

## Chapter 7

# UML-BASED WEB ENGINEERING

## *An Approach Based on Standards*

Nora Koch,<sup>1,2</sup> Alexander Knapp,<sup>1</sup> Gefei Zhang,<sup>1</sup> Hubert Baumeister<sup>3</sup>

<sup>1</sup>*Institut für Informatik, Ludwig-Maximilians-Universität München, Germany*  
{ kochn, knapp, zhang } @pst.ifi.lmu.de

<sup>2</sup>*F.A.S.T. GmbH, München, Germany*  
koch@fast.de

<sup>3</sup>*Informatik og Matematisk Modellering, Danmarks Tekniske Universitet, Lyngby, Denmark*  
hub@imm.dtu.dk

### 7.1 Overview

UML-based Web Engineering (UWE, [www.pst.ifi.lmu.de/projekte/uwe](http://www.pst.ifi.lmu.de/projekte/uwe)) came up by the end of the nineties (Baumeister et al., 1999; Wirsing et al., 1999) with the idea to find a standard way for building analysis and design models of Web systems based on the then current methods OOHDM (Schwabe and Rossi, 1995), RMM (Isakowitz et al., 1995), and WSDM (de Troyer and Leune, 1998). The aim, which is still pursued, was to use a common language or at least to define metamodel-based mappings among the existing approaches (Koch and Kraus, 2003; Escalona and Koch, 2006).

At that time the Unified Modeling Language (UML) which evolved from the integration of the three different modeling approaches of Booch, OOSE and OMT seemed to be a promising approach for system modeling. Since the early integration efforts, the UML became the “lingua franca” of (object-oriented) software engineering (Object Management Group, 2005). A prominent feature of UML is that it provides a set of aids for the definition of domain-specific modeling languages (DSL) – so called extension mechanisms. Moreover the newly defined DSLs remain UML-compliant, which allows the use of all UML features supplemented, e.g., with Web specific extensions.

Both the acceptance of the UML as a standard in the development of software systems and the flexibility provided by the extension mechanisms are the reasons for the choice of the Unified Modeling Language instead of the use of proprietary modeling techniques. The idea followed by UWE to adhere to standards is not limited to UML. UWE uses also XMI as model exchange format (in the hope of future tool interoperability enabled by a truly portable XMI), MOF for metamodeling, the model-driven principles given by OMG's Model-Driven Architecture (MDA) approach, the transformation language QVT, and XML.

UWE is continuously adapting, on the one hand, to new features of Web systems, such as more transaction-based, personalized, context-dependent, and asynchronous applications. On the other hand, UWE evolves to incorporate the state of the art of software engineering techniques, such as aspect-oriented modeling, integration of model checking using Hugo/RT (Knapp et al., 2002, [www.pst.ifi.lmu.de/projekte/hugo](http://www.pst.ifi.lmu.de/projekte/hugo)), and new model transformation languages to improve design quality.

The remainder of this chapter is structured as follows: The features distinguishing UWE's development process, visual notation and tool support, are briefly outlined below. UWE's modeling techniques are discussed step by step in Sect. 7.2 by means of a simple online movie database case study. The UWE extensions of the UML metamodel are outlined in Sect. 7.3. UWE's model-driven process and, in particular, the model transformations integrated into the process are described in Sect. 7.4. The CASE tool ArgoUWE which supports the UWE notation and method is described in Sect. 7.5. Finally, we give an outlook on future steps in the development of UWE.

### 7.1.1 Characteristics of the Process

The development of Web systems is subject to continuous changes in user and technology requirements. Models built so far in any stage of the development process have to be easily adaptable to these changes. To cope efficiently with the required flexibility, UWE advocates a strict separation of concerns in the early phases of the development and implements a model-driven development process, i.e. a process based on the construction of models and model transformations. The ultimate challenge is to support a development process that allows fully automated generation of Web systems.

**Separation of Concerns** Similarly to other Web engineering methods, the UWE process is driven by the separate modeling of concerns describing a Web system. Models are built at the different stages of requirements engineering, analysis, design, and implementation of the development process, and are used to represent different views of the same Web application corresponding to the different concerns (content, navigation structure, and presentation). The

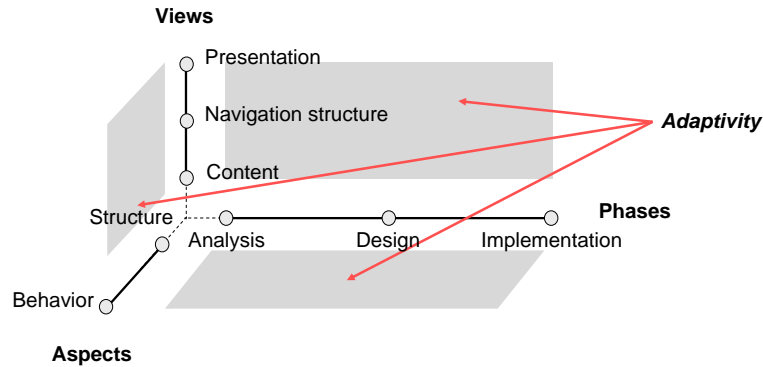


Figure 7.1. Modeling dimensions in UWE (Schwinger and Koch, 2006)

content model is used to specify the concepts that are relevant to the application domain and the relationships between these concepts. The hypertext or navigation structure is modeled separately from the content, although it is derived from the content model. The navigation model represents the navigation paths of the Web system being modeled. Presentation specification takes into account representation and user-machine communication tasks.

UWE proposes at least one type of UML diagram for the visualization of each model to represent the structural aspects of the different views. However, in addition very often UML interaction diagrams or state machines are used to represent behavioral aspects of the Web system. Figure 7.1 shows how the scope of modeling spans these three orthogonal dimensions: development stages, system's views, and aspects. Another concern also handled separately is adaptivity. Personalized and context-dependent Web systems provide the user with more appropriate information, links, or pages by being aware of user or contextual features. We propose to view adaptivity as a cross-cutting concern and thus to use aspect-oriented techniques to model adaptive Web systems. It can be seen as a fourth dimension influencing all other Web modeling dimensions: views, aspects, and phases. Requirements models and architecture models focusing on specific Web aspects complete the specification of the Web system. Separation of concerns offers advantages in the maintenance and re-engineering of a Web system as well as for the generation of Web systems for different contexts and platforms.

**Development Driven by Models** The model-driven development (MDD) approach not only advocates the use of models (as those described above) for the development of software, but also emphasizes the need of transformations in all phases of the development, from requirements specification to designs and from design models to implementations. Transformations be-

tween models provide a chain that enables the automated implementation of a system in successive steps from the different models.

The development of Web systems is a field which lends itself to applying MDD due to the Web-specific separation of concerns and continuous changes in technologies in the Web domain. Metamodel-based methods such as OO-H (Gómez et al., 2001) and UWE constitute a good basis for the implementation of a model-driven process for the development of Web systems. They included semi-automated model-based transformations even before MDD concepts became well-known. For the first guidelines for a systematic and stepwise construction of models for UWE we refer to Hennicker and Koch, 2001 and Koch, 2001.

UWE emphasizes the relevance of requirements engineering starting with modeling activities in this early development phase (Escalona and Koch, 2006). Therefore the UWE metamodel includes a set of modeling primitives that allows for simpler and more specific specification of the requirements of Web systems.

### 7.1.2 Characteristics of the Notation

A picture is worth a thousand words. Visual models are naturally used not only for documentation purposes but also as the crucial chain link in the software development process. The trend is the production of domain-specific visual models. Conversely, the importance of the selection of the modeling language is not self-evident.

From our point of view, a modeling language has to:

- 1 provide powerful primitives to construct expressive, yet intuitive models;
- 2 offer wide CASE tool support;
- 3 facilitate extension;
- 4 provide a formal or at least a semi-formal semantics;
- 5 be easy to learn.

Although UML only fulfills the first three requirements, it seems that UML is currently the best approach. UML and various UML extensions are successfully used in many different application domains. However, there is no formal semantics covering the whole UML and the fifth requirement can only be satisfied, if we restrict ourselves to a subset of the modeling constructs of UML.

**Modeling with UML** The distinguishing feature of UWE is its UML compliance since the model elements of UWE are defined in terms of a UML

profile and as an extension of the UML metamodel (Koch and Kraus, 2002; Koch and Kraus, 2003).

Although the UML is expressive enough to model all requirements that arise in modeling Web systems, it does not offer Web-domain-specific elements. To ease the modeling of special aspects of Web applications, we define in UWE special views – using UML’s extension mechanisms – graphically represented by UML diagrams, such as the navigation model and the presentation model (Koch, 2001; Koch et al., 2001).

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 communication diagrams. The UML extension mechanisms are used to define stereotypes that we utilize for the representation of Web constructs, such as nodes and links. In addition, tag definitions and constraints written in OCL (Object Constraint Language) can be used. This way we obtain a UML compliant notation – a so called UML “lightweight” extension or better known as UML profile. UWE notation is defined as such a UML profile.

The advantage of using UML diagrams is the common understanding 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 (Object Management Group, 2005). For any software designer with UML background it is easy to understand a model based on a UML profile, such as the extension that UWE suggests. We observe that UML extensions “inherit” the problems of UML, e.g., the lack of a complete formal semantics covering all modeling elements.

UWE focuses on visual modeling together with systematic design and automatic generation. The aim is to cover the entire development life cycle of Web systems providing techniques and notations to start with requirements models, moving through design models as well as including architecture and aspect models. All these models are visualized using UML diagrammatic techniques.

**Metamodeling** Metamodeling plays a fundamental role in CASE tool construction and is as well the core of the model-driven process. A metamodel is a precise definition of the elements of a modeling language, their relationships and the well-formedness rules needed for creating syntactically correct models.

Tool-supported design and model-based system generation is becoming essential in the development process of Web systems due to the need of rapid production of new Web presences and Web applications. CASE tools have to be built on a precisely specified metamodel of the modeling constructs used in the design activities, providing more flexibility if modeling requirements

change. Metamodels are essential for the definition of model transformations and automatic code generation.

