A Metamodel for UWE[1]

Andreas Kraus, Nora Koch

Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstr. 67, D-80538 München
{krausa,kochn}@informatik.uni-muenchen.de

1 Introduction

The Web Engineering field is rich in design methods, such as OOHDM, OO-H, UWE, W2000, WebML or WSDM (Baresi et al., 2002; Koch and Kraus, 2002; Schwabe and Pastor, 2001) 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.

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 meta-model 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 be therefore 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 also 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.

[1] Technical Report 0301, Institut für Informatik, Ludwig-Maximilians-Universität München, January 2003.

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 (Baresi et al., 2002). 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 (Retalis et al., 2002) 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 (Finkelstein et al., 2002) or the Munich Reference Model for Adaptive Hypermedia Applications (Koch and Wirsing, 2002) 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 (i.e. abstract syntax) 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 (UML, 2001). 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 for the semi-automatic generation of Web applications. We further define a mapping from the metamodel to the notation (i.e. concrete syntax) used in UWE.

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. 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 (Jacobson et al., 1999). 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. (Koch and Kraus, 2002). 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 (ArgoUML) is being implemented to support the construction of these UWE design models.

In Figure 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.

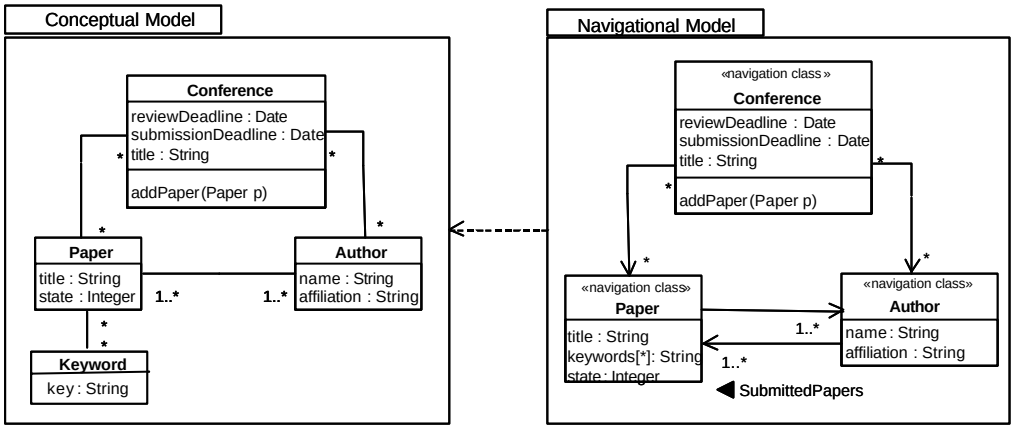

Figure 1. Example for UWE design models of a Conference Management System

The semi-automatic generation of Web applications from design models is supported by the UWEXML approach (Kraus and Koch, 2002). 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.

In addition, the UWE metamodel is "profileable" (Baresi et al., 2002), 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.

By sticking to the actual UML version we also have to deal with some of the problems of its specification. The metamodeling architecture defined by the OMG in which the UML metamodel is embedded is for example not a strict multi-level-metamodeling architecture: a modeling element at the metamodel level i is not instance of exactly one element at the i+1 level. The UML (M2) metamodel contains for example the modeling elements Class and Instance. This problem is also called the "Loose metamodeling problem" (Atkinson, 2001) and will be hopefully solved in a forthcoming version of the UML.

3.1 Package Structure

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 Figure 2 is analogous to the UML top-level package structure (shown in gray). 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.

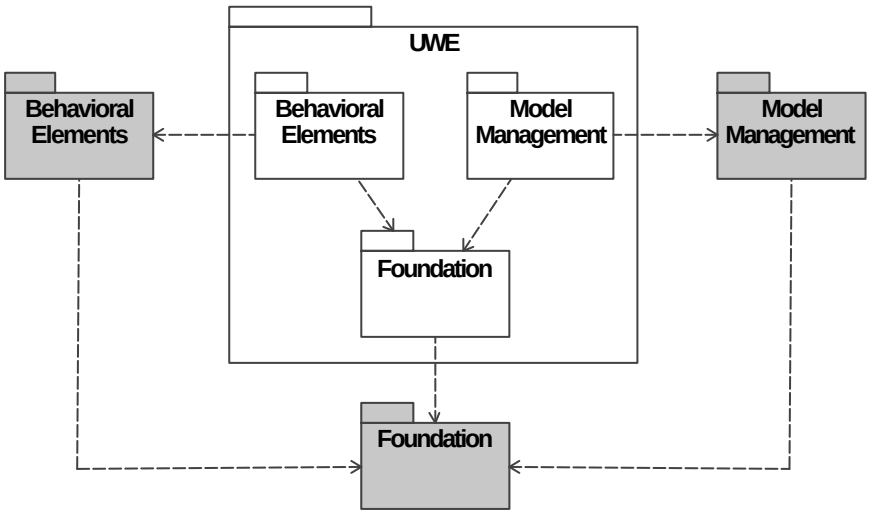

Figure 2. UWE top-level packages.
(The UML metamodel elements are depicted in gray)

The UWE Foundation package is further structured in the Core and the Context packages (see Figure 3). The former contains packages for the core (static) modeling elements for the basic aspects of Web applications which are the conceptual, the navigation and the 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.

