Towards a Common Metamodel for the Development of Web Applications[1]

Nora Koch and Andreas Kraus

Ludwig-Maximilians-Universität München,
Oettingenstr. 67, 80538 München, Germany
www.pst.informatik.uni-muenchen.de
{kochn, krausa}@informatik.uni-muenchen.de

Abstract. Many different methodologies for the development of Web applications were proposed in the last ten years. Although most of them define their own notation for building models such as the navigation, the presentation or the personalization model, we argue that in many cases it is just another notation for the same concepts, i.e. they should be based on a common metamodel for the Web application domain. In addition, tool-supported design and generation is becoming essential in the development process of Web applications due to the increasing size and complexity of such applications, and CASE-tools should be built on a precisely specified metamodel of the modeling constructs used in the design activities, providing more flexibility if modeling requirements change.

This paper presents a first step towards such a common metamodel by defining first a metamodel for the UML-based Web Engineering (UWE) approach. The metamodel is defined as a conservative extension of the UML metamodel. We further discuss how to map the UWE metamodel to the UWE modeling constructs (UML profile) of the design method which was already presented in previous works. The metamodel and this mapping are the core of the extension of the ArgoUML open source CASE-tool we developed to support the UWE design notation and method.

1 Introduction

The Web Engineering field is rich in design methods such as OOHDM, OO-H, UWE, W2000, WebML or WSDM [2,5,9] supporting the complex task of designing Web applications. These methodologies propose the construction of different views (i.e. models) which comprises at least a conceptual model, a navigation and a presentation model although naming them differently. Each model is built out of a set of modeling elements, such as nodes and links for the navigation model or image and anchor for the presentation model. In addition, all these methodologies define or choose a notation for the constructs they define.

[1] This research has been partially supported by the EC 5th Framework project AGILE (IST-2001-32747) and by the German BMBF project GLOWA-Danube.

J.M. Cueva Lovelle et al. (Eds.): ICWE 2003, LNCS 2722, pp. 497–506, 2003.

We argue that although all methodologies for the development of Web applications use different notations and propose slightly different development processes they could be based on a common metamodel for the Web application domain. A metamodel is a precise definition of the modeling elements, their relationships and the well-formedness rules needed for creating semantic models. A methodology based on this common metamodel may only use a subset of the constructs provided by the metamodel. The common Web application metamodel should therefore be the unification of the modeling constructs of current Web methodologies allowing for their better comparison and integration.

Metamodeling also plays a fundamental role in CASE-tool construction and is as well the core of automatic code generation. We propose to build the common metamodel on the standardized OMG metamodeling architecture facilitating the construction of meta CASE-tools.

A very interesting approach in terms of metamodeling for Web applications is the metamodel defined for the method W2000 to express the semantics of the design constructs of this method [2]. This metamodel is an extension of the UML metamodel complemented with Schematron rules for model checking. The CADMOS-D design method for web-based educational applications [8] defines another metamodel. It provides a UML visual representation of the modeling elements, but does not establish a relationship to the UML metamodel. Other approaches, such as the Generic Customization Model for Ubiquitous Web Applications [3] or the Munich Reference Model for Adaptive Hypermedia Applications [6], define a reference model for such applications, providing a framework for understanding relationships among entities of those specific Web domains.

As a first step towards a common metamodel we present in this paper a metamodel for the UWE methodology, which could then be joined with metamodels that are/will be defined for other methods. It is defined as a conservative extension of the UML metamodel [10]. This metamodel provides a precise description of the concepts used to model Web applications and their semantics. Our methodology UWE is based on this metamodel including tool support for the design and the semi-automatic generation of Web applications. We further define a mapping from the metamodel to the concrete syntax (i.e. notation) used in UWE. The description of the complete metamodel and the details of the mapping to the UWE notation are not within the scope of this paper (published as a technical report [7]).

