The Munich Reference Model for Adaptive Hypermedia Applications

Nora Koch, Martin Wirsing

Ludwig-Maximilians University of Munich
www.pst.informatik.uni-muenchen, Germany
{kochn,wirsing}@informatik.uni-muenchen.de

Abstract. Although adaptive applications are increasing in popularity, there are only a few approaches that focus on their generalization or the specification of a reference model. Trying to fill this gap, this paper presents a reference model for adaptive hypermedia applications, similar to AHAM. The main novelty of our approach is an object-oriented specification written in UML (Unified Modeling Language) which integrates both an intuitive visual representation and a formal unambiguous specification in OCL (Object Constraint Language). Our reference model is defined as an extension of the Dexter Hypertext Reference Model including user modeling aspects and rule-based adaptation mechanisms.

Keywords. Adaptive Hypermedia, Reference Model, Visual Modeling, UML, Formal Specification, Constraint Language, OCL.

1 Introduction

An adaptive hypermedia system is a set of nodes and links that allows one to navigate through the hypermedia structure and that dynamically "adapts" (personalizes) various visible aspects of the system to individual user's needs, preferences or knowledge [2]. These applications include an explicit representation of properties of the user. This paper presents a reference model for these adaptive hypermedia applications. The contribution of the paper is twofold. Firstly, we provide an object-oriented formalization for such a reference model. Secondly, we include a graphical representation of this model.

The objective of a reference model is to find common abstractions to the current systems and to provide a basis for the development of these applications. It is named Munich Reference Model, continuing with the tradition of choosing names of places for the reference models related to the hypermedia field, such as the Dexter Model [4], the Amsterdam Model [5] or the Dortmund Family of Hypermedia Models [9].

Adaptive hypermedia systems are first of all hypermedia systems, therefore our reference model is based on the widely used Dexter Model for hypertext systems. It includes the same three layers, but enhanced with adaptation functionality. The key aspects of the Munich Reference Model are inclusion of a user model and an adaptation model as part of the Storage Layer, the dynamic acquisition of user behavior, a dynamic rule-based adaptation and a user behavior triggered Run-Time

Published in *2nd International Conference on Adaptive Hypermedia and Adaptive Web-based Systems*, Proceedings. P. De Bra, P. Brusilovsky, and R. Conejo (eds.) LNCS 2347, ©Springer Verlag, 213-222, May 2002.

session. To our knowledge there is only one other reference model for adaptive applications: AHAM [3,12], which is semi-formally defined with tuples. The Munich Model takes an object-oriented software engineering point of view whereas AHAM takes more a database point of view. Our architecture is similar to the architecture of the AHAM reference model, where user models are always structured as tables. The AHAM adaptive engine is included in the adaptation model of our reference model as data and functionality are integrated in the object-oriented approach. An important contribution of AHAM is the adaptation rule language.

Our focus is, as already mentioned, the formal and visual description of the reference model. The Dexter Model was formalized by Halasz and Schwartz [4] in the specification language Z, early in the nineties. Since then, the use of object-oriented methodologies gained in dissemination and importance. In addition, more emphasis is now put on visual modeling languages making models more intuitive. These were our motivations to select the Unified Modeling Language (UML) – standard for object-oriented modeling – for the formalization of the Munich Reference Model. On the one side, the UML [10] provides the notation and techniques (diagrams) for the visual representation. It has the advantage of showing the relevant concepts at a glance, how they are organized and how they are related to each other. This augments the intuitive comprehension. On the other side, the Object Constraint Language (OCL) which is part of the UML, is used to supplement the semi-formal graphical representation with formally written semantics information.

The Munich Reference Model constitutes the basis for the UML-based Web Engineering (UWE) approach that focus on development of adaptive hypermedia applications [6,7]. UWE includes a design method and the description of a development process that covers the whole life-cycle of these applications. This reference model was used in the development of SmexWeb, a framework for implementing adaptive learning systems on the Web [1]. SmexWeb supports all type of dynamic adaptations, i.e. content, link and presentation adaptation.

