Chapter 7

UML-BASED WEB ENGINEERING
An Approach Based on Standards

Nora Koch,[1,2] Alexander Knapp,[1] Gefei Zhang,[1] Hubert Baumeister[3]

[1]*Institut für Informatik, Ludwig-Maximilians-Universität München, Germany,* `{kochn, knapp, zhangg}@pst.ifi.lmu.de`

[2]*F.A.S.T. GmbH, Germany,* `koch@fast.de`

[3]*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`) came up at the end of the 1990s (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 of OOHDM (Schwabe and Rossi, 1995), RMM (Isakowitz et al., 1995), and WSDM (de Troyer and Leune, 1998). The aim, which is still being pursued, was to use a common language or at least to define meta-model-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 techniques of Booch, OOSE, and OMT, seemed to be a promising approach for system modeling. Since those early integration efforts, 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 also uses XMI as a model exchange format (in the hopes of future tool interoperability enabled by a truly portable XMI), MOF for meta-modeling, 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`), 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 Section 7.2 by means of the online movie database case study. The UWE extensions of the UML meta-model are outlined in Section 7.3. UWE's model-driven process and, in particular, the model transformations integrated into the process are described in Section 7.4. The CASE tool ArgoUWE, which supports the UWE notation and method, is described in Section 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.

7.1.1.1 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 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. The presentation model 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, systems' views, and aspects.
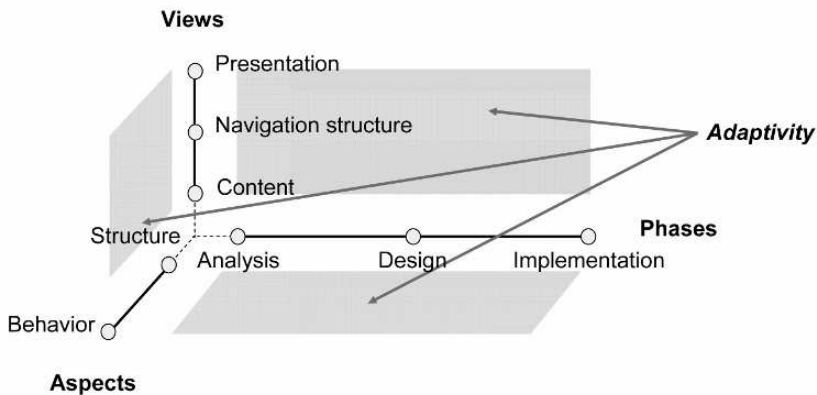

Figure 7.1. Modeling aspects in UWE (from Schwinger and Koch, 2006).

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 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.

7.1.1.2 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 between 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 that lends itself to applying MDD due to the Web-specific separation of concerns and continuous changes in technologies in the Web domain.

Meta-model-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 semiautomated 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 meta-model 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

As the saying goes, 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 semiformal semantics
5. be easy to learn

Although UML fulfills only 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.

7.1.2.1 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 meta-model (Koch and Kraus, 2002, 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 a 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 meta-model, are widely described in the OMG documentation (Object Management Group, 2005). For any software designer with a 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.

7.1.2.2 Meta-Modeling

Meta-modeling plays a fundamental role in CASE tool construction and is as well the core of the model-driven process. A meta-model 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 are becoming essential in the development process of Web systems due to the need for rapid production of new Web presences and Web applications. CASE tools have to be built on a precisely specified meta-model of the modeling constructs used in the design activities, providing more flexibility if modeling requirements change. Meta-models are essential for the definition of model transformations and automatic code generation.

The UWE meta-model is defined as a conservative extension of the UML meta-model (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 meta-model 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 meta-model, we can take advantage of meta-modeling tools based on the corresponding XML interchange format (XMI).

In addition, the UWE meta-model is "profileable" (Baresi et al., 2002), which means that it is possible to map the meta-model 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 meta-model could also be used as the basis for building a common meta-model (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 meta-modeling 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 that 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 predefined UML profiles. In such a case, the profile model elements can be used in the same way as the built-in UML model elements.