The paper is organized as follows: Section 2 gives a brief introduction to the UWE methodology. In Section 3 we propose a metamodel for the UWE methodology. This metamodel is specified in UML. In Section 4 we discuss how the metamodel elements can be mapped to the UWE notation. Finally, some conclusions and future work are outlined in the last section.

2 UWE Methodology

The UWE methodology covers the whole life-cycle of Web application development proposing an object-oriented and iterative approach based on the Unified Software

Development Process [4]. The main focus of the UWE approach is the systematic design followed by a semi-automatic generation of Web applications.

The notation used for design is a "lightweight" UML profile described in previous works, e.g. [5]. A UML profile is a UML extension based on the extension mechanisms defined by the UML itself with the advantage of using a standard notation that can be easily supported by tools and that does not impact the interchange formats. The UWE profile includes stereotypes and tagged values defined for the modeling elements needed to model the different aspects of Web applications, such as navigation, presentation, user, task and adaptation aspects. For each aspect a model is built following the guidelines provided by the UWE methodology for the systematic construction of models. For example, a navigation model is built out of navigation classes, links and a set of indexes, guided tours and queries. The navigation classes and links are views over conceptual classes. Similarly, the user is modeled by a user role, user properties and associations of these properties to the conceptual classes. Currently, an extension of the CASE-tool ArgoUML [1] is being implemented to support the construction of these UWE design models.

In Fig. 1 we give an example for the UWE design models of a Conference Management System application. On the left side the conceptual model is depicted from which in successive steps a navigation model is systematically constructed. On the right side we show the result of the first step in building the navigation model.

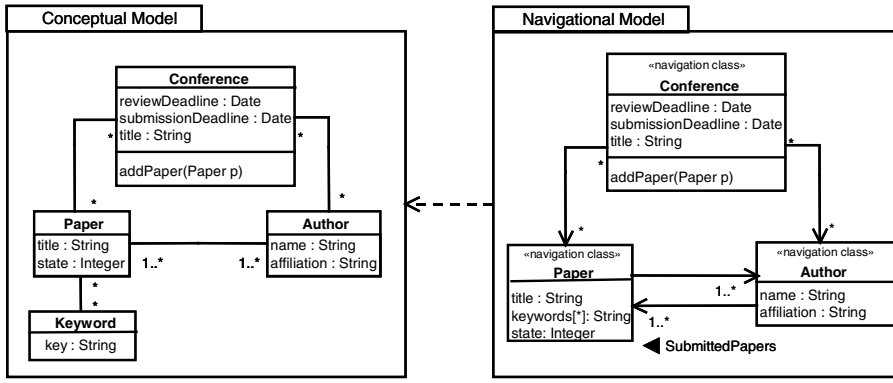

Fig. 1. Example for UWE design models of a Conference Management System

The semi-automatic generation of Web applications from design models is supported by the UWEXML approach [5]. Design models delivered by the design tools in the XMI-Format are transformed into XML documents that are published by an XML publishing framework.

3 UWE Metamodel

The UWE metamodel is designed as a conservative extension of the UML metamodel (version 1.4). Conservative means that the modeling elements of the UML metamodel are not modified e.g. by adding additional features or associations to the modeling

element *Class*. All new modeling elements of the UWE metamodel are related by inheritance to at least one modeling element of the UML metamodel. We define for them additional features and relationships to other metamodel modeling elements and use OCL constraints to specify the additional static semantics (analogous to the well-formedness rules in the UML specification). By staying thereby compatible with the MOF interchange metamodel we can take advantage of metamodeling tools that base on the corresponding XML interchange format XMI.