This paper is an outline of the specification of the Munich Reference Model; the complete version is included in [6]. It is organized as follows: Section two gives an overview of the Munich Reference Model. Section three presents the specification of the domain. Section four introduces the extensions to include user modeling and adaptivity. Section five briefly presents the specification of a hypermedia session management. Finally, in the last section some conclusions are outlined.

2 An Overview of the Reference Model

The Munich Reference Model preserves the three-layer structure of the Dexter Model describing the network of nodes and links and the navigation mechanism. It extends the functionality of each layer to include the user modeling and adaptation aspects. The Run-Time Layer, the Storage Layer and the Within-Component Layer are represented as UML subsystems as it is illustrated in Figure 1.

- The *Run-Time Layer* contains the description of the presentation of the nodes and links. It is responsible for user interaction, acquisition of user behavior and management of the sessions.

- The *Storage Layer* has more functionality than just storing information about the hypermedia structure. To support adaptation the Storage Layer is divided into three sub-models:
  - The *Domain Meta-Model* that manages the basic network structure of the hypermedia system in terms of mechanisms by which the links and nodes are related and navigated. The nodes are treated as general data containers.
  - The *User Meta-Model* manages a set of users represented by their user attributes with the objective to personalize the application.
  - The *Adaptation Meta-Model* consists of a set of rules that implement the adaptive functionality, i.e. personalization of the application.
- The content and structure within the hypermedia nodes are part of the *Within-Component Layer*, which is not further detailed as its structure and content depend on the application.

The functionality of adaptive hypermedia systems is specified by three types of operations included in the classes of the reference model:

- *Authoring operations* are needed by adaptive hypermedia systems to update components, rules and user attributes, e.g. to create a link or a composite component, to create a rule, to add an user attribute to the model, to delete components or rules.
- *Retrieval operations* are required to access the hypermedia domain structure and the User Model, e.g. to get a component, to get all rules triggered by a user's behavior or another rule.
- *Adaptation operations* are used to dynamically adapt the User Model content to the user behavior and to adapt the presentation to the current state of the User Model, e.g. the adaptive resolver, the constructor or the rule executor.

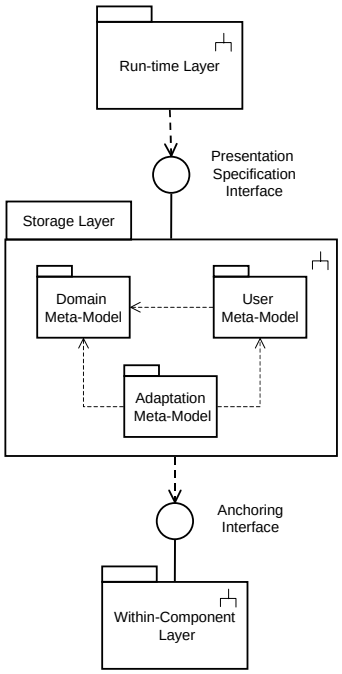

Fig. 1: Architecture of Adaptive Hypermedia Applications

The remainder of this paper presents the visual specification (slightly simplified) of the layers of the reference model and includes a few constraints of the formal specification out of a total of seventy constraints that comprise the complete specification of the Munich Reference Model [6].

3 Specification of the Domain Model

The Domain Meta-Model describes the structure of a hypermedia as a finite set of components together with three main operations, a *resolver*, an *accessor* and a

constructor. These concepts are modeled by a *class Domain* and a *class Component.* Every component has a globally unique identity *(class UID)*. With the operations *resolver*, *accessor* and *constructor* it is possible to "retrieve" and "construct" adaptive components. The accessor operation allows one to "access" a component given its UID. UIDs "resolve" to a component. This way UIDs provide a guaranteed mechanism for addressing any component in the hypermedia domain. As in the Dexter Model, this addressing is accomplished in a indirect way based on the entities called anchor *(class Anchor)* consisting of two parts: an anchor ID *(class AnchorID)* and an anchor value *(class AnchorValue)*. The anchor value is an arbitrary value that specifies some location within a component. The anchor ID is an identifier that uniquely identifies the anchor within the scope of the component.