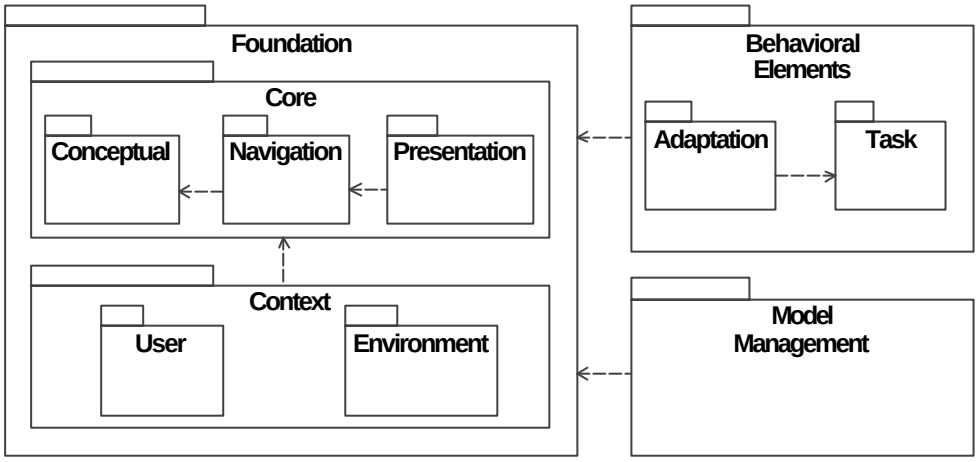

Figure 3. Package substructure of the UWE metamodel

3.2 Conceptual Package

The following sections describe the modeling elements and the well-formedness rules of the conceptual package.

3.2.1 Abstract Syntax

Conceptual modeling for Web applications within UWE does not differ from conceptual modeling for regular applications. But for the reason of a conservative extension described at the beginning of this section for all standard static UML modeling elements for which we want to define associations to other elements of the UWE metamodel must first be specialized to a corresponding UWE conceptual modeling element. So we introduce for example a new class ConceptualClass which is inherited from the UML element Class but has no additional features. We do the same for the Attribute, Operation and Association elements as it is shown

in Figure 4. OCL constraints are defined to assure that a conceptual model is built only with the new defined classes for classes, attributes and operations.

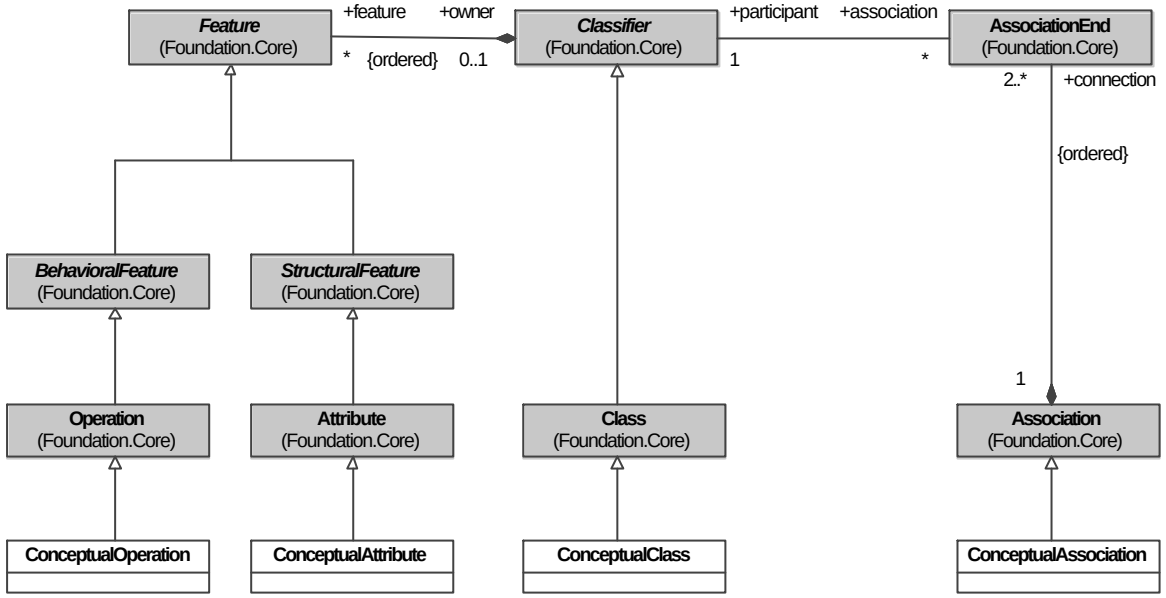

Figure 4. UWE Conceptual Package

3.2.2 Well-Formedness Rules

The following well-formedness rules apply to the conceptual package.

1. A ConceptualClass can only have associations to ConceptualAssociations and can only have ConceptualOperation or ConceptualAttribute features.

   context ConceptualClass
   inv: self.feature->forAll(oclIsKindOf(ConceptualOperation) or
   f.oclIsKindOf(ConceptualAttribute))
   inv: self.association->forAll(association.oclIsKindOf(ConceptualAssociation))

2. A ConceptualAssociation must have two association ends and may be used only for ConceptualClass classifiers.

   context ConceptualAssociation
   inv: self.connection->size() = 2
   inv: self.connection.participant->forAll(oclIsKindOf(ConceptualClass))

3. The ConceptualAttribute and ConceptualOperation features can only be owned by ConceptualClass classifiers.

   context ConceptualAttribute
   inv: self.owner->notEmpty() implies self.owner.oclIsKindOf(ConceptualClass)

```
context     ConceptualOperation
inv:        self.owner->notEmpty() implies self.owner.oclIsKindOf(ConceptualClass)
```

Additional Operations

1. The operation transitiveClosure results in a set containing the transitive closure of a ConceptualClass respective to associations.

```
context     ConceptualClass
def:        transitiveClosure : Set(ConceptualClass) = Set{ self }->union(
                self.association.association.connection.participant.transitiveClosure )
```

3.3 Navigation Package

The following sections describe the abstract syntax and the well-formedness rules of the navigation package.