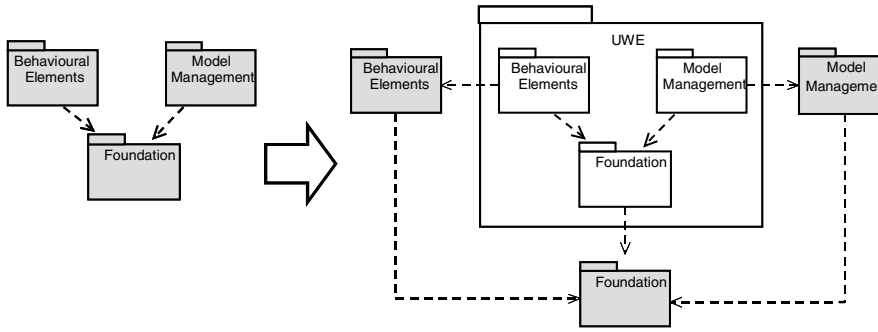

Fig. 2. Embedding of the UWE metamodel within the UML metamodel

In addition, the UWE metamodel is "profileable" [2], which means that it is possible to map the metamodel to a UML profile. Then standard UML CASE-tools with support for UML profiles or the UML extension mechanisms, i.e. stereotypes, tagged values and OCL constraints can be used to create the UWE models of Web applications. If technically possible these CASE-tools can further be extended to support the UWE method. All UWE modeling elements are contained within one top-level package *UWE* which is added to the three UML top-level packages. The structure of the packages inside the UWE package depicted in Fig. 2 is analogous to the UML top-level package structure (shown in gray).

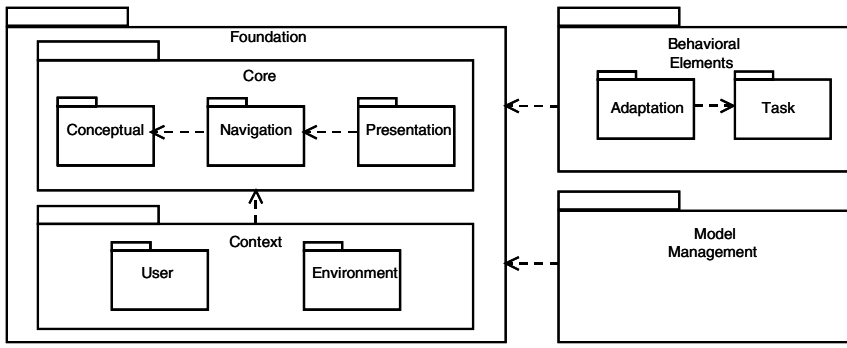

Fig. 3. Package substructure of the UWE metamodel

The package *Foundation* contains all basic static modeling elements, the package *Behavioral Elements* depends from it and contains all elements for behavioral modeling and finally the package *Model Management* which also depends from the

Foundation package contains all elements to describe the models themselves specific to UWE. These UWE packages depend on the corresponding UML top-level packages.

The UWE *Foundation* package is further structured in the *Core* and the *Context* packages (see Fig. 3). The former contains packages for the core (static) modeling elements for the basic aspects of Web applications which are the conceptual, navigation and presentation aspects. The latter depends on the *Core* package and contains further sub-packages for modeling the user and the environment context. The *Behavioral Elements* package consists of the two sub-packages *Task* and *Adaptation* that comprise modeling elements for the workflow and personalization aspects of a Web application respectively. All together one can say that the separation of concerns of Web applications is represented by the package structure of the UWE metamodel.

In the following sections we focus on the *Navigation* and *Presentation* packages of the *Core* package; for description of other packages see [7].

3.1 Navigation Package

The basic elements in navigation models are nodes and links. The corresponding modeling elements in the UWE metamodel are *NavigationNode* and *Link,* which are derived from the UML *Core* elements *Class* and *Association*, respectively. The backbone of the navigation metamodel is shown in Fig. 4. The *NavigationNode* metaclass is abstract which means that only further specialized classes may be instantiated; furthermore it can be designated to be an entry node of the application with the *isLandmark* attribute. The *Link* class is also an abstract class and the *isAutomatic* attribute is used to express that the link should be followed automatically by the system and not by the user. *Links* connect a source *NavigationNode* with one or more target *NavigationNodes* as expressed by the two associations between *Link* and *NavigationNode*. Note that this is an extension to the semantics of links in HTML where only one target is allowed (unless some technical tricks are employed). The associations between *Link* and *NavigationNode* are purely conceptual because we reuse the structure defined in the UML *Core* package where *Classes* are connected to *Associations* via *AssociationEnds*. For further details see the UML specification [10].