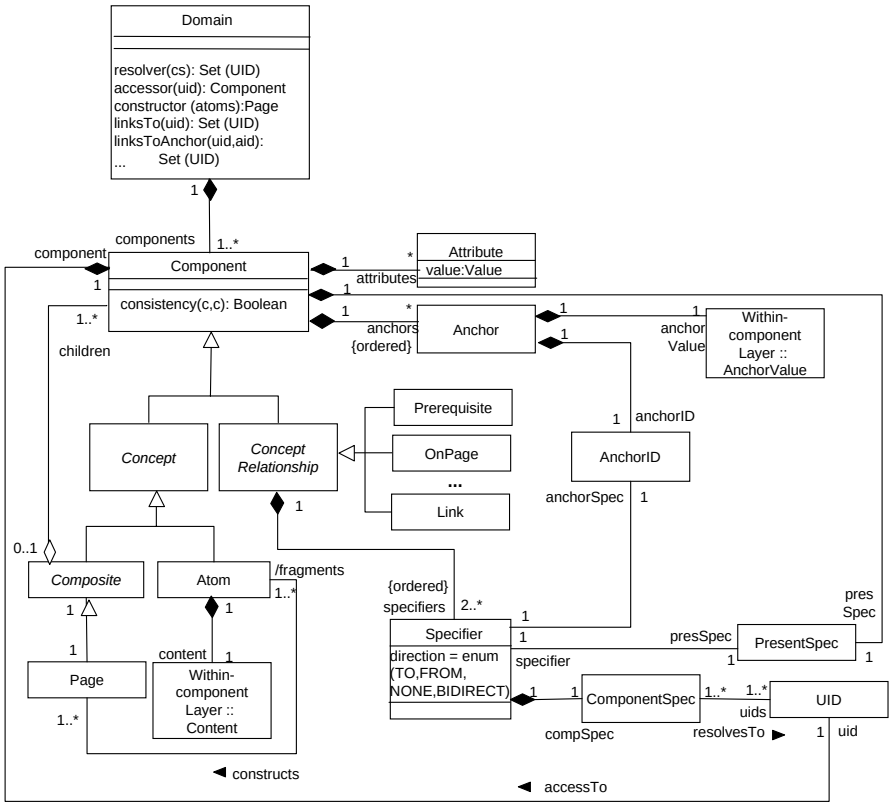

Fig. 2: View of the Domain Meta-Model of the Munich Reference Model

The visual specification of the Domain Meta-Model of the Munich Reference Model is represented by an UML class diagram. Part of it is shown in Figure 2. Note that for classes belonging to another package, the name of the package is included before the class name, e.g. Within-Component-Layer::AnchorValue. As examples, the classes Component and Domain are described with some more detail below.

Component. A component is an abstract representation of an information item from the application domain. It is represented by an abstract *class Component*. A component can either be a concept *(class Concept)* or a concept relationship *(class ConceptRelationship).* A concept, in turn, can either be an atom *(class Atom)* or a composite *(class Composite).* A concept relationship can be a link *(class Link)* or a prerequisite *(class Prerequisite),* or a is-part-of relation (*class On page),* etc. This inheritance hierarchy is shown in the UML class diagram (Figure 2). The component information consists of attributes *(class Attribute),* a presentation specification *(class Present Spec)* and a sequence of anchors *(class Anchor).* The UML visual specification is insufficient to model the "type consistency" between components. Therefore, the following OCL constraint is added to the specification to express that two components are "type consistent" if both are of the same type.

```
context Component :: consistency (c1:Component, c2:Component):Boolean
post:   result = c1.oclIsTypeOf(Composite) and c2.oclIsTypeOf(Composite)
              or  c1.oclIsTypeOf(Link) and c2.oclIsTypeOf(Link)
              or  c1.oclIsTypeOf(Atom) and c2.oclIsTypeOf(Atom)
               or .....
```

