalism for Complex Systems.* Science of Computer Programming, 8(3).

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

HENNICKER R. and KOCH N. 2001. Modeling the User Interface of Web Apllications with UML. In *Practical UML-Based Rigorous Development Methods, Workshop of the pUML-Group at the UML´01*, Gesselschaft für Informatik, Köllen Druck-Verlag.

JACOBSON I., BOOCH G. and RUMBAUGH J. 1999. *The Unified Software Development Process*, Addison Wesley.

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

KOCH N., KRAUS A. and HENNICKER R. 2001. The Authoring Process of the UML-based Web Engineering Approach, In *First International Workshop on Web-Oriented Software Technology IWWOST'2001*, Valencia, to appear.

KRAUS A. and KOCH N. 2001. Generation of Web Applications from UML Models using an XML Publishing Framework, to be published in the *Conference Proceeding of the Integrated Design and Process Technology Conference, IDPT'2002*, Pasadena.

LIEBERMAN B. 2001. UML Activity Diagrams: Versatile Roadmaps for Understanding System Behavior, *Rational Edge Electronic Magazine for the Rational Community.*

MARKOPOULUS P. 2000. Supporting Interaction Design with UML, Task Modelling, In *Proceedings of the TUPIS'2000 Workshop at the UML'2000*.

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

NUNES J. N. and CUNHA J. F. 2000. Towards a UML Profile for Interaction Design: The Wisdom approach, In *Proceedings of the Unified Modeling Language Conference, UML´2000,* Evans A. and Kent S., Eds., LNCS 1939, Springer Verlag, 100-116.

PATERNÒ F. 2000, ConcurTaskTrees and UML: how to marry them?, In *Proceedings of the TUPIS'2000 Workshop at the UML'2000.*

ROSSI G., SCHWABE D., and LYARDET F. 2000. Web Applications Models are More than Conceptual Models. In *Proceedings of the Web Engineering Workshop at WWWCM´99*.

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

SELIC B. 2000. Using UML for Developing Complex Real Time Systems. In *Proceedings of the ECOOP´2000 Conference.*

UML Version 1.4. 2001. *Unified Modeling Language*. The Object Management Group (OMG). http://www.omg.org

VAN HARMELEN M. 2001. Interactive System Design Using Oo&hci Methods, In *Object Modeling and User Interface Design,* van Harmelen M., Ed., Addison Wesley, 365-427.

Received April 2002, revised and accepted May 2002.