7.1.3.1 CASE Tool Support

A wider developer support is achieved by the open source plug-in ArgoUWE (`www.pst.ifi.lmu.de/projekte/uwe`) for the open source CASE tool ArgoUML (`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.

7.1.3.2 Model Consistency Check

ArgoUWE also checks the consistency of models according to the OCL constraints specified for the UWE meta-model. Consistency checking is embedded into the cognitive design critiques 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` and named the "Movie UWE Case Study" (MUC). Movies are characterized, among other things, by their genre, the cast, memorable quotes, trailers, and a soundtrack. Persons related to the movie production include 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 address 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 Figure 7.1). Model transformations are described as part of the model-driven process in Section 7.4.

7.2.1 Starting with Requirements Specification

The first step toward 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.

7.2.1.1 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 among 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 Figure 7.2, for example, includes the «navigation» (□) use cases ViewMovie, Search, and GoToExternalSite. Process 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 database. 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.

7.2.1.2 Detailed View of Use Cases

The level of detail and formality of requirements specifications depends on project risks and the 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 postconditions, a workflow description, exceptions and error situations, information sources, sample results, and references to other documents. In particular, for the development of Web
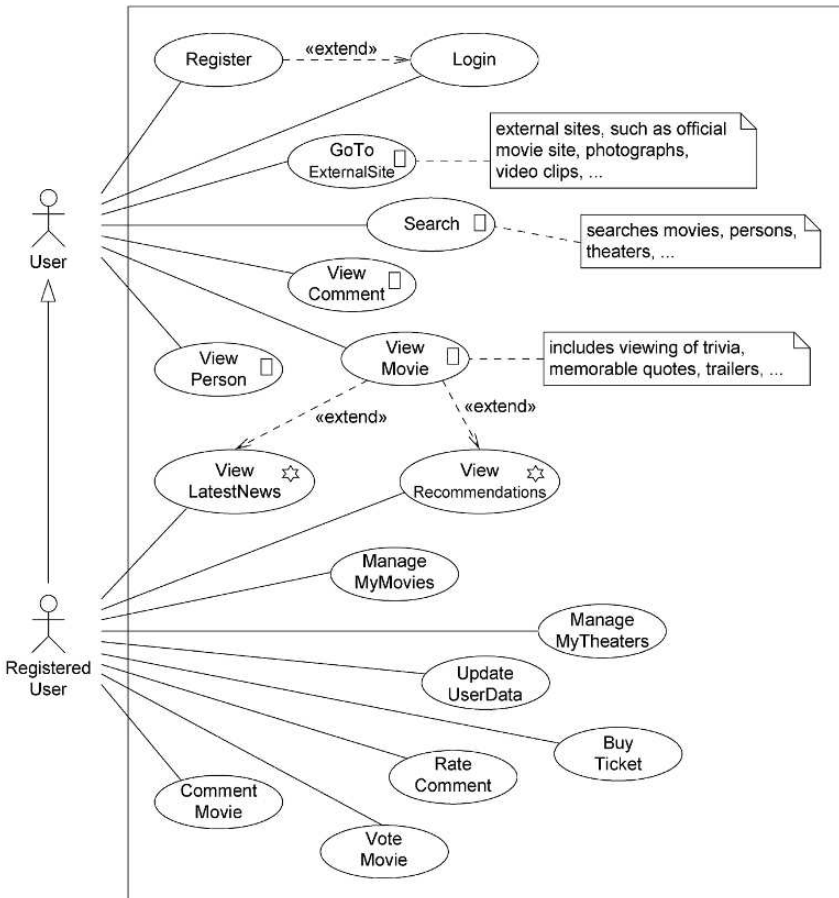
Figure 7.2. UWE use case model for MUC.