3.3.1 Abstract Syntax

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 Figure 5. The NavigationNode meta-class 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 (UML, 2001).

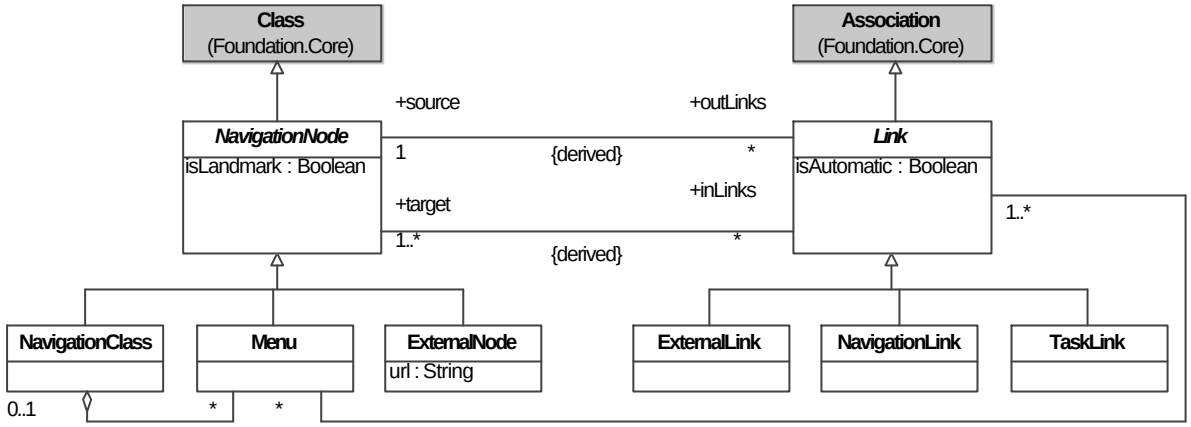

Figure 5. UWE Navigation - Backbone

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 have a Menu that contains Links to NavigationNodes.

Figure 6 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 can exists several navigation views on a conceptual class. The NavigationClass consists of NavigationAttributes (derived from the UML Core element Attribute) which are themselves 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 listed in the following section.

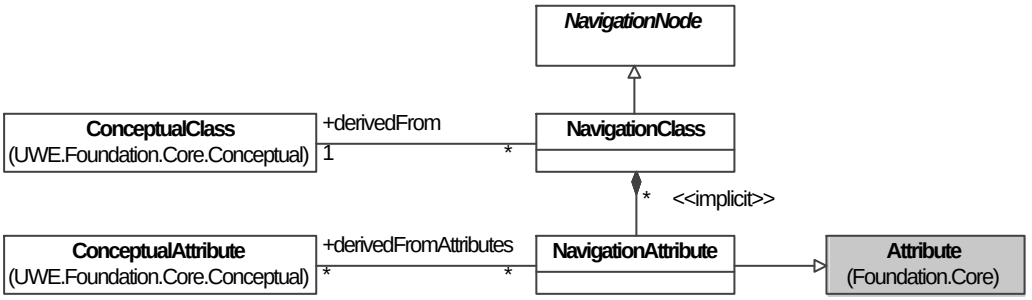

Figure 6. UWE Navigation Package – Connection between navigation and conceptual objects

We distinguish the following types of links that are specializations of the class Link as shown in Figure 7:

- NavigationLink is used for modeling the (static) navigation with the usual semantics in hypermedia applications. Additionally we can specify a sequence of one or more

AccessPrimitives, such as Index, Query and GuidedTour. Each one is associated to one or more NavigationAttributes;

- TaskLink connects the source node with the definition of a part of its dynamic behavior specified in a UWE task model, a TaskGraph; and
- ExternalLink links nodes outside the application scope, so called ExternalNodes.