Domain. The domain is represented by a *class Domain*, which is a composition of objects of type *Component*. The class *Domain* includes two operations for links and anchors ensuring the navigation functionality of the hypermedia system. These are the *linksTo* and the *linksToAnchor* operations. The *linksTo* operation returns the set of links that resolve to a specific component. The *linksToAnchor* obtains the set of links that resolve to a specific anchor. The following is the OCL specification of the pre-condition and post-condition of *linksTo.* The post-condition expresses that *result* consists of the set of all *Link* identifiers such that one of the component specifications of the corresponding Link resolves to the given UID.

```
context Domain :: linksTo ( uid : UID ) : Set (UID)
 pre: components → exists ( c : Component | accessor (uid) = c )
 post: result = UID.allInstances → select ( lid : UID |
         Component.allInstances → exists (link :Component |
           link.oclIsTypeOf (Link)  and  link  =  accessor (lid)
          and ComponentSpec.allInstances  → exists ( cs :
          ComponentSpecs | link.specifiers.compSpecs → includes (cs)
              and   uid = resolver (cs) ) ) )
```

4 Modeling Adaptive Hypermedia Applications

The Munich Reference Model includes adaptation and user modeling functionality. The User Meta-Model defines the user attributes and attribute-values that are relevant to the adaptive application. The adaptive mechanisms are specified in the Adaptation Meta-Model and they are responsible for adaptive content, adaptive links and adaptive presentation. The presentation specification builds pages out of page fragments, taking into account the adaptive mechanisms.

4.1 The User Meta-Model

The User Meta-Model describes the structure of the individual models of each user and how these models are administrated. User modeling comprises initialization, updating and retrieval of the current state of a User Model. The User Meta-Model is modeled as a subsystem that consists of a class *UserManager* and a set of *Users* and operations *initializer*, *updater* and an *evaluator*. Figure 3 depicts the classes of the User Meta-Model subsystem and its relationship to the Domain Model.

A user of an adaptive hypermedia application is modeled by a class *User*, which is related through an aggregation association to a *UserIdentification* and to a set of *User Attributes*. The user ID identifies the user uniquely in the universe of the application. With the user attributes the system provides a representation of the user's characteristics that are relevant for the application. One can distinguish different types of information contained in user models: user's knowledge, user's preferences, user's background experience, user's tasks, etc., summarized in two categories: "user knowledge related to the domain components" and "user general characteristics".

The first group includes domain dependent attributes while those of the second group are domain independent. The second group includes knowledge not related to the components, such as background knowledge and preferences. Classification like this can be found in Hynecos [11] and SmexWeb [1]. We model these two groups of user's characteristics with *Class DependentAttr* and *Class IndependentAttr*. The separation has the advantage that the domain independent attributes can be shared with other applications. The following constraint defines the invariant for a domain independent User Model, i.e. all user attributes are independent of the domain.

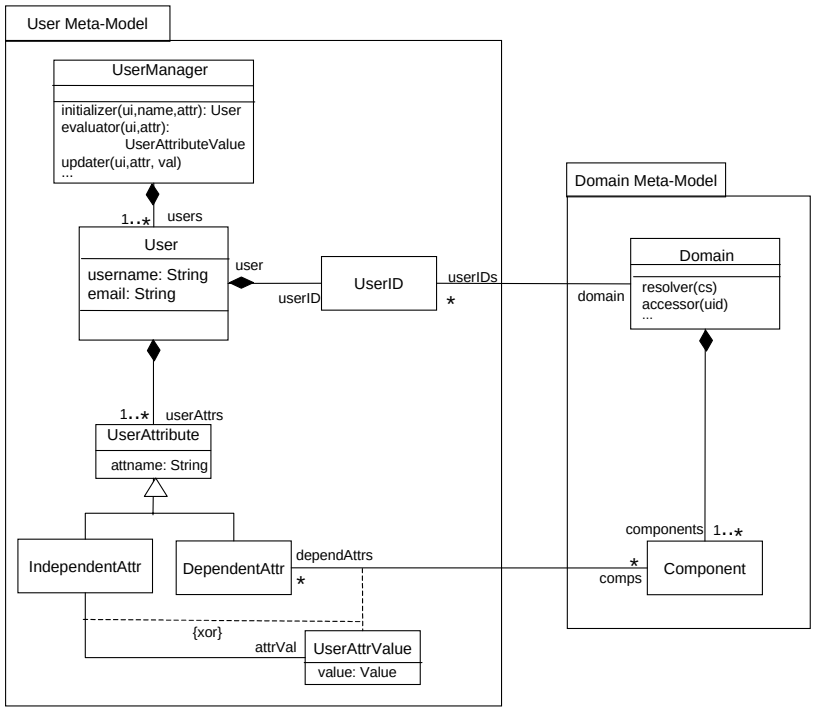

Fig. 3: View of the User Meta-Model of the Munich Reference Model

```
context User
inv domain independent user model:
      userAttrs → forAll ( uat:: UserAttribute |
            uat.oclIsTypeOf (IndependentAttr) )
```