systems, the informational, navigational, and process goals have to be gathered and specified. Informational goals indicate content requirements. Navigational goals point toward 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 nontrivial 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 the input or output of those activities.
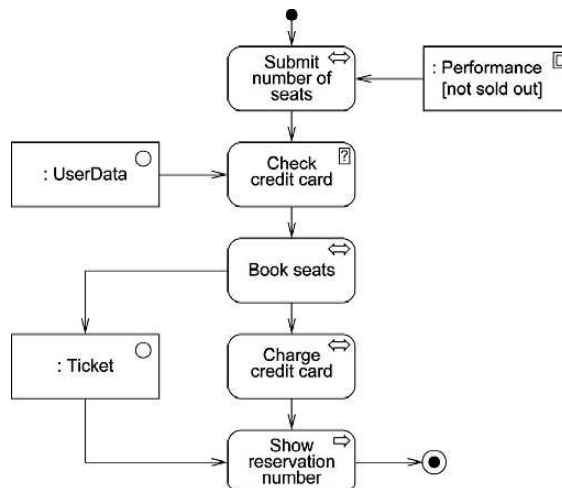

Figure 7.3. MUC case study: UWE activity diagram detailing the buy-ticket use case.

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, 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 also includes entities of the domain required for customized Web applications. These entities constitute the so-called user profile or user model.

Customization deals not only 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.
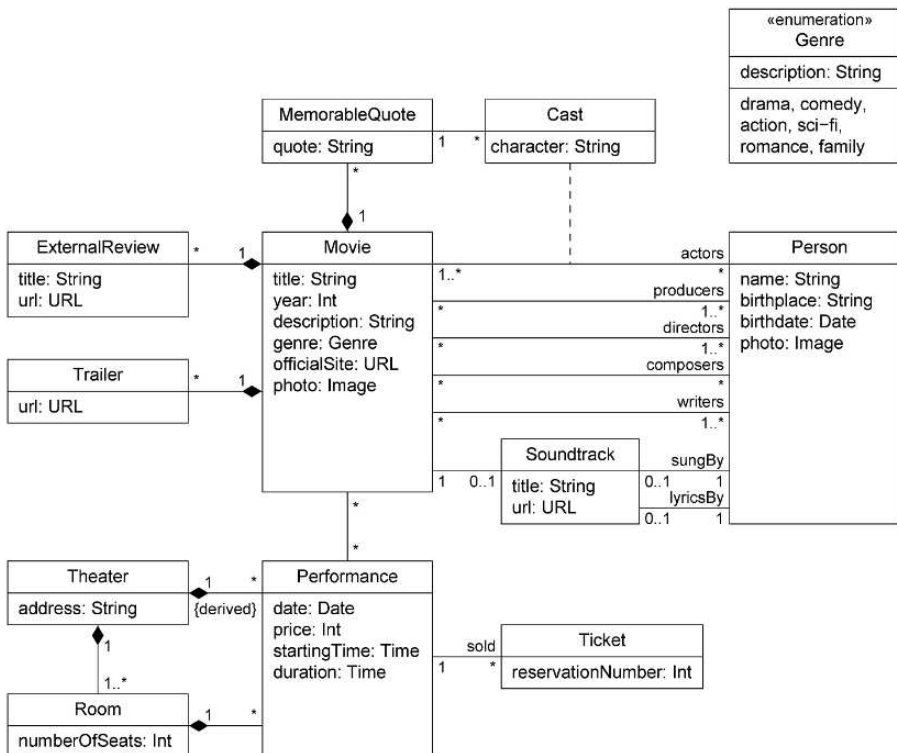

Figure 7.4. MUC case study: content model.

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 Figure 7.4; the user model is shown in Figure 7.5. The entities representing content and

user or context properties respectively, are modeled by classes, i.e., instances of the UML metaclass Class. 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

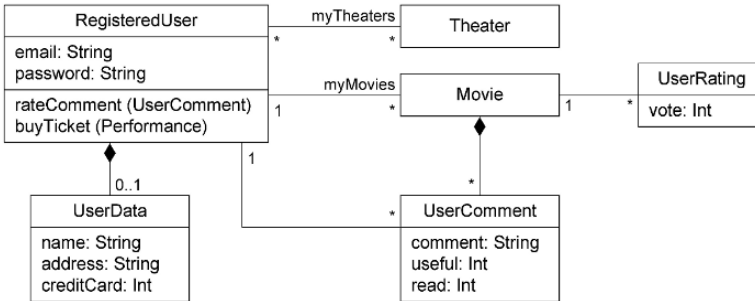*Figure 7.5.* MUC case study: user model.