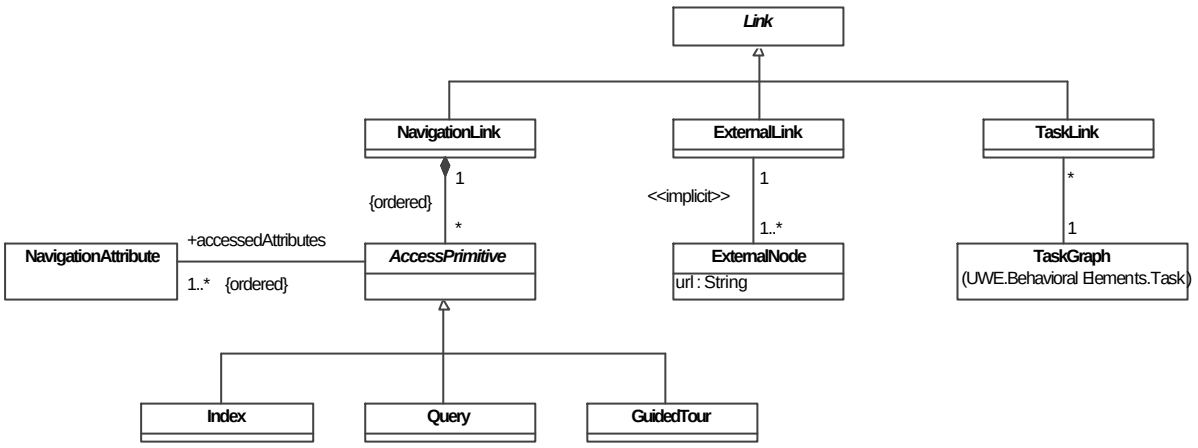

Figure 7. UWE Navigation – Specialized link types and access primitives

3.3.2 Well-Formedness Rules

The following well-formedness rule apply to the navigational package.

1. 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.

```
context     NavigationClass
inv:        self.feature->select(oclIsKindOf(NavigationAttribute)).derivedFromAttributes->
                forAll( f | self.derivedFrom.transitiveClosure->exists( feature = f ) )
```

3.4 Presentation Package

The following sections describe the abstract syntax and the well-formedness rules of the presentation package.

3.4.1 Abstract Syntax

The central element for structuring the presentation space is the abstract class Location (see Figure 8). 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. 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 that should be presented to the user depending on the link chosen.

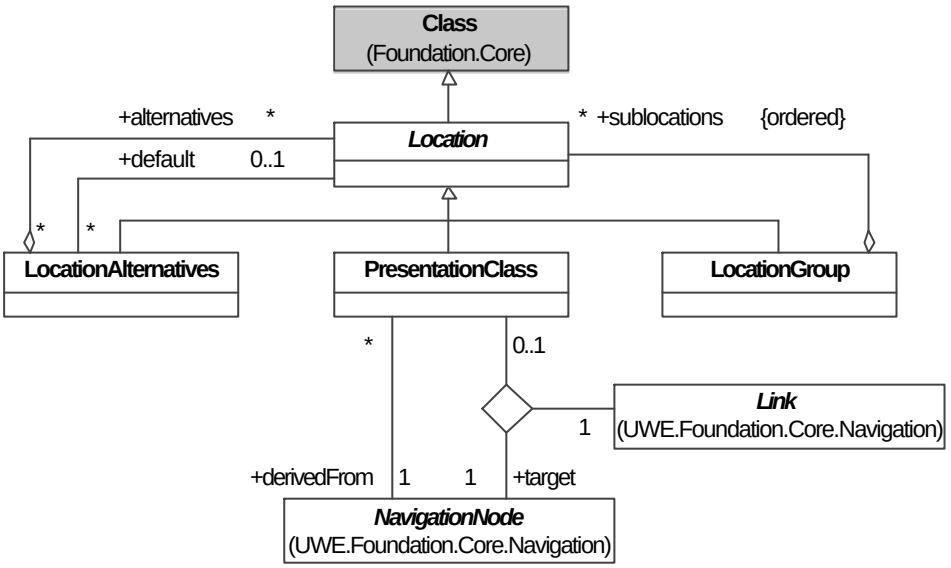

Figure 8. UWE Presentation - Backbone

All user interface elements depicted in Figure 9 are specialization of the abstract class UIElement which is associated to zero or more NavigationAttributes. User interface elements are either group-like (with the base type UIElementGroup) or primitives as for example Image, Text or TextInput. Collections are used to view homogenous sets of NavigationNodes and the subtype AnchoredCollection is connected to the Index element that represents the corresponding selection of elements. The UI elements contained within the Collection group element are used to present specific features of the set of NavigationNodes. An Anchor in general is associated to a Link element, i.e. a NavigationLink, an ExternalLink or a TaskLink. The latter may only be used for the specialization Button of Anchor. This Button is contained within a Form element that contains the input elements used as input parameters for executing a task.