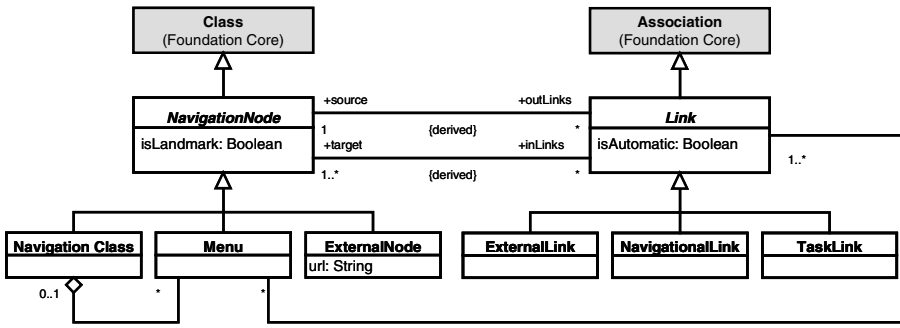

Fig. 4. UWE Navigation package – Backbone

The *NavigationNode* is further specialized to the concrete node types *NavigationClass*, *Menu* and *ExternalNode*. The *NavigationClass* element connects the navigation model with the conceptual model as described in the next paragraph. It may contain a *Menu* that contains *Links* to *NavigationNodes*. We also distinguish the following types of links that are specializations of the class *Link*:

- the *NavigationLink* is used for modeling the (static) navigation with the usual semantics in hypermedia applications and may contain one or more *AccessPrimitives*, such as *Index*, *Query* and *GuidedTour* (these classes are not visualized in Fig. 4 due to space problems).
- the *TaskLink* connects the source node with the definition of a part of its dynamic behavior specified in a UWE task model; and
- the *ExternalLink* links nodes outside the application scope, the so-called *External Nodes*.

Fig. 5 shows the connection between navigation and conceptual objects. A *NavigationClass* is derived from the *ConceptualClass* at the association end with the role name *derivedFrom* – or – one could say that there may exists several navigation views on a conceptual class. The *NavigationClass* consists of *NavigationAttributes* (derived from the UML *Core* element *Attribute*) which themselves are derived from *ConceptualAttributes*. An important invariant is that all *ConceptualAttributes* from which the *NavigationAttributes* of a *NavigationClass* are derived, have to be *ConceptualAttributes* of a *ConceptualClass* in the transitive closure of the *ConceptualClass* from that the *NavigationClass* is derived. This can be formally expressed as an OCL constraint.

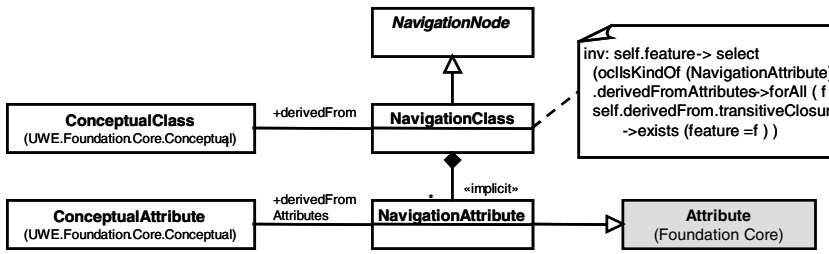

Fig. 5. UWE Navigation package – Connection between navigation and conceptual objects

Further it is possible to specify what types of access primitives should be used for navigation links with a target multiplicity greater than one. For more details see [7].