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 the class Person. Note that Performance and Ticket were inferred from the activity diagram in Figure 7.3.

The user model contains the user data (again see Figure 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 are used for adaptation and are modeled as a cross-cutting aspect and woven into the user model and other parts of the system (see Section 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 constructing class diagrams can be used, in particular, packages, enumerations (e.g., Genre in Figure 7.4), generalizations, compositions, and association classes (e.g., Cast in Figure 7.4).

7.2.3 Laying Down the Navigation Structure

Based on the requirements analysis and the content modeling, the *navigation structure* of a Web application is modeled. Navigation classes (visualized as □) represent navigable nodes of the hypertext structure; navigation links show direct links between navigation classes. Alternative navigation paths

are handled by «menu» (▤). Access primitives are used to reach multiple instances of a navigation class («index» ☰, or «guided tour» ⇝) or to select items («query» ▣). 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 (⟫) 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 that models a business process. Navigation
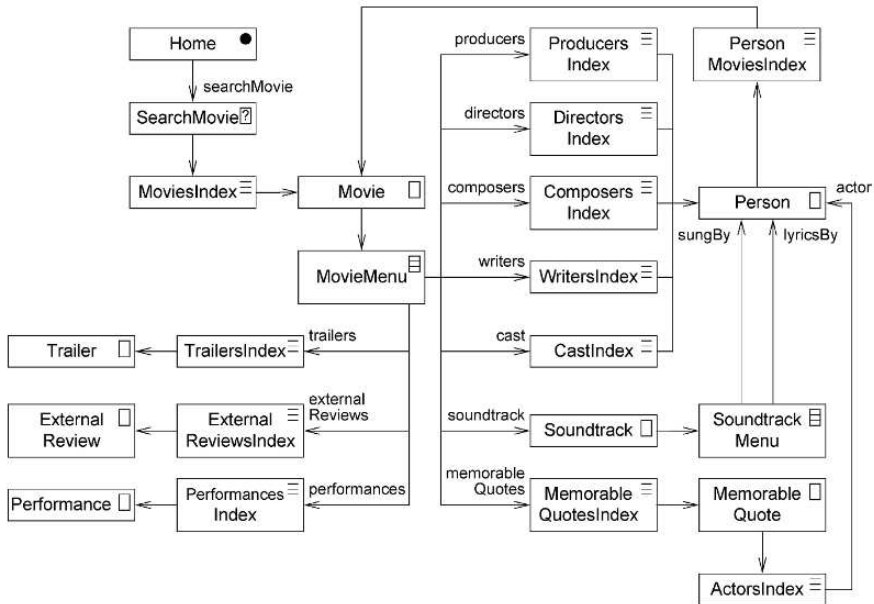

Figure 7.6. MUC case study: navigation from Movie (fragment).

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.

7.2.3.1 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 1. 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 Figure 7.4 is shown in Figure 7.6.

From the home page 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 among which to select 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, one can choose from all movies this person has contributed to. 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.
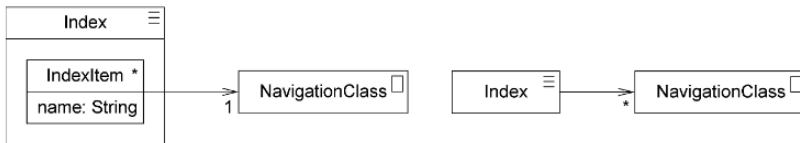

Figure 7.7. "Pure" UML (left) and shorthand notation (right) for index.

The UWE profile notation for menus and access primitives provides a compact representation of patterns frequently used in the Web domain. Figure 7.7 (right) 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 Figure 7.7 (left). The result would be an overloaded model if it contains many such indexes.