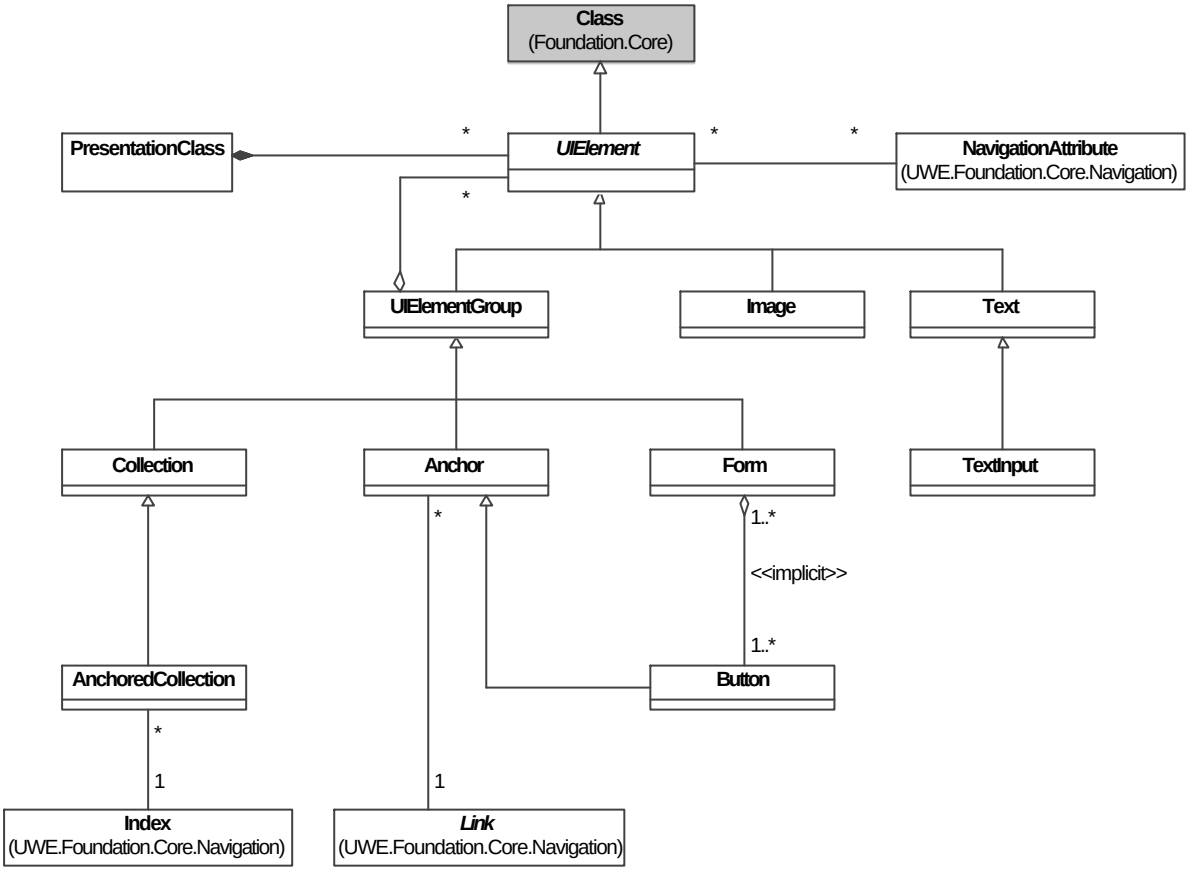

Figure 9. UWE Presentation - User Interface Elements

3.4.2 Well-Formedness Rules

The following well-formedness rules apply to the presentation package.

1. A Button may only be associated to a TaskLink (which is a subtype of Link) and a TaskLink may be only associated to a Button.

```
context     Anchor
inv:        self.oclIsKindOf(Button) implies self.link.oclIsKindOf(TaskLink)
inv:        self.link.oclIsKindOf(TaskLink) implies self.oclIsKindOf(Button)
```

3.5 User Package

The following sections describe the abstract syntax and the well-formedness rules of the user package.

3.5.1 Abstract Syntax

The basic element in the User metamodel that is shown in Figure 10 is the metaclass User that in turn is a specialization of the UML Actor element in the Use Cases package. Every user has a unique user identification and can have assigned different user roles. The important element for the adaptation aspect is UserProfile which can be assigned either to a UserRole for a group of users or to an individual user. Such a UserProfile consists of Property elements. Properties are specialized on the one hand to application independent properties such as user name or address; and on the other hand to application dependent properties where we further distinguish between the different aspects of Web applications as the conceptual, the navigation and the presentation aspect. ConceptualProperties for example are connected to ConceptualAttributes and NavigationProperties are connected to NavigationAttributes. The latter may be used to personalize the application's navigation behavior on the user behavior.

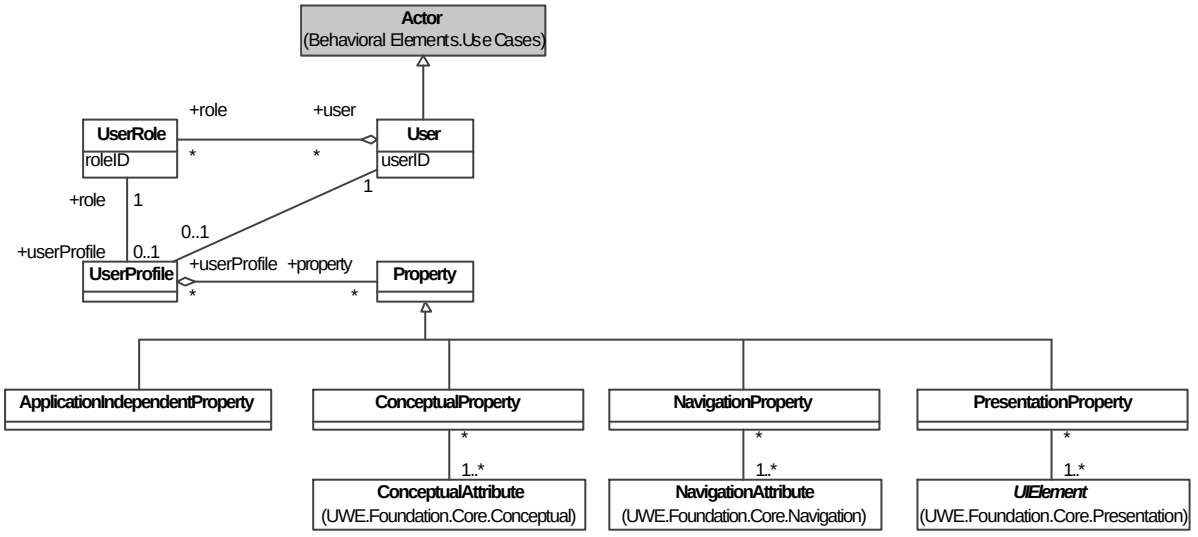

Figure 10. UWE User Package

3.6 Adaptation Package

The following sections describe the abstract syntax and the well-formedness rules of the adaptation package.