3.2 Presentation Package

The central element for structuring the presentation space is the abstract class *Location* (see Fig. 6). The presentation sub-structure is modeled with the specialized class *LocationGroup* that consists of a list of sub-locations whereas presentation alternatives between different *Locations* are modeled with the class

LocationAlternative; optionally a default alternative can be specified. Finally, the "atomic" subclass *PresentationClass* contains all the logical user interface (UI) elements presented to the user of the application. It is derived from exactly one *NavigationNode*. The user interface elements are for example *Image*, *Text* or UI group elements, such as *Collection*, *Anchor* and *Form*. How these elements are related to *Link* and *Index* can be seen in the complete description of this package [7]. Further we use a ternary association for expressing link-sensitive presentation, i.e. when following a link from one *NavigationNode* to another we can specify the *PresentationClass* which should be presented to the user depending on the link chosen.

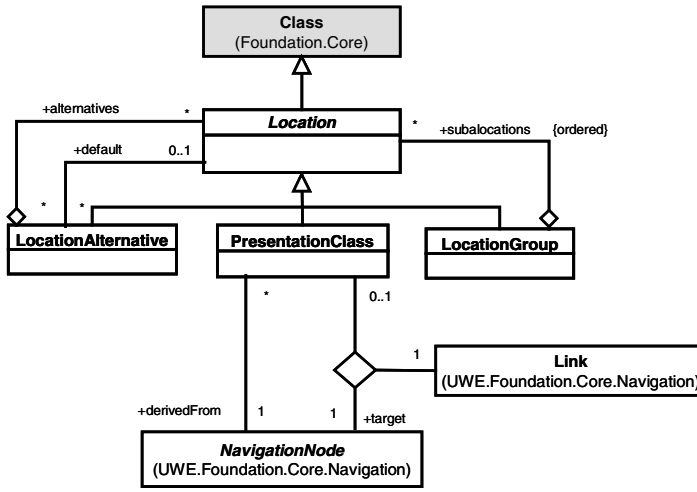

Fig. 6. UWE Presentation Package – Backbone

4 Mapping to the UWE Notation

Metamodels define the concepts and their relationships used in the modeling activities of a certain domain – Web Design in our case – whereas designers build application models using a concrete notation, i.e. the concrete syntax.

One way of mapping a metamodel to a concrete syntax often found in literature is to extend the UML syntax in a non-standard way. This means for example that instead of using the built-in extension mechanism of the UML new graphical symbols are introduced or existing symbols are decorated or its shapes are changed. This could technically be easily achieved e.g. using ArgoUML [1]; by using the NSUML Java framework one can make ArgoUML work with the extended UML metamodel and customize the graphical appearance of all modeling elements. The drawback of this approach is on the one hand that the syntax and semantic of the new notation has to be documented thoroughly. On the other hand the corresponding metamodel interchange format is no longer the same as the UML interchange format. The consequence is that one can no longer use tools which rely on the UML XMI format.

We chose to map the metamodel concepts to a UML profile. A UML profile comprises the definition of stereotypes and tagged values and specifies how they can be used by OCL constraints (i.e. well-formedness of a model). With appropriate tool support a model can be automatically checked if it is conform to the profile. The definition of a UML profile has the advantage of being supported by nearly every UML CASE-tool either automatically, by a tool plug-in or passively when the model is saved and then checked by an external tool.

A simplified version of the mapping rules is the following:

- Metamodel classes (e.g. *NavigationClass*) are mapped to stereotyped classes. The name of the class is mapped to the name of the stereotype and the inheritance structure is mapped to a corresponding inheritance structure between stereotypes.
- Attributes in the metamodel (e.g. the *isAutomatic* attribute of *Link*) are mapped directly to tagged values of the owner class with the corresponding name and type.
- Associations are mapped to tagged values or associations. Mapping to associations is only possible if both classes connected to the association ends are a subtype of *Classifier*, which means that they have a class-like notation. This is for example true for the aggregation between *Location* and *LocationGroup* in the presentation package. On the other hand we can always map associations to tagged values with the drawback of worse readability in the diagrams, e.g. the association between *NavigationClass* and *ConceptualClass*. In the case of binary associations we assign a tagged value to the corresponding stereotyped class of each association end.