Let us mention here only the formalization of one functionality related to the User Meta-Model subsystem: the registration of a new user. We define an *initializer* operation that creates a new instance of class *User* for each new user that registers to the adaptive hypermedia application and assigns a given set of attributes to this user.

```
context UserManager :: initializer (userIdentification:UserID, n:String,
                        defaultAttrs: Set(UserAttribute)) : User
post:   result.oclIsNew and users = users@pre → including (result)
        and result.userID = userIdentification and result.username = n
        and result.userAttrs = defaultAttrs
```

4.2 The Adaptation Meta-Model

The adaptation is performed using a set of rules, such as in most adaptive hypermedia applications; typical examples of rule-based adaptation is supported by the frameworks AHA [3] and SmexWeb [1]. These rules determine how pages are built and how they are presented to the user. The Adaptation Meta-Model is specified by a UML class diagram, which is depicted in Figure 4.

The core elements used to model the adaptation are the class *Adaptation* and the class *Rule*. The class *Adaptation* includes three main operations: an *adaptation resolver*, a *finder* and a *trigger*. The first one "resolves" a component specification into a UID of an appropriate component that builds an adapted page. The second one implements a trigger mechanism that returns all the rules triggered by one given rule, i.e. the rules to be used at a given time. The first rule to be used is triggered by the user behavior. The *executor* operation of the class *Rule* allows the system to select the appropriate components, and to perform content-adaptation, presentation-adaptation and link-adaptation as well as to update the User Model. These operations play the role of the adaptive engine in AHAM.

The specification of the Adaptation Meta-Model is supplemented with a set of OCL constraints. For example, the following OCL invariant assures the dynamic update of the User Model: For at least one user attribute there exists a rule that modifies an attribute value of the User Model.

```
context Adaptation
inv dynamic update of the user model:
     Rules.allInstances → exists ( r:Rule | r.oclIsTypeOf (AcquisitionRule)
        and r.action.elements → exists (m: ModelElement |
              m.values.oclIsTypeOf (UserAttributeValue)
              and m.modified ) )
```

An object of class *Rule* consists of one condition (class *Condition*), one action (class *Action*) and attributes, such as phase and propagate proposed by De Bra et. al [3]. *phase* determines whether rules are applied before or after the User Model is

updated while *propagate* with a value true allows the system to trigger other rules. Conditions and actions are expressions containing model elements and operators.

ModelElements are defined by two attributes: an element identifier (*elementID)* and a Boolean value (*modified)* which indicates whether the model element is being modified in the actual action. Only certain types of model elements, i.e. User Model attribute values and presentation specifications can have a *modified* value true.

Our formalization of rules is very general, thus our reference model does not prevent problems with confluence and termination of rule-based systems. Depending on the chosen rule language, rule applications may be non-terminating and non-confluent. These problems can be analyzed in each case using different approaches, such as those used in rewriting systems or in the active database field.