The UWE metamodel is defined as a conservative extension of the UML metamodel (Koch and Kraus, 2003). It is the basis for the UWE notation and UWE tool support. Conservative means that the modeling elements of the UML metamodel are not modified, e.g., by adding additional features or associations to the UML modeling element Class. OCL constraints are used to specify 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 based 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. A UML profile consists of a hierarchy of stereotypes and a set of constraints. Stereotypes are used for representing instances of metaclasses and are written in guillemets, like «menu» or «anchor». 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. The UWE metamodel could be also used as basis for building a common metamodel (or ontology) of the concepts needed for the design in the Web domain (cf. Koch and Kraus, 2003; Escalona and Koch, 2006). Using for this purpose the standardized OMG metamodeling architecture would facilitate the construction of meta CASE tools.

### 7.1.3 Characteristics of the Tool Environment

The UML compliance of UWE has an important advantage: all CASE tools which support the Unified Modeling Language can be used to build UWE models. For this purpose it is sufficient to name stereotypes after the names of the UWE modeling concepts. Many tools offer additional support with an import functionality of pre-defined UML profiles. In such a case the profile model elements can be used in the same way as the built-in UML model elements.

**CASE Tool Support** A wider developer support is achieved by the open source plug-in ArgoUWE ([www.pst.ifi.lmu.de/projekte/uwe](http://www.pst.ifi.lmu.de/projekte/uwe)) for the open source CASE tool ArgoUML ([www.argouml.org](http://www.argouml.org)). In addition to providing an editor for the UWE notation, ArgoUWE checks the consistency of models and supports the systematic transformation techniques of the UWE method. Using the UWE profile, models designed with other UML CASE tools can be exchanged with ArgoUWE. The use of tools that support not only the modeling itself but also a model-driven approach shortens development cycles and facilitates re-engineering of Web systems.

**Model Consistency Check** ArgoUWE also checks the consistency of models according to the OCL constraints specified for the UWE metamodel. Consistency checking is embedded into the cognitive design critics feature of ArgoUML and runs in a background thread. Thus, model deficiencies and inconsistencies are gathered during the modeling process but the designer is not interrupted. The designer obtains feedback at any time by taking a look at this continuously updated list of design critiques, which is shown in the to-do pane of the tool.

In the following, we exemplify how UWE's model-driven process, notation, and tool support are used to develop Web applications.

## 7.2 Method by Case Study

We use a simple online movie database example that allows users to explore information about movies and persons related to the production of the movies. This example is inspired by [www.imdb.org](http://www.imdb.org) and named the "Movie UWE Case study" (MUC). Movies are characterized, among others, by their genre, the cast, memorable quotes, trailers, and a soundtrack. Persons related to the movie production are the director, producer, composer, and the actors. The user interested in watching a movie can access information on theaters that show the movie. Registered users – identified by an email and a password – can provide comments, rate comments, vote movies, manage "their movies", and buy tickets in theaters of their preference. The MUC online movie database personalizes the application giving some recommendations about movies and providing personalized news to the user.

The focus in the following is on the models built for the different views of the analysis and design phases (see Fig. 7.1). Model transformations are described as part of the model-driven process in Sect. 7.4.

### 7.2.1 Starting with Requirements Specification

The first step towards developing a Web system is the identification of the requirements for such an application that are specified in UWE with a *requirements model*. Requirements can be documented at different levels of detail. UWE proposes two levels of granularity when modeling Web system requirements. First, a rough description of the functionalities is produced, which are modeled with UML use cases. In a second step, a more detailed description of the use cases is developed, e.g., by UML activity diagrams that depict the responsibilities and actions of the stakeholders.

**Overview of Use Cases** Use case diagrams are built with the UML elements Actor and UseCase. Actors are used to model the users of the Web

system. Typical users of Web systems are the anonymous user (called User) in the MUC case study, the registered user (RegisteredUser) and the Web system administrator. Use cases are used to visualize the functionalities that the system will provide. The use case diagram depicts use cases, actors, and associations among them showing the roles the actors play in the interaction with the system, e.g., triggering some use cases.

In addition to the UML features, UWE distinguishes between three types of use cases: navigation, process and personalized use cases. Navigation use cases are used to model typical user behavior when interacting with a Web application, such as browsing through the Web application content or searching information by keywords. The use case model of Fig. 7.2 for example includes the «navigation» (□) use cases ViewMovie, Search and GoToExternalSite. Pro-

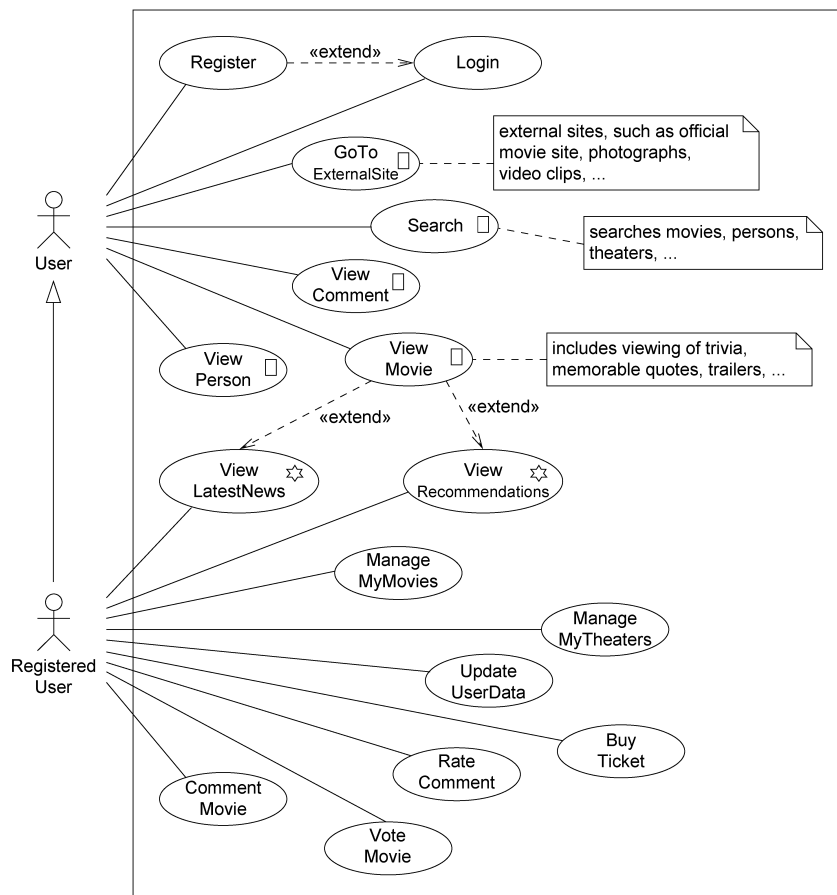


Figure 7.2. UWE use case model for MUC



cess use cases are used to describe business tasks that end users will perform with the system; they are modeled in the same way as it is done for traditional software. These business tasks normally imply transactional actions on the underlying data base. We use “pure” UML notation for their graphical representation. Typical examples for business use cases are Register, CommentMovie, and BuyTicket. A third group of use cases are those that imply personalization of a Web system, such as ViewRecommendations and ViewLatestNews. They are denoted by a stereotype `<<personalized>>` (☆). Personalization is triggered by user behavior.

All UML elements for modeling use case diagrams are available, such as system boundary box, package, generalization relationship, stereotyped dependencies `<<extend>>` and `<<include>>` among use cases. Figure 7.2 illustrates the use case diagram for the MUC case study restricted to the functional requirements from the User and RegisteredUser viewpoint.

**Detailed View of Use Cases** The level of detail and formality of requirements specifications depends on project risks and complexity of the Web application to be built. But very often a specification based only on use cases is not enough (Vilain et al., 2000). Analysts use different kinds of refinement techniques to obtain a more detailed specification of the functional requirements, such as workflows, formatted specifications, or prototypes. These representations usually include actors, pre- and post-conditions, a workflow description, exceptions and error situations, information sources, sample results, and references to other documents. In particular, for the development of Web systems the informational, navigational, and process goals have to be gathered and specified. Informational goals indicate content requirements. Navigational goals point towards the kind of access to content, and process goals specify the ability of the user to perform some tasks within the Web system (Pressman, 2005).

Following the principle of using UML whenever possible for the specification, we refine requirements with UML activity diagrams. For each non-trivial business use case we build at least one activity diagram for the main stream of tasks to be performed in order to provide the functionality indicated by the corresponding use case. Optionally, additional diagrams can be depicted for exceptions and variants. Activity diagrams include activities, shareholders responsible for these activities (optional) and control flow elements. They can be enriched with object flows showing relevant objects for input or output of those activities.

Figure 7.3 illustrates the activity diagram for the use case BuyTicket of our MUC case study. The UWE profile includes a set of stereotypes adding Web specific semantics to UML activity and object nodes. For example, a distinction is made between the objects that define content, nodes of the application,

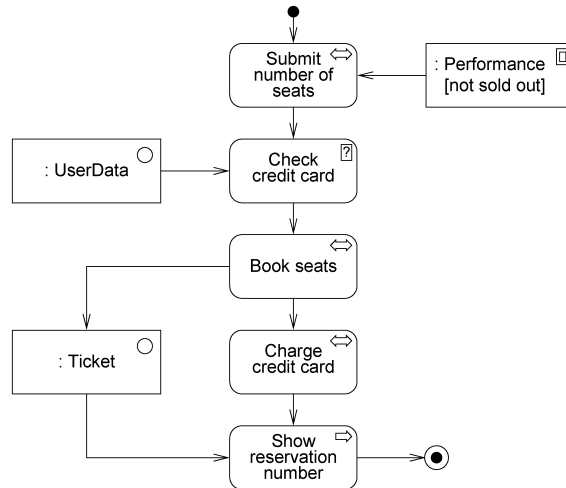


Figure 7.3. MUC case study: UWE activity diagram detailing the buy-ticket use case

and presentation elements. Visualization is improved by the use of the corresponding icons: ○ for «content», □ for «node», and ◻ for Web user interface («WebUI»). Stereotypes of activities are used to distinguish possible actions of the user in the Web environment: browse, search, and transactional activities that comprise changes in at least one database. To this category of stereotypes belong: ⇨ for «browse», ◻ for «query», and ⇔ for transactional actions.

## 7.2.2 Defining the Content

Analysis models provide the basis for the design models, in particular the *content model* of a Web system. The aim of the content model is to provide a visual specification of the domain relevant information for the Web system that mainly comprises the content of the Web application. However, very often it includes also entities of the domain required for customized Web applications. These entities constitute the so-called *user profile* or *user model*. Customization not only deals with adaptation to the properties of users or user groups, but also with adaptation to features of the environment. A so called *context profile* or *context model* is built in such a case. The objects occurring in the detailed view of the use cases provide natural candidates of domain entities for the content and user model.