We propose to resolve inheritance in the metamodel by repeating the mapping of attributes and associations for all subclasses, e.g. the *isLandmark* attribute of the abstract class *NavigationNode* which is also mapped for the subclass *NavigationClass*.

In the following sections we present the notation for some of the UWE models using the UWE UML profile. For more details about the mapping process refer to [7].

4.1 UML Profile for the Navigation Model

We use the simplified example of a conference management system presented in Fig. 1 to illustrate the mapping process and the notation of the UWE profile for the navigation model. The central element in the metamodel *NavigationClass* is mapped to the stereotype «*navigation class*» (see Fig. 5 and Fig. 7). The metaattribute *isLandmark* indicating that the *Conference* model element is an entry point is represented as a corresponding tagged value of the model element. Another tagged value *derivedFrom* is a mapping of the metaassociation between *NavigationClass* and *ConceptualClass*. As shown in the example for each model attribute the relation to the attributes of the conceptual model is specified by the *derivedFromAttributes* tagged value. The *keywords* attribute of the class *Paper* is a non-trivial example of this relationship, hence the *derivedFromAttributes* tagged value states that this attribute is related to the *key* attribute of the *Keyword* class in the conceptual model associated to the *Paper* class in the conceptual model.

As the metaclass *Link* is a subclass of the UML metaclass *Association* it is also visualized like a UML association. We decorate links with a stereotype such as for example *«navigation link»*. Each link must have an explicit direction and

multiplicities defined. For better readability the stereotype for links may be hidden when the context is clear.

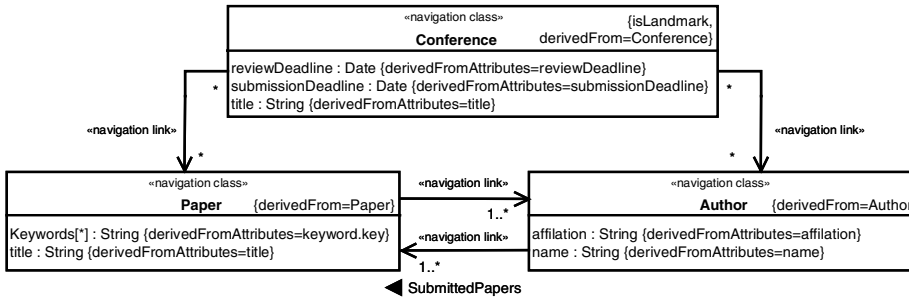

Fig. 7. Example for a navigation model using the UWE UML profile

4.2 UML Profile for the Presentation Model

The three specializations of the abstract class *Location* (see Fig. 6) are mapped to the corresponding stereotypes for the class elements *«location alternative»*, *«location group»* and *«presentation class»*. The presentation grouping expressed by the aggregation association of the *LocationGroup* element is mapped to aggregation associations of the *«location group»* classes where the aggregation is ordered and the association ends have classifier scope and multiplicity one. *LocationAlternatives* are mapped in a similar way, only that we express the default alternative by a tagged value.