3.6.1 Abstract Syntax

The basic elements in adaptation models are the rules and the events that trigger these rules. The corresponding modeling elements in the UWE metamodel are AdaptationRule and RuleTrigger. Figure 11 shows the backbone of the adaptation metamodel. AdaptationRule is derived from the UML State Machine element Transition, and RuleTrigger is derived from the UML State Machine element Event. A Transition may have associated a Guard and an Action. Conversely, an AdaptationRule consists of exactly one AdaptationGuard and at least one

AdaptationAction. Adaptation rules are classes related to user properties and to the core elements of the conceptual, navigation and presentation packages (not visualized here). Rules are triggered by other rules or by events (called RuleTrigger in the metamodel) due to user behavior (clicking, browsing, etc) or environment behavior (mobility, network changes, etc). The RuleTrigger hierarchy is depicted in Figure 12.

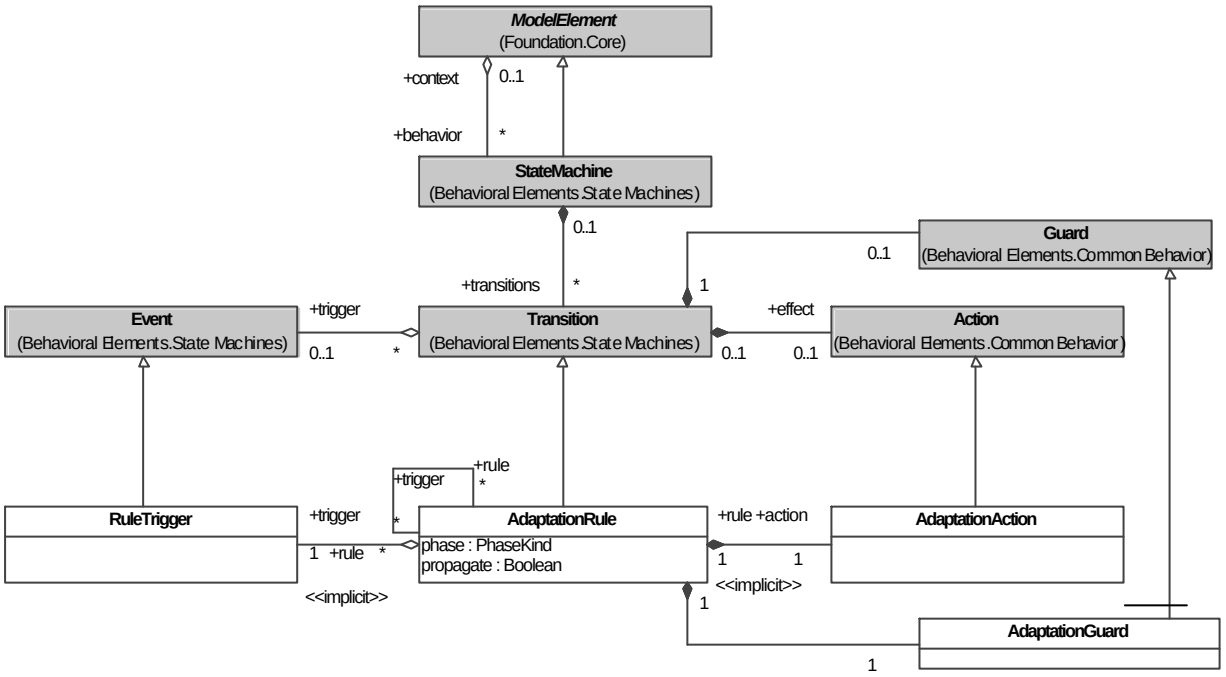

Figure 11. UWE Adaptation – Backbone

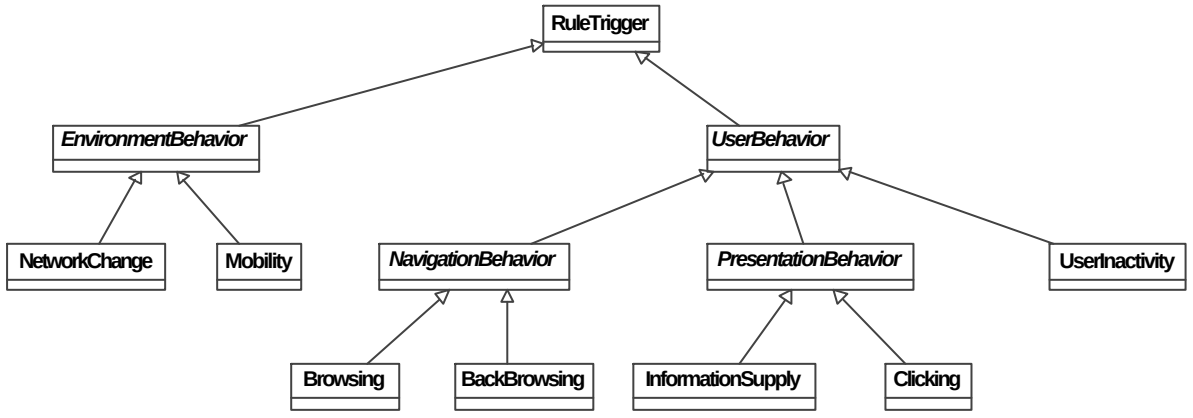

Figure 12. UWE Adaptation – RuleTrigger Hierarchy

3.7 Task Package

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).