The separation of content and user model (or context model) has proven its value in practice. Both are graphically represented as UML class diagrams. The content model of MUC is depicted in Fig. 7.4; the user model is shown in Fig. 7.5. The entities representing content and respectively user or context properties are modeled by classes, i.e. instances of the UML metaclass Class.

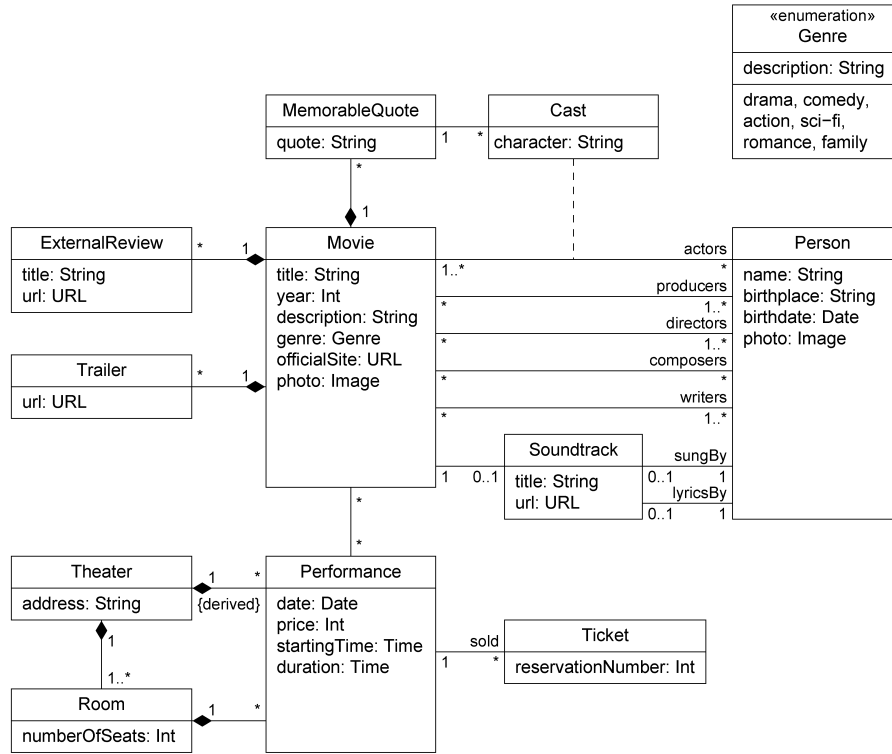


Figure 7.4. MUC case study: Content model

Relationships between content and user properties are modeled by UML associations. In particular, movies are modeled by a class *Movie* with a set of properties, such as title and genre forming the attributes of the class *Movie*, or as classes associated to *Movie* like *Trailer* and *ExternalReview*. Stakeholders of the film production, e.g. a movie’s producer, composer, and cast, are modeled as roles of associations to class *Person*. Note that *Performance* and *Ticket* were inferred from the activity diagram in Fig. 7.3.

The user model contains the user data (see again Fig. 7.3) needed for the login of the user and the comments and rating of the movies. All these data are provided by the users themselves during registration or use of the Web application. In addition, the system collects information on users by observing their behavior. The collected data is used for adaptation and is modeled as a cross-cutting aspect and woven into the user model and other parts of the system (see Sect. 7.2.6 on aspect-oriented modeling of adaptivity).

There is no need for the definition of additional elements as there is no distinction to modeling of non-Web applications. We use “pure” UML notation and semantics. All the features provided by the UML specification for con-

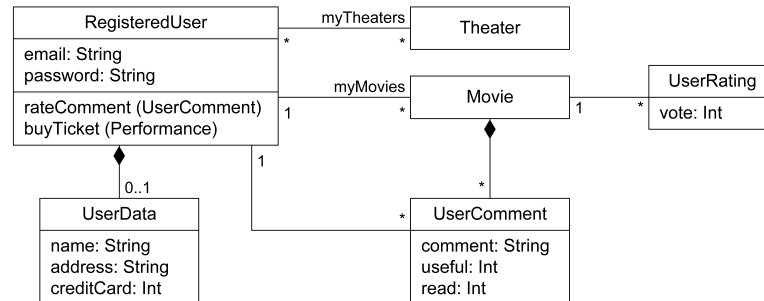


Figure 7.5. MUC case study: User model

structuring class diagrams can be used, in particular, packages and enumerations (e.g. Genre in Fig. 7.4) and relationships like generalizations, compositions, or association classes (e.g. Cast in Fig. 7.4).

### 7.2.3 Laying Down the Navigation Structure

Based on the requirement analysis and the content modeling, the *navigation structure* of a Web application is modeled. Navigation classes (visualized as  $\square$ ) represent navigable nodes of the hypertext structure; navigation links show direct links between navigation classes. Alternative navigation paths are handled by  $\langle\langle\text{menu}\rangle\rangle$ s ( $\boxplus$ ). Access primitives are used to reach multiple instances of a navigation class ( $\langle\langle\text{index}\rangle\rangle$   $\boxminus$ , or  $\langle\langle\text{guided tour}\rangle\rangle$   $\boxrightarrow$ ), or to select items ( $\langle\langle\text{query}\rangle\rangle$   $\boxdot$ ). In Web applications that contain business logic the business processes must be integrated into the navigation structure. The entry and exit points of the business processes are modeled by process classes ( $\Sigma$ ) in the navigation model, the linkage between each other and to the navigation classes is modeled by process links. Each process class is associated with a use case which models a business process. Navigation structures are laid down in stereotyped UML class diagrams with navigation and process classes, menus, and access primitives extending the UML metaclass Class, and navigation and process links extending the UML metaclass Association.

**Initial Navigation Structure** UWE provides methodological guidelines for developing an initial sketch of the navigation structure from the content model of a Web application (see also Koch and Kraus, 2002; Knapp et al., 2003): Content classes deemed to be relevant for navigation are selected from the content model and these classes as well as their associations are put into a navigation model as navigation classes and navigation links, respectively. Navigation links represent possible steps to be followed by the user, and thus these links have to be directed; if navigation back and forth between two navigation

classes is desired, an association is split into two. Menus are added to every navigation class that has more than one outgoing association. Finally, access primitives (index, guided tours and queries) allow for selecting a single information entity, as represented by a navigation class. An index, a guided tour or a query should be added between two navigation classes whenever the multiplicity of the end target of their linking association is greater than one. The properties of the content class corresponding to the navigation class over which the index or the query runs are added as navigation attributes to the navigation class.

The result of applying these steps of the UWE method to the content model of the MUC case study in Fig. 7.4 is shown in Fig. 7.6. From the home page

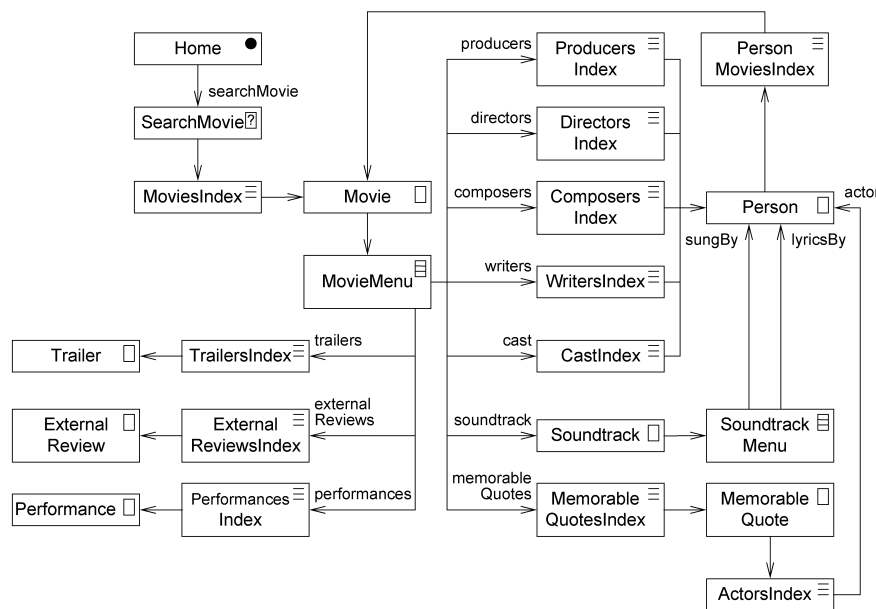


Figure 7.6. MUC case study: Navigation from Movie (fragment)

Home the user can, by means of a query SearchMovie, search for movies of his interest by criteria like movie name, actors, or directors etc. Soundtrack is directly reachable through MovieMenu as there may be at most one soundtrack for each movie whereas there may be several directors which have to be selected from DirectorsIndex. As an example for a bidirectional linkage between navigation classes, the actors of a movie can be selected from CastIndex reaching a Person, where conversely all movies which this person has contributed to can be chosen from. The navigation structure has been refined by adding a home node (●) as the initial node of the MUC Web application, as well as a main menu.

The UWE profile notation for menus and access primitives provides a compact representation of patterns frequently used in the Web domain. Fig. 7.7(b) shows the shorthand notation for indexes. Using “pure” UML for modeling an index would instead, require an additional model element: an index item as depicted in Fig. 7.7(a). The result would be an overloaded model if it contains many of such indexes.

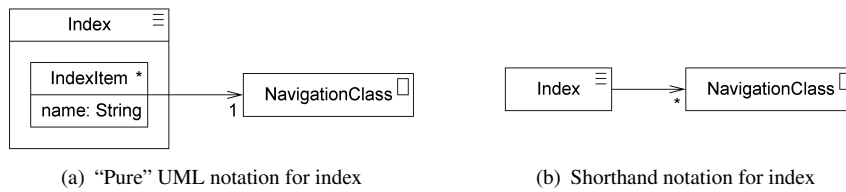


Figure 7.7. UWE Profile: Index

**Adding Business Processes** In a next step, the navigation structure can now be extended by process classes which represent the entry and exit points to business processes. These process classes are derived from the non-navigational use cases. In Fig. 7.8, the business processes Register (linked to the use case Register) and Login (linked to the use case Login) have been added. The integration of these classes in the navigation model requires an additional menu (MainMenu) which provides links to Register, Login and SearchMovies. A user may only manage his movies, if he has logged in previously. Finally, a user can buy tickets for a selected movie and a selected performance by navigating to BuyTicket.