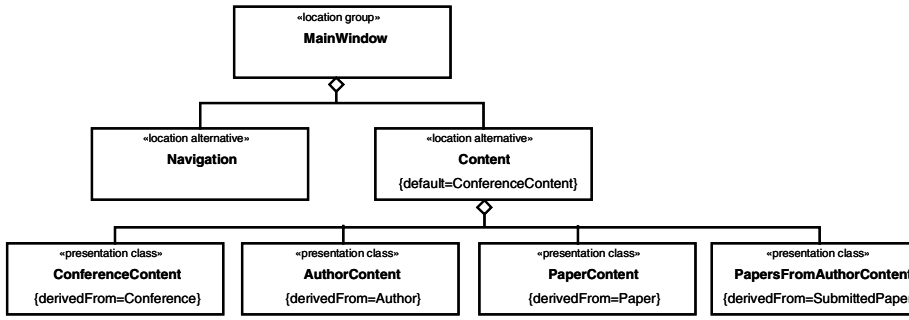

Fig. 8. Example for a presentation model using the UWE UML profile

The relationship between *PresentationClasses*, *NavigationNodes* and *Links* is expressed by one tagged value of the *«presentation class»* element with the name *derivedFrom*. The value has to be the full qualified name of the corresponding *NavigationNode* for entry presentation classes corresponding to entry navigation nodes (i.e. *isLandmark=true*) or for not-link-sensitive presentation classes. In the case of a link- sensitive presentation the name of the corresponding *Link* is assigned to the tagged value. In Fig. 8 we give an example for a presentation model of the conference application example. The location group *MainWindow* divides the

presentation space into the *Navigation* and the *Content* location alternatives. The possible alternatives are the presentation classes *ConferenceContent* (which is the default one), *AuthorContent* and *PaperContent*. For the latter we added a link-sensitive presentation class *PaperFromAuthorContent* which is presented when the link *SubmittedPapers* is used to navigate to the *Paper* node. This is expressed by the *derivedFrom* tagged value. As in the description of the metamodel we omit further details about mapping the user interface part of the metamodel. Here we only want to mention that the user interface elements (e.g. button, text or image) are aggregated to the *«presentation class»* elements.

5 Conclusions

In this paper we presented a metamodel for the UWE methodology and sketched the mapping to a concrete syntax (i.e. notation), the UWE notation defined as a UML profile. The UWE metamodel is defined as a conservative extension of the UML metamodel. This metamodel is the basis for a common metamodel for the Web application domain and for the CASE-Tool supported design.

In our future work we will concentrate on the further refinement of the UWE metamodel to cope with the needs for automatic code generation, especially for the dynamic aspects like tasks and adaptation. At the same time we will extend our tools: on the one hand we have to adapt the CASE-tool ArgoUWE to easily cope with a evolving metamodel and on the other hand our tool for the semi-automatic generation of Web applications UWEXML [5] has to be extended.

References

1. ArgoUML. www.tigris.org
2. Baresi L., Garzotto F., Paolini P. Meta-modeling Techniques meets Web Application Design Tools. Proc. of FASE 2002, LNCS 2306, Springer Verlag, pp. 294–307, 2002.
3. Finkelstein A., Savigni A., Kappel G., Retschitzegger W., Pöll B., Kimmerstorfer E., Schwinger W., Hofer T., Feichtner C., "Ubiquitous Web Application Development - A Framework for Understanding", Proc. of SCI2002, July 2002.
4. Jacobson I., Booch G., Rumbaugh J. The Unified Software Development Process. Addison Wesley, 1999.
5. Koch N., Kraus A. The expressive Power of UML-based Web Engineering. Proc. of IWWOST´02, CYTED, pp. 105–119, 2002.
6. Koch N., Wirsing M. The Munich Reference Model for Adaptive Hypermedia Applications. Proc. of AH´2002, LNCS 2347, Springer Verlag, pp 213–222, 2002.
7. Kraus A., Koch N. A Metamodel for UWE. Technical Report 0301, University of Munich, www.pst.informatik.uni-muenchen.de/publications/TR0301_UWE.pdf, 2003.
8. Retalis S., Papasalourus A., Skordalakis M. Towards a generic conceptual design meta-model for web-based educational applications. Proc. of IWWOST´02, CYTED, 2002.
9. Schwabe D., Pastor O. (Eds.). Online Proc. of IWWOST´01. www.dsic.upv.es /~west2001/iwwost01
10. UML, The Unified Modeling Language, Version 1.4. Object Management Group (OMG). www.omg.org, 2001.