The following sections describe the abstract syntax and the well-formedness rules of the task package.

3.7.1 Abstract Syntax

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. The use cases of the system can already be considered as tasks at analysis level. Because UML activity diagrams are normally used to further refine use cases we the UWE metamodel for task modeling is defined as extension of the UML metamodeling elements for activity diagrams. 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).

As can be seen in Figure 13 the TaskGraph element is defined as extension of the UML ActivityGraph. They themselves are extensions of StateMachines and are composed of ActionsStates on the one hand and ObjectFlowStates on the other hand. This is the extension point in the UML metamodel. We introduce an extension of CallState (which is itself a specialization of ActionState) called TaskCallState for the atomic actions (or tasks) of TaskGraphs. Further two different types of objects flows are distinguished: presentation object flow represented by the PresentationObjectFlowState is used for modeling user input and output whereas conceptual object flow represented by the ConceptualObjectFlowState is used for modeling system input and output. Task hierarchies similar to the ConcurTaskTrees of Paternó (2000) can be expressed by the concept of sub-states of UML state machines and the temporal order (with branches) between tasks is expressed by the transitions between activities.

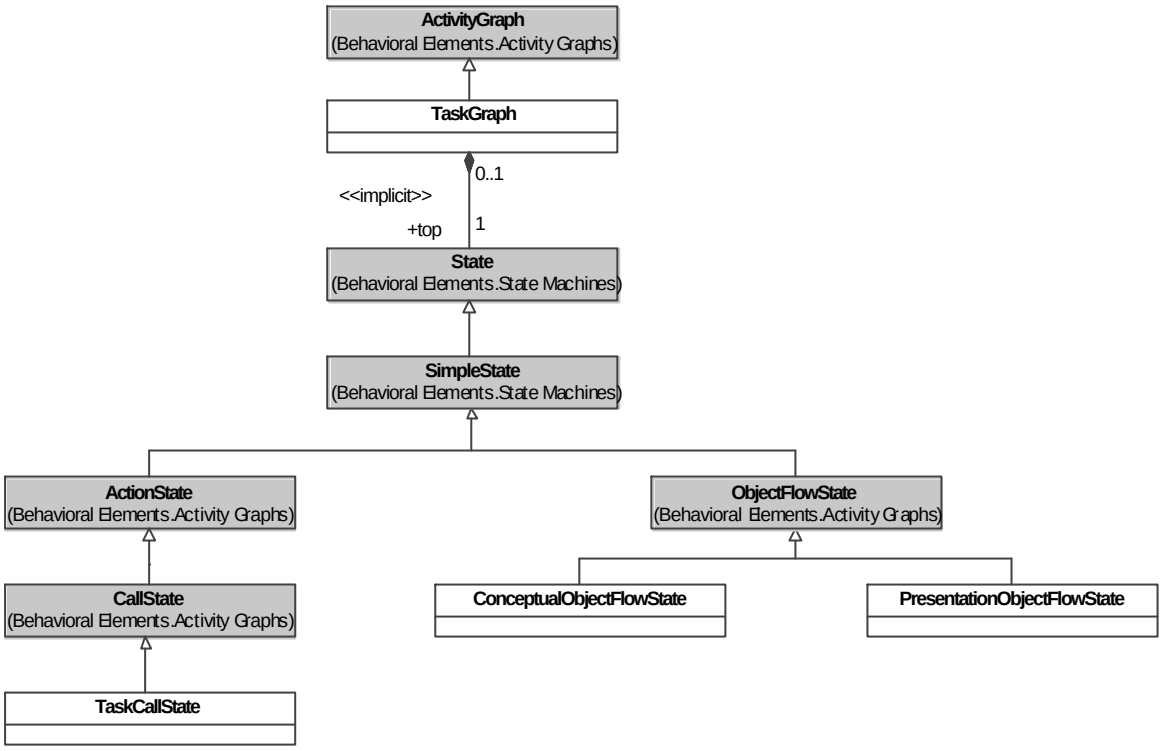

Figure 13. UWE Task Package

3.7.2 Well-Formedness Rules

The following well-formedness rules apply to the task package.

1. A ConceptualObjectFlowState may only be used together with a ConceptualClass Classifier, the same should hold for the PresentationObjectFlowState and the PresentationClass

```
context     ObjectFlowState
inv:        self.oclIsKindOf(ConceptualObjectFlowState)
                implies self.type.oclIsKindOf(ConceptualClass)
inv:        self.type.oclIsKindOf(ConceptualClass)
                implies self.oclIsKindOf(ConceptualObjectFlowState)
inv:        self.oclIsKindOf(PresentationObjectFlowState)
                implies self.type.oclIsKindOf(PresentationClass)
inv:        self.type.oclIsKindOf(PresentationClass)
                implies self.oclIsKindOf(PresentationObjectFlowState)
```

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 (ArgoUML); 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 that 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 conforms to the profile. The definition of a UML profile has the advantage that it is 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 (concrete syntax) for some (the navigation and the presentation model) of the UWE models using the UWE UML profile.