7.2.3.2 Adding Business Processes

In a next step, the navigation structure can now be extended by process classes that represent the entry and exit points to business processes. These process classes are derived from the nonnavigational use cases. In Figure 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 her movies if she has logged in previously. Finally, a user can buy tickets for a selected movie and a selected performance by navigating to BuyTicket.

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.
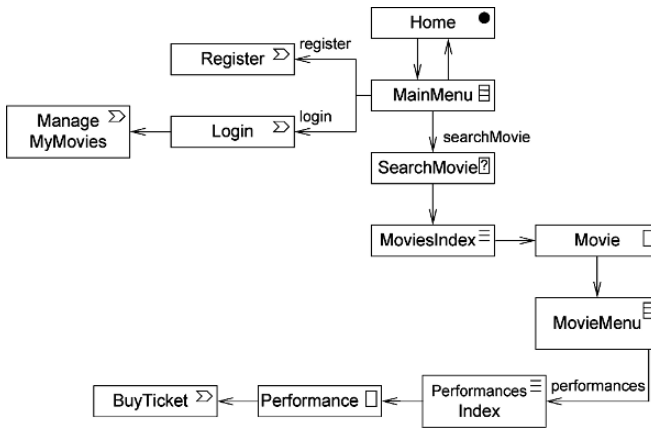

Figure 7.8. MUC case study: integration of business processes into navigation (fragment).

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 a UML activity diagram. It is the result of a refinement process that starts from the workflow in the requirements model.

Figure 7.9 illustrates the result of the refinement process applied to Figure 7.3. This process mainly consists of 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 of 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 the location of UI elements 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.

   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 («text» ≈), anchor («anchor» ▬), button («button» ●), image («image» ▣), form («form» 🗒), and anchored collection («anchored collection» ☰).
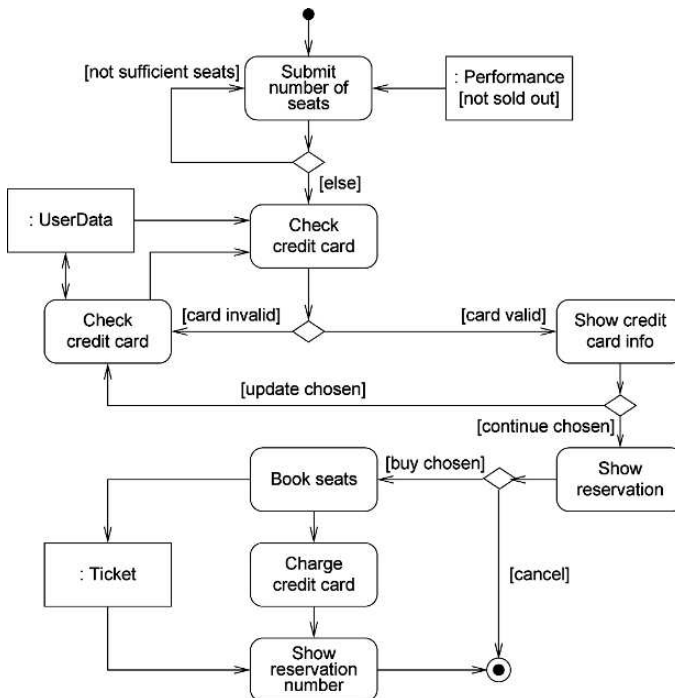

Figure 7.9. MUC case study: UWE process flow model for the buy-ticket process.

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.
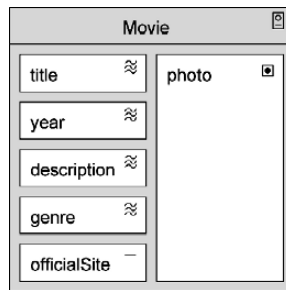

Figure 7.10. MUC case study: presentation class Movie.