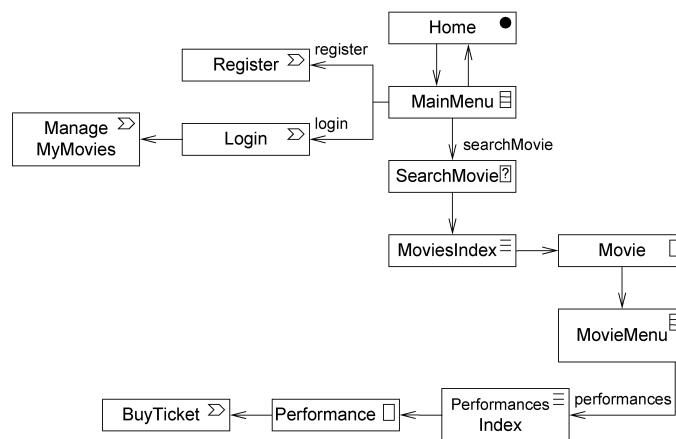


Figure 7.8. MUC case study: Integration of business processes into navigation (fragment)

A single navigation structure diagram for a whole Web application would inevitably lead to cognitive overload. Different views to the navigation structure should be produced from the content model focusing on different aspects of the application, like navigation to particular content or integration of related business processes.

## 7.2.4 Refining the Processes

Each process class included in the navigation model is refined into a process model consisting of a process flow model and optionally of a process structure model. The control and data flow is modeled in the process flow model in the form of an UML activity diagram. It is the result of a refinement process that starts from the workflow in the requirements model. Figure 7.9 illustrates

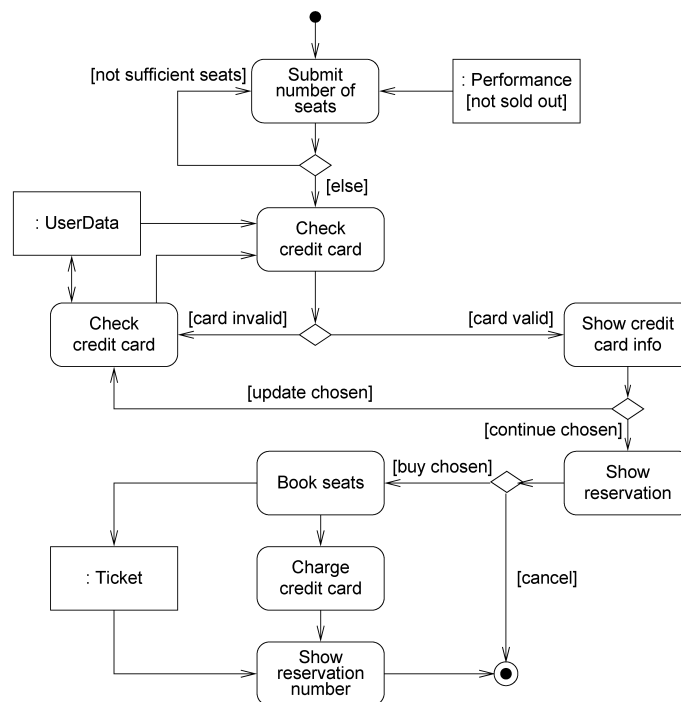


Figure 7.9. MUC case study: UWE process flow model for the buy-ticket process

the result of the refinement process applied to Fig. 7.3. This process mainly consists in the integration of the main stream of the actions with alternatives, such as Enter new credit card info in case of invalid card numbers or exception handling (not included in this example). Control elements are added with the purpose of providing the business logic. Activities and objects can be added to the activity diagram. A process structure model has the form of a class diagram

and describes the relationship between a process class and other classes whose instances are used to support the business process.

### 7.2.5 Sketching the Presentation

The presentation model provides an abstract view on the user interface (UI) of a Web application. It is based on the navigation model and abstracts from concrete aspects of the UI, like the use of colors, fonts, and where the UI elements are placed on the Web page; instead, the presentation model describes the basic structure of the user interface, i.e., which UI elements (e.g. text, images, anchors, forms) are used to present the navigation nodes. The advantage of the presentation model is that it is independent of the actual techniques used to implement the Web site, thus allowing the stakeholders to discuss the appropriateness of the presentation before actually implementing it.

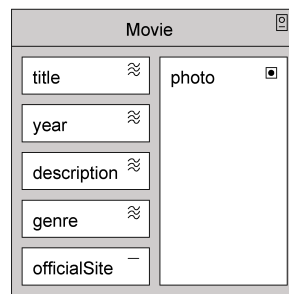


Figure 7.10. MUC case study: Presentation class Movie

The basic elements of a presentation model are the presentation classes which are directly based on nodes from the navigation model, i.e. navigation classes, menus, access primitives, and process classes. A presentation class (☐) is composed of UI elements, like text (≍), anchor (—), button (●), image (▣), form (☒), and anchored collection (☷). Figure 7.10 shows an example of a presentation class for the navigation class Movie. Note that to ease the identification of which navigation node is presented by a presentation class, the presentation class uses by default the same name as the corresponding navigation node. Each attribute of a navigation class is presented with an appropriate UI element. For example, a text element is used for the title attribute and an image element is used for the photo attribute. The relationship between presentation classes and UI elements is that of composition. For presentation models, composition is pictured by drawing the component, i.e. the UI element, inside the composite, i.e. the presentation class; note, however, that this notation is not supported by all CASE tools.



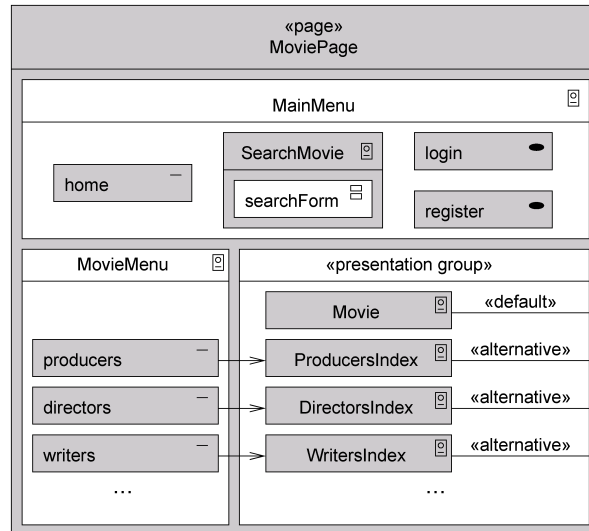


Figure 7.11. MUC case study: The presentation model of the movie page

Usually, the information of several navigation nodes is presented on one Web page, which is modeled by pages (`«page»`) in UWE. Pages can contain, among others, presentation classes and presentation groups (`«presentation group»`). A presentation group can itself contain presentation groups and presentation classes. An excerpt of the presentation model of the movie page is shown in Fig. 7.11. It contains a presentation class for the main menu, which in turn contains a link (represented by the anchor UI element) to home, a presentation class for the SearchMovie query, and button UI elements to start the login and registration processes. The SearchMovie query also provides an example of the form UI element to enter the movie name to search for. The presentation class for MovieMenu contains links to the presentation classes of the corresponding indexes – based on the navigation model in Fig. 7.6 – providing additional information on the movie. The presentation classes of these indexes plus the presentation classes for movie are assembled in a presentation group. The use of the stereotypes `«default»` and `«alternative»` for the associations from Movie, ProducersIndex, etc. to MovieMenu indicates that the elements of the presentation groups are alternatives, i.e., only one of them is shown depending on which link was followed from the movie menu, with the presentation class Movie being shown by default. For example, when the user follows the producers link in the MovieMenu, the ProducersIndex is shown, containing the list of the producers of that film.

## 7.2.6 Aspect-Oriented Modeling of Adaptivity

Adaptivity is an increasingly important feature of Web applications. Adaptive Web applications provide more appropriate pages to the user by being aware of user or context properties. An example of adaptivity are recommendations based on user behavior, like movie of favorite actors in our MUC case study. In general, adaptivity is orthogonal to the three views: content, navigation structure, and presentation (see Fig. 7.1). In order to model adaptive features of Web applications non-invasively, we use techniques of Aspect-Oriented Modeling (AOM, cf. Filman et al., 2004) in UWE.

We introduce a new model element named *aspect*. An aspect is composed of a *pointcut* part and an *advice* part. It is a (graphical) statement expressing that additionally to the features specified in the principal model, each model element selected by the pointcut also has the features specified by the advice. In other words, a complete description, including both general system functionality and additional, cross-cutting features of the quantified model elements, is given by the composition of the principal model and the aspect. The process of composition is called *weaving*.

UWE defines several kinds of aspects for modeling different static and runtime adaptivity (Baumeister et al., 2005). In order to model the recommendation feature modularly we use on the one hand a model aspect and a runtime aspect for keeping track of the number of visits of movies pages. On the other hand, another runtime aspect integrates the recommendation feature into the login process: A list of movies is presented ranked according to the appearing actors, which in turn are ranked according to their relevance in the visited movies.

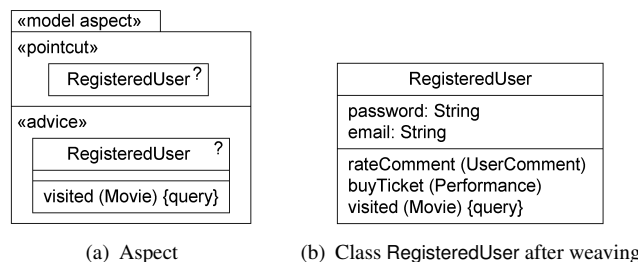


Figure 7.12. MUC case study: Model aspect

The static model aspect for extending the user model (see Fig. 7.5) by an operation which returns the number of visits of a registered user to a movie page is shown in Fig. 7.12(a). The pointcut is a pattern containing a special element, the *formal parameter*, which is annotated by a question mark. The pointcut selects all model elements in the base model that match the pattern thereby instantiating the formal parameter. In our case the formal parame-

ter is a class of which only the name `RegisteredUser` is specified. The pointcut therefore selects all classes (actually, there is exactly one such class) in the navigation model with the name `RegisteredUser`. The advice defines the change to the selected model elements. After weaving, our `RegisteredUser` class is thus extended by the operation `visited`, see Fig. 7.12(b); no other elements are affected by this aspect. Model aspects are a special case of aspect-oriented class diagrams (AOCDs), which are also defined in a lightweight UML extension and therefore UML compatible, see (Zhang, 2005). Since a model aspect specifies a static modification of the base model, other, standardized model transformation languages such as the Atlas Transformation Language (ATL [Jouault and Kurtev, 2005]), QVT-P (QVT-Partners, 2003), or QVT (QVT-Merge Group, 2004) may also be used. The advantage of AOCD compared with these languages is, however, that it does not require the modeler to have expert knowledge of the UML metamodel, which may make AOCD easier to use (cf. Sect. 7.4).