4.1 UML Profile for the Navigation Model

We use the simplified example of a conference management system presented in Figure 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 Figure 6 and Figure 14). 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 which is 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.

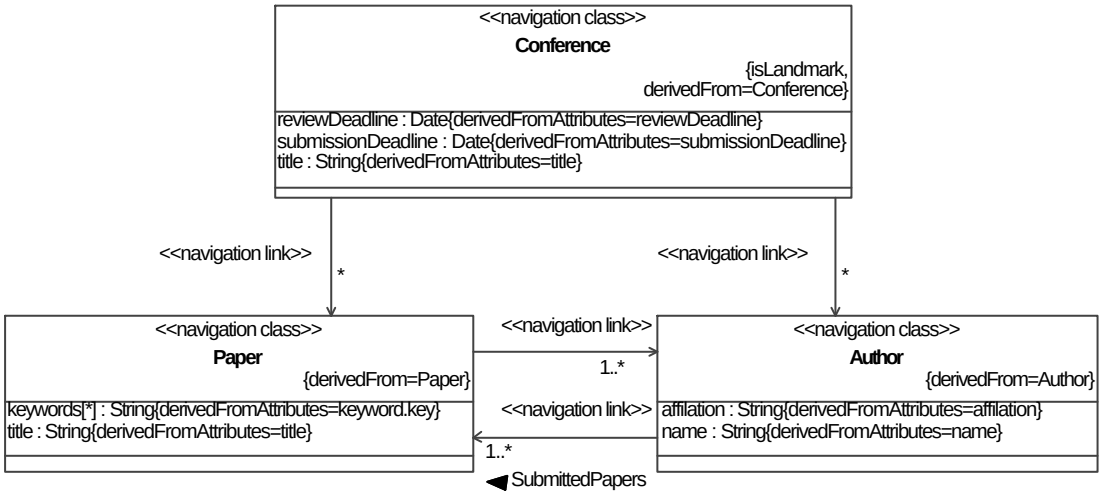

Figure 14. Example for a navigation model using the UWE UML profile

4.2 UML Profile for the Presentation Model

The three specializations of the abstract class Location (see Figure 8) 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.

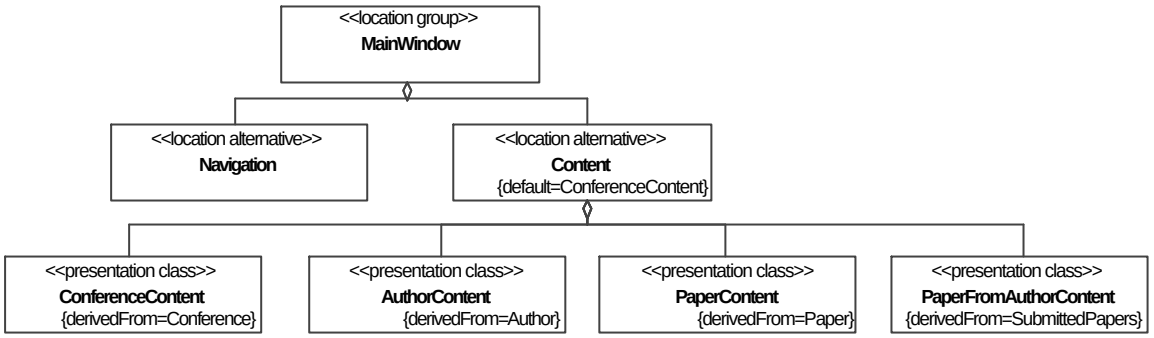

Figure 15. Example for a presentation model using the UWE UML profile

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 Figure 15 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 and Future Work

In this report 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 (Kraus and Koch, 2002) has to be extended.

6 References

ArgoUML. www.tigris.org

Atkinson C. and Kühne T. (2001). The Essence of Multilevel Metamodeling, Proc. of UML´2001, LNCS 2185, Springer Verlag, pp. 19-33.

Baresi L., Garzotto F: and Paolini P. (2002). Meta-modeling Techniques meets Web Application Design Tools. Proc. of FASE 2002, LNCS 2306, Springer Verlag, pp. 294-307.

Finkelstein A., Savigni A., Kappel G., Retschitzegger W., Pöll B., Kimmerstorfer E., Schwinger W., Hofer T.,Feichtner C. (2002). Ubiquitous Web Application Development - A Framework for Understanding, Proc. of SCI2002.

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.

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

Koch N. and Kraus A. (2002). The expressive Power of UML-based Web Engineering. Proc. of IWWOST´02, CYTED, pp. 105-119.

Koch N. and Wirsing M. (2002). The Munich Reference Model for Adaptive Hypermedia Appli-cations. Proc. of AH´2002, LNCS 2347, Springer Verlag, pp 213-222.

Kraus N. and Koch. N. (2002). Generation of Web Applications from UML Models using an XML Publishing Framework. Proc. of IDPT´2002.

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

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.

Retalis S., Papasalourus A. & Skordalakis M. (2002). Towards a generic conceptual design meta-model for web-based educational applications. *2nd. International Workshop on Web oriented Software Technology (IWWOST´02)*, CYTED.

Schwabe D & Pastor O. (Ed.). (2001). Online Proc. of IWWOST´01. www.dsic.upv.es /~west2001/iwwost01

UML (2001). The Unified Modeling Language, Version 1.4. Object Management Group (OMG). www.omg.org.