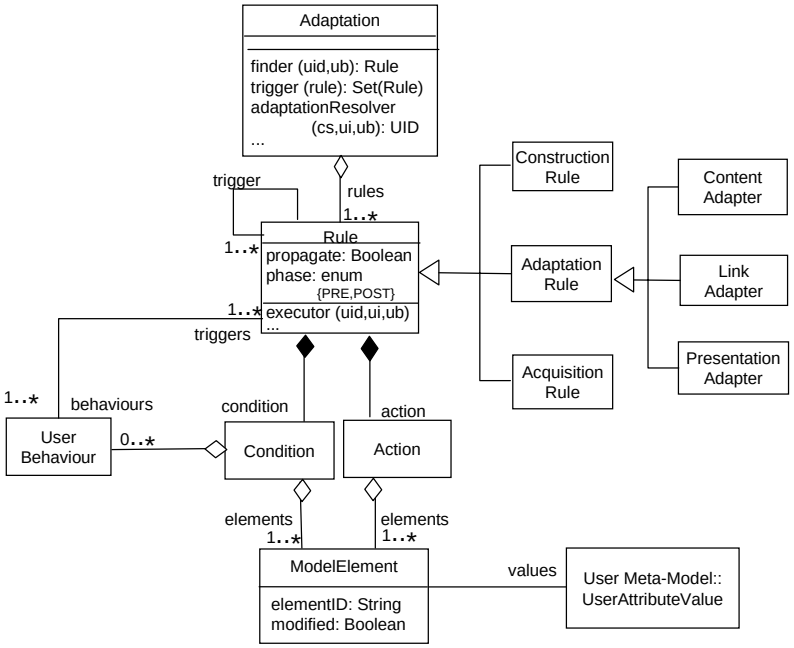

Fig. 4: View of the Adaptation Model of the Munich Reference Model

Rules are classified according to their objectives into: construction rules, acquisition rules and adaptation rules [6]. Adaptation rules adapt content, links or the presentation of the application. They differ in the executor method. The different types of rules are represented as a hierarchy of rules as it is shown in the class diagram of the Adaptation Model (see Figure 4).

5 Session Management

The Run-Time Layer manages different sessions for the users generating and presenting the instances of pages. The Run-Time Layer describes how the

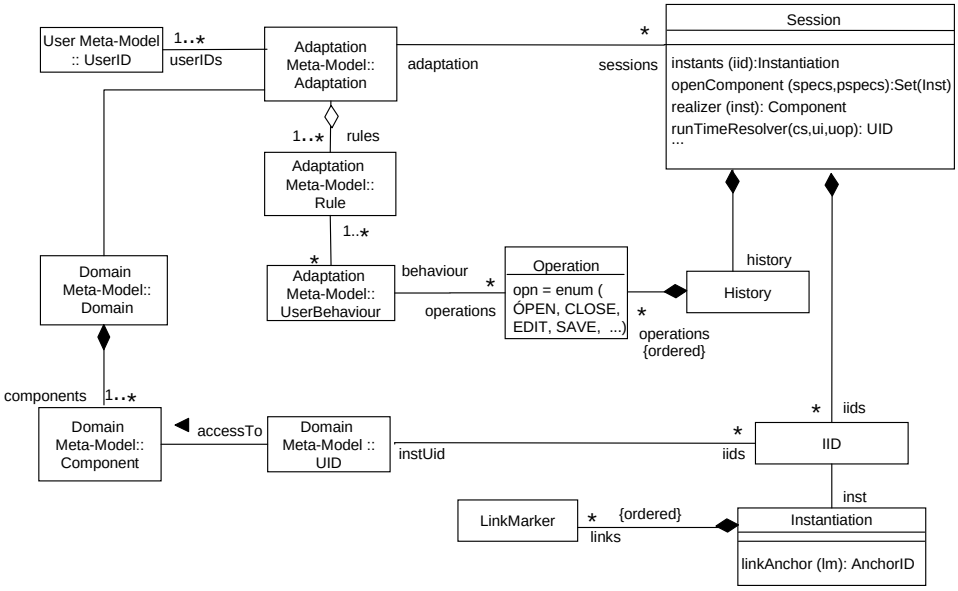

Fig. 5: View of the Run-Time Layer of the Munich Reference Model

components are presented to the user. This presentation is based on the concept of instantiation of a component, i.e. a copy of the component is cached to the user. The copy receives an instantiation identifier (class *IID*). It should be noted that more than one instantiation for a component may exist simultaneously and that a user may be viewing more than one component.