Usually, the information from several navigation nodes is presented on one Web page, which is modeled by pages («page») in UWE. Pages can contain, among other things, 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 Figure 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 Figure 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.
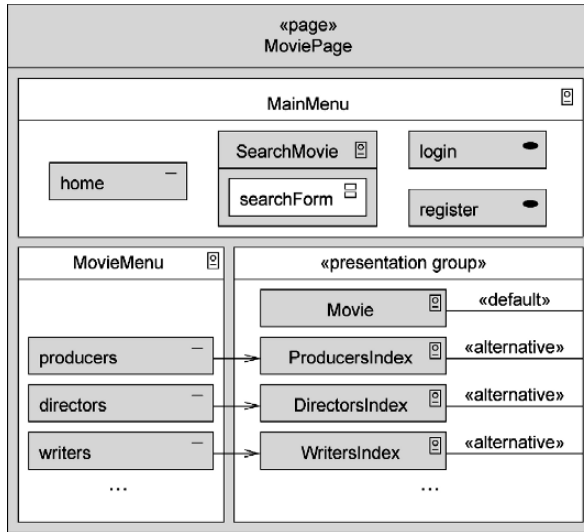

Figure 7.11. MUC case study: the presentation model of the movie page.

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 is recommendations based on user behavior, like movie of favorite actors in our MUC case study. In general, adaptivity is orthogonal to three views: content, navigation structure, and presentation (see Figure 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, in addition to the features specified in the principal model, each model element selected by the pointcut also has 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 run-time adaptivity (Baumeister et al., 2005). In order to model the recommendation feature modularly, we use on the one hand a model aspect and a run-time aspect for keeping track of the number of visits to movie pages. On the other hand, another run-time aspect integrates the recommendation feature into the login process: A list of movies is presented ranked according to the appearing actors, who in turn are ranked according to their relevance in the visited movies.

The static model aspect for extending the user model (see Figure 7.5) by an operation that returns the number of visits of a registered user to a movie page is shown in Figure 7.12 (left). 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 parameter is a class in 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 Figure 7.12, right); 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 are 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 meta-model, which may make AOCD easier to use (cf. Section 7.4).
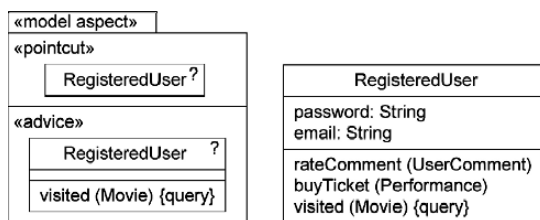

Figure 7.12. MUC case study: model aspect (left) and the weaving result (right).

The dynamic behavior of our MUC system is extended by two run-time 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 1. After weaving, the system's behavior is thus enriched by counting user visits to the movie pages.
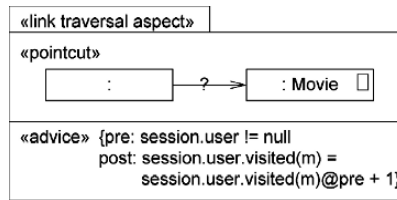

Figure 7.13. MUC case study: link traversal aspect for counting movie visits.

Figure 7.14 shows how the business process Login is extended by a flow aspect. The base model depicted in Figure 7.14 (top) defines the normal workflow without considering adaptivity: The user is asked to input her email address and password, and then the system verifies the input and responds accordingly.
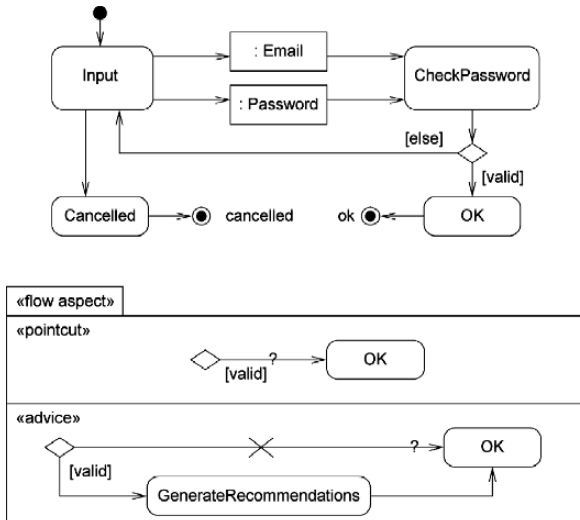

Figure 7.14. MUC case study: flow aspect (bottom) extending business process Login (top).