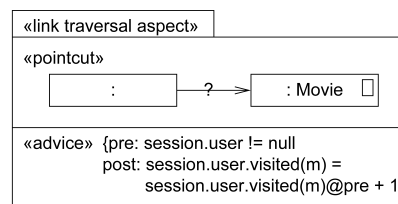


Figure 7.13. MUC case study: Link traversal aspect for counting movie visits

The dynamic behavior of our MUC system is extended by two runtime aspects. Figure 7.13 shows a link traversal aspect, used to ensure that `visited` returns the correct result: the pointcut selects all links from any object – note that neither the name nor the type of the object to the left is specified and thus it matches any object – to some `Movie` object. The advice defines with an OCL constraint the result of the action fired when such a link is visited: if the current user is logged in, the system increases his respective record by one. After weaving, the behavior of the system is thus enriched by counting user visits to the movie pages.

Figure 7.14 shows how the business process `Login` is extended by a flow aspect. The base model depicted in Fig. 7.14(a) defines the normal workflow without considering adaptivity: the user is asked to input his email address and password, then the system verifies the input and responds accordingly. The adaptive feature of generating recommendations for the user is added by the aspect shown in Fig. 7.14(b). The pointcut selects every (in this concrete example, exactly one) control flow edge from a decision point to the `OK` action, which is guarded by the condition `valid`. The advice deletes this edge by

crossing it out and adds an action for recommendation generation and two new control flow edges to bind it into the process.

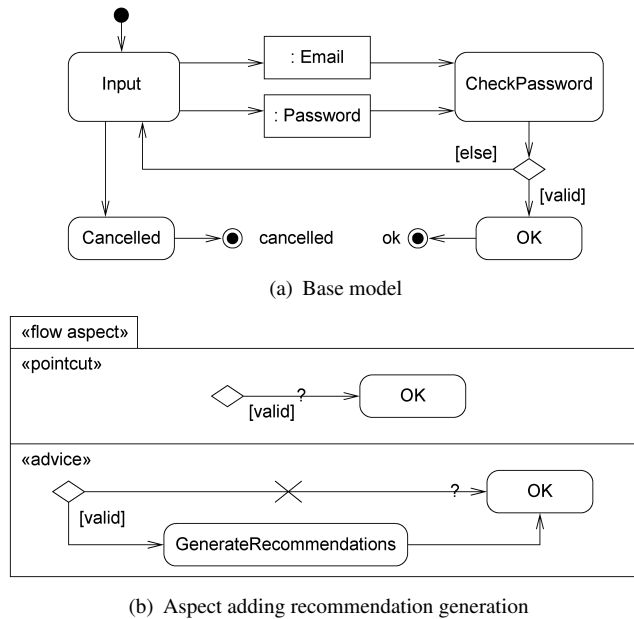


Figure 7.14. MUC case study: Flow aspect extending Login

### 7.3 UWE Metamodel

The UWE metamodel is defined as a conservative extension of the UML 2.0 metamodel. Conservative means that the model elements of the UML metamodel are not modified. Instead, all new model elements of the UWE metamodel are related by inheritance to at least one model element of the UML metamodel. We define additional features and relationships for the new elements. Analogous to the well-formedness rules in the UML specification, we use OCL constraints to specify the additional static semantics of these new elements. The resulting UWE metamodel is profileable, which means that it is possible to map the metamodel to a UML profile (Koch and Kraus, 2003). In particular, UWE stays compatible with the MOF interchange metamodel and therefore with tools that are based on the corresponding XML interchange format XMI. The advantage is that all standard UML CASE tools which support UML profiles or UML extension mechanisms can be used to create UWE models of Web applications. If technically possible, these CASE tools can further be extended to support the UWE method. ArgoUWE, see Sect. 7.5, presents an instance of such CASE tool support for UWE based on the UWE metamodel.

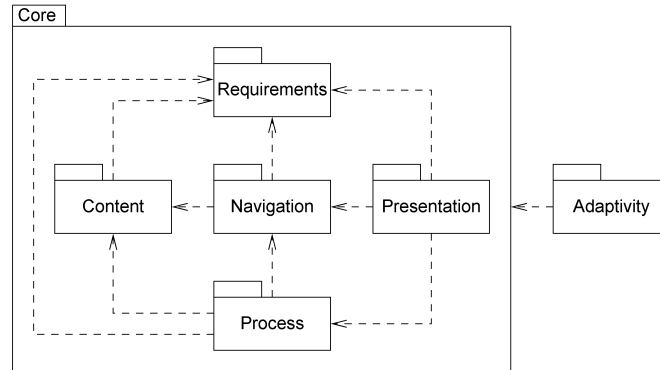


Figure 7.15. Overview of the UWE metamodel

The UWE extension of the UML metamodel consists of adding two top-level packages Core and Adaptivity to the UML, cf. Fig. 7.15. The separation of concerns of Web applications is reflected by the package structure of Core, the cross-cutting of adaptation by the dependency of Adaptivity on Core (see Fig. 7.1). The package Requirements comprises the UWE extensions on UseCase for discerning navigational from business process and personalized use cases and the different markings for ActivityNode (`<<browse>>`, `<<query>>`, and `<<transaction>>`) and ObjectNode (`<<content>>`, `<<node>>`, and `<<WebUI>>`), see Escalona and Koch, 2006. The navigation and presentation packages bundle UWE's extensions for the corresponding models. Figure 7.16 details a part of the metamodel for Navigation with the connection between Node and Link and their various subclasses. NavigationClass and ProcessClass with the related NavigationLink and ProcessLink as well as Menu and the access primitives Index, GuidedTour and Query provide the Web domain specific metaclasses for building the navigation model. Packages Contents and Process are currently only used as a stub, reflecting the fact that UWE allows the designer to develop content and process models using all UML features. Finally, Adaptation contains UWE's aspect facilities by representing Aspect as a UML Package with two subpackages Pointcut and Advice.

In order to transfer the UWE metamodel into a UML profile we use UML's extension mechanisms (see Sect. 7.1). Figure 7.17 shows how the metaclasses of the UWE navigation metamodel are rendered as a stereotype hierarchy, forming the UWE navigation profile: Node becomes a stereotype of Class, NavigationAttribute a stereotype of Property, and Link a stereotype of Association. The associations of the UWE navigation metamodel, e.g., connecting Link to Node cannot be represented by meta-associations (see Object Management Group, 2005) and have to be added either by stereotyping the UML metaclass Dependency or by using the association from

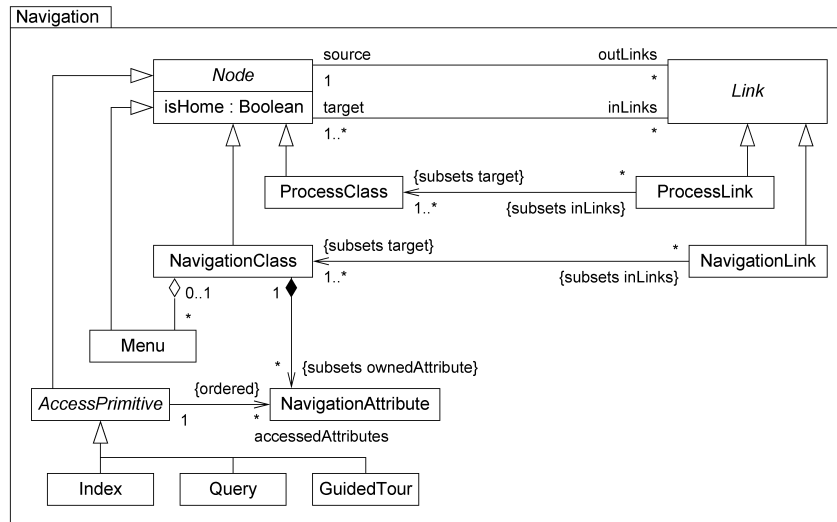


Figure 7.16. UWE navigation metamodel

the UML metamodel from which the association is derived. The latter approach is used for representing the composition between `NavigationClass` and `NavigationAttribute` using the association `ownedAttributes`; for the association between `AccessPrimitive` and `NavigationAttribute` and the association between `NavigationClass` and `Menu` we stereotype `Dependency` leading, e.g., to the following constraint:

```
context Dependency
inv: self.stereotypes->
    includes("Primitive2Attribute") implies
    (self.client.stereotypes->
        includes("AccessPrimitive") and
        self.supplier.stereotypes->
            includes("NavigationAttribute"))
```

where `client` and `supplier` denote the ends of the `Dependency` relationship.