Instantiation of a component also results in instantiation of its anchors. An instantiated anchor is known as a link marker. These concepts are modeled with the *classes Instantiation, IID,* and *LinkMarker*. In order to keep track of all these instantiations the Run-Time Layer uses an entity session (*class Session*) as shown in Figure 5. A session can be open or closed and in a session the user can perform operations, such as open a component that results in the creation of an instantiation, edit an instantiation and follow a link. All these operations that result from the user interactions are recorded in a history which constitutes the basis of the observation of the user behavior and the adaptation mechanism. As example a constraint for the *instantiator* operation is shown. Given an UID of a component, the function returns an instantiation of the component that is part of the session. The presentation specification is a primitive in the model, which contains information about how the component is to be presented by the system during instantiation (see Figures 5 and 2).

```
context  Session :: instantiator (uid: UID, ps: PresentSpec): Instantiation
pre: adaptation.domain.components → includes (accessor(uid) )
post: result = iids.inst → select (ins:Instantiation |
        ins.presSpec = ps  and  ins.iid.instUID = uid ) → asSequence → first
```

Figure 5 depicts part of the Run-Time Layer for adaptive hypermedia systems. The UML class diagram shows how core classes of the Domain Meta-Model, the User Meta-Model and the Adaptation Meta-Model collaborate with classes of the Run-Time Layer.

6 Conclusions and Future Work

In this paper we introduced an object-oriented reference model for adaptive hypermedia applications – called Munich Reference Model. It constitutes the basis of our UWE engineering approach for adaptive Web applications [6,8]. This model was defined in parallel to the development of SmexWeb. SmexWeb is a framework for adaptive Web learning applications [1]. The architecture of this model is similar to the AHAM architecture, i.e. it extends the Dexter model by including an user model and an adaptation model in the Storage Layer as well as adapting the domain model and the Run-Time Layer.

We presented an integrated visual and formal specification of the reference model. The model is visually represented using UML notation and is formally specified in OCL. UML was chosen as it is the de facto standard modeling language. A visual representation is missing when other kind of specification, e.g. Z or VDM are selected [4]. OCL is part of the UML and is used for the specification of invariants for the model elements and for the specification of pre-conditions and post-conditions on operations describing the adaptive functionality.

Acknowledgment. We thank the reviewers for their valuable feedback and their requests for additional explanations**.**

References

1. Albrecht F., Koch N. and Tiller T. (2000). SmexWeb: An Adaptive Web-based Hypermedia Teaching System. *Journal of Interactive Learning Research*. Kommers P. & Mizoguchi R. (Eds.).
2. Brusilovsky P. (1996). Adaptive Hypermedia: An attempt to analyze and generalize. *Proceedings of First International Conference on Multimedia, Hypermedia and Virtual Reality 1994*. Brusilovsky P. & Streitz N. (Eds.) LNCS 1077, Springer Verlag, 288-304.
3. De Bra P., Houben G.-J., and Wu H. (1999). AHAM: A Dexter-based Reference Model of Adaptive Hypermedia. *Proceeding of the ACM Hypertext Conference*.
4. Halasz F. and Schwartz M. (1990). The Dexter Hypertext Reference Model. NIST Hypertext Standardization Workshop.
5. Hardman L., Bulterman C. and Rossum G. (1994). The Amsterdam Reference Model. *Communications of the ACM 37(2).*
6. Koch N. (2000). Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process. *PhD. Thesis*, Uni-Druck.
7. Koch N. (2002). An Object-Oriented Hypermedia Reference Model. *In Information Modeling for Internet Applications,* van Bommel P. (Ed.), to appear.
8. Koch N. and Wirsing M. (2001). Software Engineering for Adaptive Hypermedia Applications? *Third Workshop on Adaptive Hypertext and Hypermedia at the UM´2001*.
9. Tochtermann K. and Dittrich G. (1996). The Dortmund Family of Hypermedia Systems. *Journal of Universal Computer Science*.
10. UML: The Unified Modeling Language. Version 1.3. (1999). http://www.omg.org/uml
11. Vassileva J. (1994). A Practical Architecture for User Modeling in a Hypermedia-based Information System. *Proceeding of the 4th International Conference on User Modeling.*
12. Wu H., De Bra P., Aerts A. and Houben G.-J. (2000): Adaptation Control in Adaptive Hypermedia Systems. *Proceedings of the Adaptive Hypermedia and Adaptive Web-based Systems*. Brusilovsky P, Stock O., Strapparava C. (Eds.). LNCS 1892, Springer Verlag,250-259.