**Consistency Rules** Following the UML, we use OCL to state more precisely the static semantics of UWE's new metamodel elements as well as the dependencies of metamodel elements both inside a single metamodel package and between packages. As an example, the following constraint states that every use case which is neither a navigation nor a personalized use case needs a process class and that the converse direction holds as well (cf. Fig. 7.18):

```
context ProcessClass
inv: not self.useCase.ocIsKindOf(NavigationUseCase) and
```

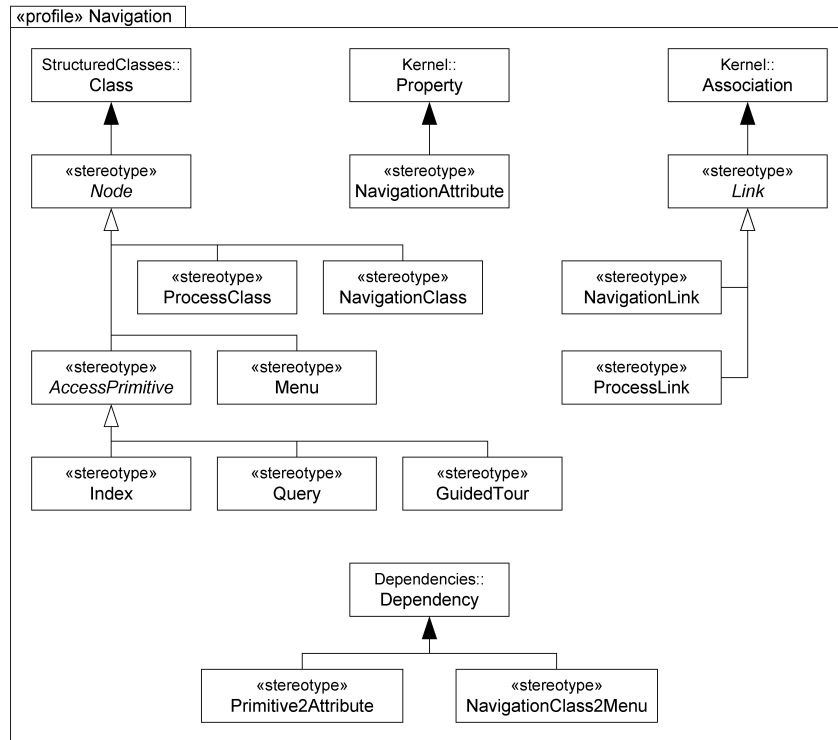


Figure 7.17. UWE navigation profile

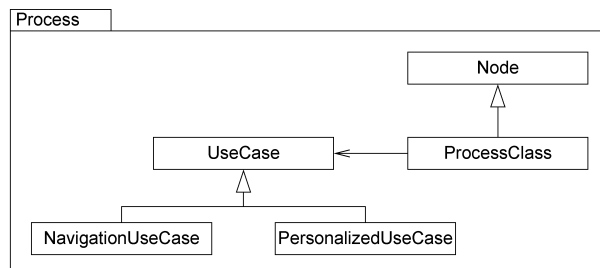


Figure 7.18. UWE process metamodel

```

not self.useCase.oclIsKindOf(PersonalizedUseCase)

context UseCase
inv: (not self.oclIsKindOf(NavigationUseCase) and
not self.oclIsKindOf(PersonalizedUseCase)) implies
ProcessClass.allInstances()->
exists(pn | pn.useCase = self)
    
```

## 7.4 Model-Driven Development in UWE

The UWE approach includes the specification of a process for the development of Web systems in addition to the UML profile and the UWE metamodel. The UWE process is model-driven following the MDA principles and using several other OMG standards, like MOF, UML, OCL, and XMI, and forthcoming standards, like QVT (QVT-Merge Group, 2004). The process relies on modeling and model transformations, and its main characteristic is the systematic and semi-automatic development of Web systems as detailed in this book in the chapter on model-driven Web engineering by N. Moreno et al. The aim of such an MDD process is automatic model transformation which in each step is based on transformation rules.

Focusing on model transformations the UWE process is depicted in Fig. 7.19 as a stereotyped UML activity diagram (Meliá et al., 2005). Models are shown as objects, and transformations are represented with stereotyped activities (special circular icon).

The process starts with the business model, which MDA calls computational independent model (CIM), used to specify the requirements. Platform-independent models (PIMs) are derived from these requirement models. The set of design models represents the different concerns of the Web applications comprising: the content, the navigation, the business processes, the presentation, and the adaptation of the Web system (summarized as FunctionalModels in Fig. 7.19). In a next step, the different views are integrated into a “big picture” model of the Web systems, which can be used for validation (Knapp and Zhang, 2006) and also for generation of platform-dependent models (see below). A merge with architectural modeling features, either of the “big-picture model” or the design models directly, results in an integrated PIM covering functional and architectural aspects. Finally, the platform-specific models (PSMs) derived from the integration model are the starting point for code generation.

### 7.4.1 Transformations from Requirements to Functional Models

The overall objective of modeling the requirements is the specification of the system as a CIM and to provide input for the construction of models in the other development phases (see Figs. 7.1, 7.19, and Sect. 7.2). In particular, specific objectives for Web systems are the specification of content requirements, the specification of the functional requirements in terms of navigation needs and business processes, the definition of interaction scenarios for different groups of Web users, and if required, the specification of personalization and context adaptation. The first model transformation step of the UWE process consists of mapping these Web system requirements models to the UWE



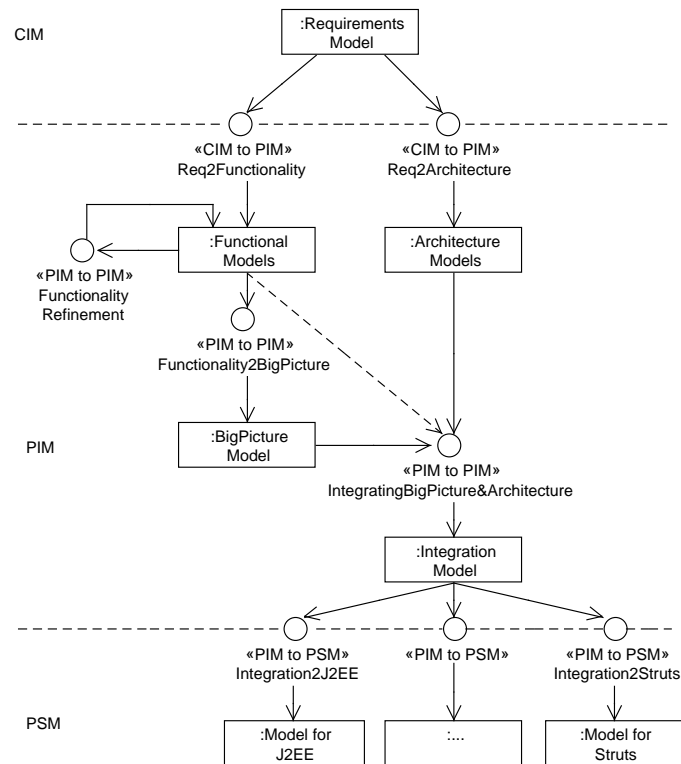


Figure 7.19. Overview of model transformations in the UWE process

functional models. Transformation rules are defined therefore as mappings from the requirements metamodel package to the content, navigation, presentation, process, and adaptivity packages of the metamodel. How these packages depend on each other is shown in Fig. 7.15.

For example, UWE distinguishes in the requirements model between different types of navigation functionality: browsing, searching, and transactional activities. Browse actions can be used to enforce the existence of a navigation path between source and target node. An action of type search indicates the need of a query in the navigation model in order to allow for user input of a term and the system responding with a resulting set matching this term (see Sect. 7.2.1).

Figure 7.20 shows the Search2Query transformation rule specified in QVT's graphical notation (QVT-Merge Group, 2004). Source and target of the transformation is the UWE metamodel defined as *checkonly* and *enforce*, respectively (identified with an "c" and "e" in Fig. 7.20). For each search with con-

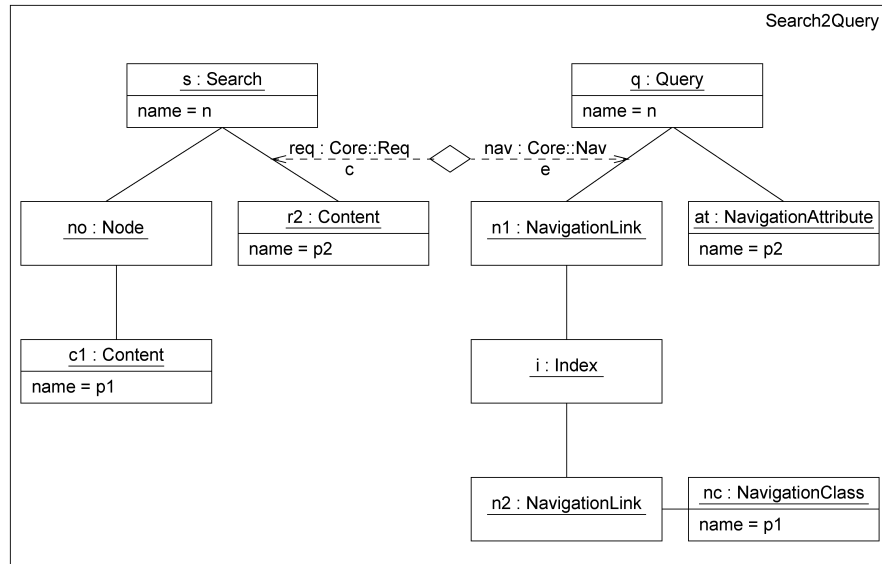


Figure 7.20. Transformation rule Search2Query

tent p2 in the requirements model, a query in the navigation model is generated with an associated navigation attribute p2. For the associated node object in the requirements model, an index and objects of a navigation class, as well as corresponding links will be generated.

For more details about the UWE metamodel for Web requirements we refer the reader to Escalona and Koch, 2006. A detailed description of the transformation rules between CIMs and PIMs for the functional aspects of Web applications has been presented in Koch et al., 2006. A metamodel of the non-functional requirements for Web applications and mappings of non-functional requirements to architectural model elements are subject to future work.

## 7.4.2 Refinement of Functional Models

The transformations for refining the functional models comprise mappings from content to navigation model, refinements of the navigation model, and from the navigation into the presentation model. In UWE, an initial navigation model is generated based on classes of the content model marked as navigation relevant (see Sect. 7.2.3). This generation step can be rendered as a transformation Content2Navigation. From a single content model different navigation views can be obtained, e.g., for different stakeholders of the Web system like anonymous user, registered user, and administrator. The generation of each navigation view requires a set of marks on elements of the content model which form a so-called marking model kept separately from the content model. The

development process cannot be completed in an entirely automatic way, as the designer has to take the decision about the “navigation relevance” marks; the Content2Navigation transformation is applied once the marks have been set.

Conversely, the remaining transformation steps for navigation models mentioned in Sect. 7.2.3 are turned into transformation rules that can be applied fully automatically. These rules include for example the insertion of indexes and menus. Presentation elements are generated from navigation elements. For example, for each link in the navigation model an appropriate anchor is required in the presentation model. The main difficulty is the introduction of the look and feel aspects.

All these transformations are defined as OCL constraints (by preconditions and postconditions) in UWE and are implemented in Java in the CASE tool ArgoUWE.

### 7.4.3 Creation of Validation and Integration Models

The UWE MDD process comprises two main integration steps: the integration of all functional models and the integration of functional and non-functional aspects; the latter integration step is related to architectural design decisions.

The aim of the first step is the creation of a single model for validating the correctness of the different functional models and that allows seamless creation of PSMs. This “big picture” model is a UML state machine, representing the content, navigation structure, and the business processes of the Web application as a whole (presentation aspects will be added in the future). The state machine can be checked by the tool Hugo/RT (Knapp et al., 2002) – a UML model translator for model checking, theorem proving, and code generation.

The transformation rules Functional2BigPicture are defined based on a meta-model graph transformation system. For the implementation of the graph transformation rules any (non-Web specific) tool for graph transformations can be used. An example of the graph transformation of a navigation node to a state of the validation model is sketched in Fig. 7.21.

The aim of the second step is the merge of the validation model elements with information on architectural styles. Following the WebSA approach (Meliá et al., 2005) we propose to merge functional design models and architecture models at the PIM level. For example, the elements of the WebSA models provide a layer-view and a component-view of the architecture, which are also specified as PIMs. Transformation rules are defined based on the UWE and WebSA metamodels.

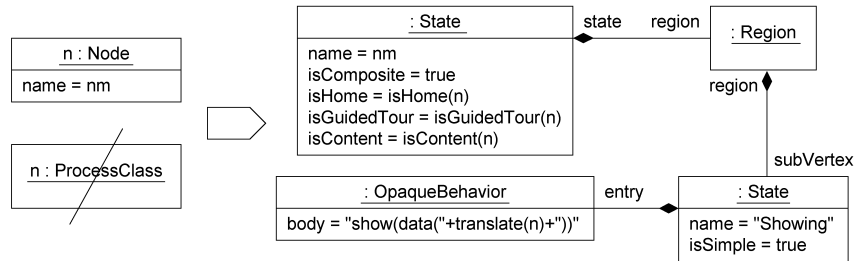


Figure 7.21. Transformation rule Node2State

#### 7.4.4 Generation of Models and Code for Specific Platforms

In order to transform PIMs into PSMs additional information of the platform is required. It can be provided as an additional model or it is implicitly contained in the transformations. For mappings from UWE design models (PIMs) to PSMs for Web applications we tested different model transformation languages. The query-view-transformation languages we use are ATL (Jouault and Kurtev, 2005), QVT-P (QVT-Partners, 2003), and QVT (QVT-Merge Group, 2004). For example, the following QVT-P transformation tackles the generation of J2EE elements from Java server pages of the integration model.

```

relation ServerPage2J2EE {
  domain { (IM.IntegrationModel)
    [(ServerPage)
      [name = nc,
        services = { (WebService) [name = on,
                              type = ot] },
        views = { (View) [name = vn] }]] }
  domain { (JM.J2EEModel)
    [(JavaServerPage)
      [name = nc,
        forms = { (Form) [name = on,
                          type = ot] },
        beans = { (JavaClass) [name = vn] }]] }
  when { services->forall(s |
    WebService2Form(s, Flset.toChoice()))
    views->forall(v |
    View2Bean(v, J1set.toChoice())) }
}

```

The ATL code below exemplifies a transformation rule that maps the element Anchor of the UWE presentation model to a JSP element. Note that the transformation rule also involves elements of the navigation model (NavigationLink).

```

rule Anchor2JSP {
  from
    uie : UWE!Anchor
      (not uie.presentationClass.oclIsUndefined() and
       not uie.navigationLink.oclIsUndefined())
  to
    jsp : JSP!Element
      (name <- 'a',
       children <- Sequence { hrefAttribute,
                              contentNode }),
    hrefAttribute : JSP!Attribute
      (name <- 'href',
       value <- thisModule.createJSTLURLExpr
         (uie.navigationLink.target.name, 'objID')),
    contentNode : JSP!TextNode
      (value <- uie.name)
}

```

## 7.5 CASE Tool ArgoUWE

We have extended the CASE tool ArgoUML into a tool for UWE-based Web application development, called ArgoUWE (Knapp et al., 2003, [www.pst.ifi.lmu.de/projekte/argouwe](http://www.pst.ifi.lmu.de/projekte/argouwe)). We decided to extend ArgoUML as it is a feature-rich, open-source tool and offers a plugin architecture. The drawback of this decision is that the UWE metamodel can not be used directly since ArgoUML is based on UML 1.3/4. However, a UML 1.x compatible profile can easily be derived from the UWE metamodel along the same lines as sketched in Sect. 7.3.

ArgoUML provides support for designing Web applications in the phases of requirements elicitation and content, navigation, business process as well as presentation modeling. It provides not only tailored editors for UWE diagrams, but also semi-automatic model transformations defined in the UWE development process. As these model transformations are based on the UWE metamodel, the tool ensures both consistency between the different models and integrity of the overall Web application model with respect to UWE's OCL constraints. ArgoUWE fully integrates the UWE metamodel (Koch and Kraus, 2003) and provides XMI export, and thus facilitates data transfer with other UML-compliant tools. Design deficiencies, such as violations of the OCL constraints, are reported by an extension of the cognitive design critics of ArgoUML and can also be checked upon request (see Sect. 7.5.2).

Working with ArgoUWE is intuitive for ArgoUML users, as ArgoUWE makes use of the graphical interface of ArgoUML. In particular, the UML model elements and diagrams are structured in a tree view in the *explorer* ((1) in Fig. 7.22); the diagrams are edited in the *editor pane* (2); to-do items of the designer are listed in the *to-do pane* (3); tagged values, constraints, and doc-

umentation of the currently selected model as well as automatically generated code skeletons are shown in the *details pane* (4).

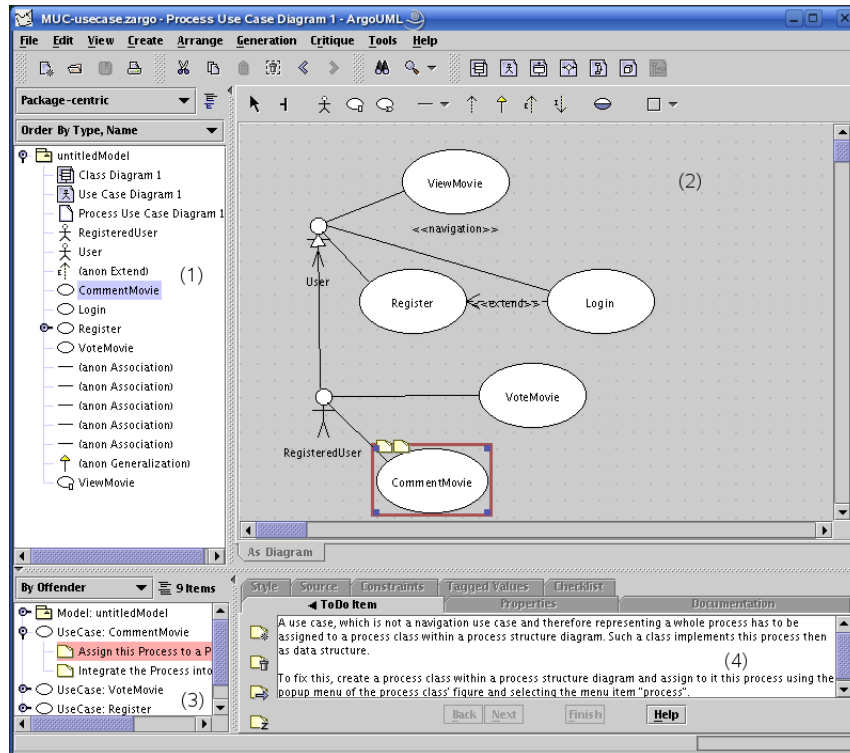


Figure 7.22. MUC case study: ArgoUWE-screenshot of a fragment of the use case model

### 7.5.1 Model Transformations

ArgoUWE implements some of the aforementioned model transformations as semi-automatic procedures.

- In the content model, the designer may mark classes as navigation relevant. ArgoUWE can then generate an initial navigation model by creating for each navigation relevant class a navigation class and for each association between navigation relevant classes a link between the corresponding navigation classes.
- In the navigation model, ArgoUWE can add indexes and menus automatically. The designer may add queries and guided tours between navigation nodes manually or, alternatively, by selecting a generated index and changing it into a query or a guided tour.

- From the navigation model, ArgoUWE can generate a first draft of a presentation model. For each navigation class and each of its attributes a presentation class is created. The presentation classes of attributes are associated to those of the navigation classes by composition.

The generation of Web applications from the presentation model is out of scope for ArgoUWE. This is done either by hand by the Web designer or semi-automatically by using frameworks for the implementation of Web applications, such as Struts ([www.struts.apache.org](http://www.struts.apache.org)).

### 7.5.2 Model Consistency

An important requirement of any CASE tool is to support the modeler to keep his models consistent. Upon model inconsistency, the tool may either interrupt the modeler and force him first to correct it before continuing modeling, or simply give a warning. We implemented ArgoUWE to do the latter since we believe that the usability of the modeler being warned yet not interrupted outweighs the drawback of the model being inconsistent for a short time. Moreover, the ArgoUML feature of design critics provides an excellent starting point for the implementation of the non-interruptive warnings for UWE models.

The “cognitive design critics” of ArgoUML is one of its distinguishing features compared to other modeling tools (cf. Robbins, 1999). During run time, a thread running in the background keeps checking if the current model shows deficiencies. For each deficiency found, a design critique item is created and added to the to-do pane. Design critics not only warn the user that his design may be improved but can also, by means of a wizard, lead to a better design. The design critique items range from incompleteness, such as unnamed model elements, to inconsistency, such as name collisions of different attributes or operations in a class. Furthermore, design critics also suggest the use of certain design patterns (Gamma et al., 1995). The issues of design critics can be sorted by several criteria like priority or the model element causing the design critique. Design critiques are only warnings and do not interrupt the designer.

ArgoUWE inherits the feature of design critics from ArgoUML. In fact, all well-formedness constraints of UWE have been fully integrated and are continuously checked by ArgoUWE in the background at runtime. In Fig. 7.22 the highlighted design critique indicates that the use case `CommentMovie` does not show a corresponding process class yet; this critique corresponds to the metamodel constraints shown in Sect. 7.3.

## 7.6 Outlook

The UML-based Web Engineering (UWE) approach is continuously evolving. Evolution is due to: improvement of existing features, such as person-

alization of Web systems; adaptation to new technologies, as asynchronous client-server communication; and introduction of new software engineering techniques like aspect-orientation and model-driven principles. The challenge in all these cases is to provide a more intuitive and useful tool for the methodological development of Web systems, to increase Web systems quality and to reduce development time.

The evolution we can currently observe is driven by a set of improvements that are being addressed and a set of extensions we are planning for UWE. The most important are:

- Specification of the transformations (at metamodel level) of (non-functional) requirements to architecture models.
- Implementation of the “weaving” process for the integration of the aspect-oriented features in UWE models.
- Engineering of Rich Internet Applications (RIAs), e.g. Web applications based on asynchronous communication like using AJAX (Garrett, 2005).
- Tool support for transformations from CIM models to PIM models and for the UML 2.0 features in UWE.
- Integration of a QVT engine (when available) in the tool environment.
- Extension of UWE with test models.

Our higher-level goal is the convergence of Web design/development methods. It is the only way to obtain a powerful domain-specific modeling and development language that benefits from the advantages of the different methods. Obviously, there is a trend towards using UML as the common notation language. Some methods are moving from their proprietary notation to a UML compliant one and introduce a UML profile; others define a MOF-based metamodel. It is currently hard to predict how far this converging trend will go and whether it will eventually lead to a “Unified Web Modeling Language”.

## Acknowledgments

Thanks go to Andreas Kraus for providing the ATL transformation rule and fruitful discussions. This work has been partially supported by the project MAEWA “Model Driven Development of Web Applications” (WI841/7-1) of the Deutsche Forschungsgemeinschaft (DFG), Germany and the EC 6th Framework project SENSORIA “Software Engineering for Service-Oriented Overlay Computers” (FET-IST 016004).



## References

- Baresi, Luciano, Garzotto, Franca, Mainetti, Luca, and Paolini, Paolo (2002). Meta-modeling Techniques Meet Web Application Design Tools. In Kutsche, Ralf-Detlef and Weber, Herbert, editors, *Proc. 5<sup>th</sup> Int. Conf. Fundamental Approaches to Software Engineering (FASE'02)*, volume 2306 of *Lect. Notes Comp. Sci.*, pages 294–307. Springer, Berlin.
- Baumeister, Hubert, Knapp, Alexander, Koch, Nora, and Zhang, Gefei (2005). Modelling Adaptivity with Aspects. In Lowe and Gaedke, 2005, pages 406–416.
- Baumeister, Hubert, Koch, Nora, and Mandel, Luis (1999). Towards a UML Extension for Hypermedia Design. In France, Robert and Rumpe, Bernhard, editors, *Proc. 2<sup>nd</sup> Int. Conf. Unified Modeling Language (UML'99)*, volume 1723 of *Lect. Notes Comp. Sci.*, pages 614–629. Springer, Berlin.
- de Troyer, Olga and Leune, Corneli Jan (1998). WSDM: A User Centered Design Method for Web Sites. *Computer Networks*, 30(1–7):85–94.
- Escalona, María José and Koch, Nora (2006). Metamodeling the Requirements of Web Systems. In *Proc. 2<sup>nd</sup> Int. Conf. Web Information Systems and Technologies (WebIST'06)*, Setubal, Portugal.
- Filman, Robert E., Elrad, Tzilla, Clarke, Siobhán, and Aksit, Mehmet, editors (2004). *Aspect-Oriented Software Development*. Addison-Wesley, Reading, Mass., &c.
- Gamma, Erich, Helm, Richard, Johnson, Ralph, and Vlissides, John (1995). *Design Patterns*. Addison-Wesley, Boston, &c.
- Garrett, Jesse James (2005). Ajax: A New Approach to Web Applications. <http://www.adaptivepath.com/publications/essays/archives/000385.php>.
- Gómez, Jaime, Cachero, Cristina, and Pastor, Oscar (2001). Conceptual Modeling of Device-Independent Web Applications. *IEEE Multimedia*, 8(2):26–39.
- Hennicker, Rolf and Koch, Nora (2001). Systematic Design of Web Applications with UML. In Siau, Keng and Halpin, Terry A., editors, *Unified Modeling Language: Systems Analysis, Design and Development Issues*, pages 1–20. Idea Group.
- Isakowitz, Tomás, Stohr, Eduard A., and Balasubramanian, P. (1995). RMM: A Methodology for Structuring Hypermedia Design. *Comm. ACM*, 38(8):34–44.
- Jouault, Frédéric and Kurtev, Ivan (2005). Transforming Models with ATL. In Bruel, Jean-Michel, editor, *Revised Sel. Papers Satellite Events at the MoDELS 2005 Conf.*, volume 3844 of *Lect. Notes Comp. Sci.*, pages 128–138. Springer, Berlin.

- Knapp, Alexander, Koch, Nora, Moser, Flavia, and Zhang, Gefei (2003). ArgoUWE: A CASE Tool for Web Applications. In *Proc. 1<sup>st</sup> Int. Wsh. Engineering Methods to Support Information Systems Evolution (EMSISE'03)*, Genève. 14 pages.
- Knapp, Alexander, Merz, Stephan, and Rauh, Christopher (2002). Model Checking Timed UML State Machines and Collaborations. In Damm, Werner and Olderog, Ernst Rüdiger, editors, *Proc. 7<sup>th</sup> Int. Symp. Formal Techniques in Real-Time and Fault Tolerant Systems*, volume 2469 of *Lect. Notes Comp. Sci.*, pages 395–416. Springer, Berlin.
- Knapp, Alexander and Zhang, Gefei (2006). Model Transformations for Integrating and Validating Web Application Models. In Mayr, Heinrich C. and Breu, Ruth, editors, *Proc. Modellierung 2006 (MOD'06)*, volume P-82 of *Lect. Notes Inform.*, pages 115–128. Gesellschaft für Informatik.
- Koch, Nora (2001). *Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process*. PhD thesis, Ludwig-Maximilians-Universität München.
- Koch, Nora and Kraus, Andreas (2002). The Expressive Power of UML-based Web Engineering. In Schwabe, Daniel, Pastor, Oscar, Rossi, Gustavo, and Olsina, Luis, editors, *Proc. 2<sup>nd</sup> Int. Wsh. Web-Oriented Software Technology (IWWOST'02)*, pages 105–119. CYTED.
- Koch, Nora and Kraus, Andreas (2003). Towards a Common Metamodel for the Development of Web Applications. In Lovelle, Juan Manuel Cueva, Rodríguez, Bernardo Martín González, Aguilar, Luis Joyanes, Gayo, José Emilio Labra, and del Puerto Paule Ruiz, María, editors, *Proc. Int. Conf. Web Engineering (ICWE'03)*, volume 2722 of *Lect. Notes Comp. Sci.*, pages 495–506. Springer, Berlin.
- Koch, Nora, Kraus, Andreas, and Hennicker, Rolf (2001). The Authoring Process of the UML-based Web Engineering Approach. In Schwabe, Daniel, editor, *Proc. 1<sup>st</sup> Int. Wsh. Web-Oriented Software Technology (IWWOST'01)*. <http://www.dsic.upv.es/~west2001/iwwost01/>.
- Koch, Nora, Zhang, Gefei, and Escalona, María José (2006). Model Transformations from Requirements to Web System Design. In Wolber, Dave, Calder, Neil, Brooks, Chris, and Ginige, Athula, editors, *Proc. 6<sup>th</sup> Int. Conf. Web Engineering (ICWE'06)*, pages 281–288. ACM.
- Lowe, David and Gaedke, Martin, editors (2005). *Proc. 5<sup>th</sup> Int. Conf. Web Engineering (ICWE'05)*, volume 3579 of *Lect. Notes Comp. Sci.* Springer, Berlin.
- Meliá, Santiago, Kraus, Andreas, and Koch, Nora (2005). MDA Transformations Applied to Web Application Development. In Lowe and Gaedke, 2005, pages 465–471.

- Object Management Group (2005). Unified Modeling Language. [www.uml.org](http://www.uml.org).
- Object Management Group (2005). Unified Modeling Language: Superstructure, version 2.0. Specification, OMG. <http://www.omg.org/cgi-bin/doc?formal/05-07-04>.
- Pressman, Roger (2005). *Software Engineering — A Practitioner's Approach*. McGraw-Hill, Boston–Singapore, 6<sup>th</sup> edition.
- QVT-Merge Group (2004). Revised Submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10). Submission, OMG. <http://www.omg.org/cgi-bin/doc?ad/04-04-01.pdf>.
- QVT-Partners (2003). Revised Submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10). Submission, OMG. <http://qvt.org>.
- Robbins, Jason Elliot (1999). *Cognitive Support Features for Software Development Tools*. PhD thesis, University of California, Irvine.
- Schwabe, Daniel and Rossi, Gustavo (1995). The Object-Oriented Hypermedia Design Model. *Comm. ACM*, 38(8):45–46.
- Schwinger, Wieland and Koch, Nora (2006). Modelling of Web Applications. In Kappel, Gerti, Pröll, Birgit, Reich, Siegfried, and Retschitzegger, Werner, editors, *Web Engineering: Systematic Development of Web Applications*, pages 39–64. John Wiley, Hoboken.
- Vilain, Patricia, Schwabe, Daniel, and de Souza, Clarisse Sieckenius (2000). A Diagrammatic Tool for Representing User Interaction in UML. In Evans, Andy, Kent, Stuart, and Selic, Bran, editors, *Proc. 3<sup>rd</sup> Int. Conf. Unified Modeling Language (UML'00)*, volume 1939 of *Lect. Notes Comp. Sci.*, pages 133–147. Springer, Berlin.
- Wirsing, Martin, Koch, Nora, Rossi, Gustavo, Garrido, Alejandra, Mandel, Luis, Helmerich, Alfred, and Olsina, Luis (1999). Hyper-UML: Specification and Modeling of Multimedia and Hypermedia Applications in Distributed Systems. In *Proc. 2<sup>nd</sup> Wsh. German-Argentinian Bilateral Programme for Scientific and Technological Cooperation*, Königswinter.
- Zhang, Gefei (2005). Towards Aspect-Oriented Class Diagrams. In *Proc. 12<sup>th</sup> Asia Pacific Software Engineering Conf. (APSEC'05)*, pages 763–768. IEEE.