The adaptive feature of generating recommendations for the user is added by the aspect shown in Figure 7.14 (bottom). 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.

7.3 UWE META-MODEL

The UWE meta-model is defined as a conservative extension of the UML 2.0 meta-model. "Conservative" means that the model elements of the UML meta-model are not modified. Instead, all new model elements of the UWE meta-model are related by inheritance to at least one model element of the UML meta-model. 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 meta-model is profileable, which means that it is possible to map the meta-model to a UML profile (Koch and Kraus, 2003). In particular, UWE stays compatible with the MOF interchange meta-model and therefore with tools that are based on the corresponding XML interchange format XMI. The advantage is that all standard UML CASE tools that 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 Section 7.5) presents an instance of such CASE tool support for UWE based on the UWE meta-model.
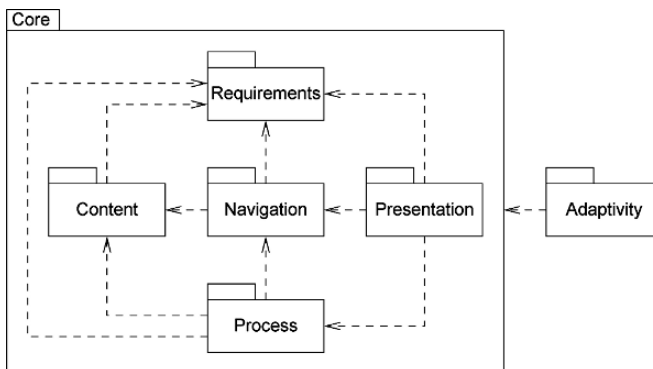

Figure 7.15. Overview of the UWE meta-model.

The UWE extension of the UML meta-model consists of adding two top-level packages, Core and Adaptivity, to the UML (cf. Figure 7.15). The separation of concerns of Web applications is reflected by the package

structure of Core and the cross-cutting of adaptation by the dependency of Adaptivity on Core (see Figure 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 meta-model 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. The 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.

Figure 7.16. UWE navigation meta-model.
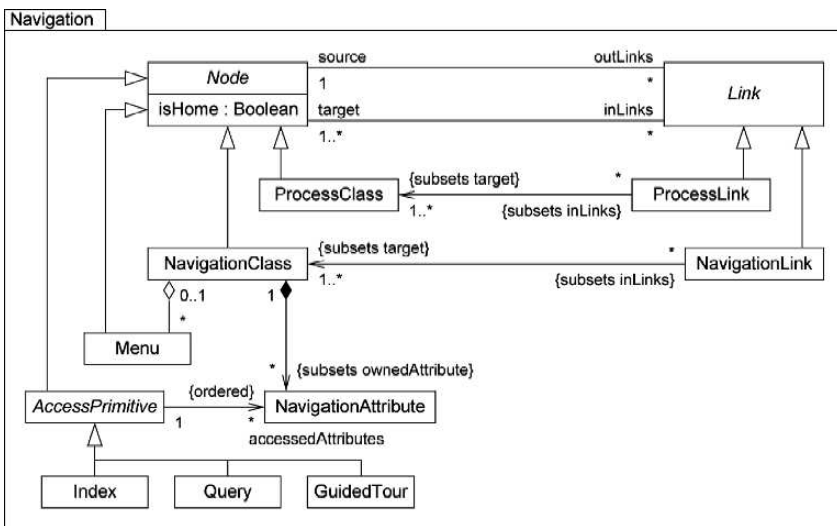
In order to transfer the UWE meta-model into a UML profile, we use UML's extension mechanisms (see Section 7.1). Figure 7.17 shows how the metaclasses of the UWE navigation meta-model are rendered as a stereotype hierarchy, forming the UWE navigation profile: Node becomes a stereotype of Class, NavigationAttribute a stereotype of Property, and Link a stereotype of Association.
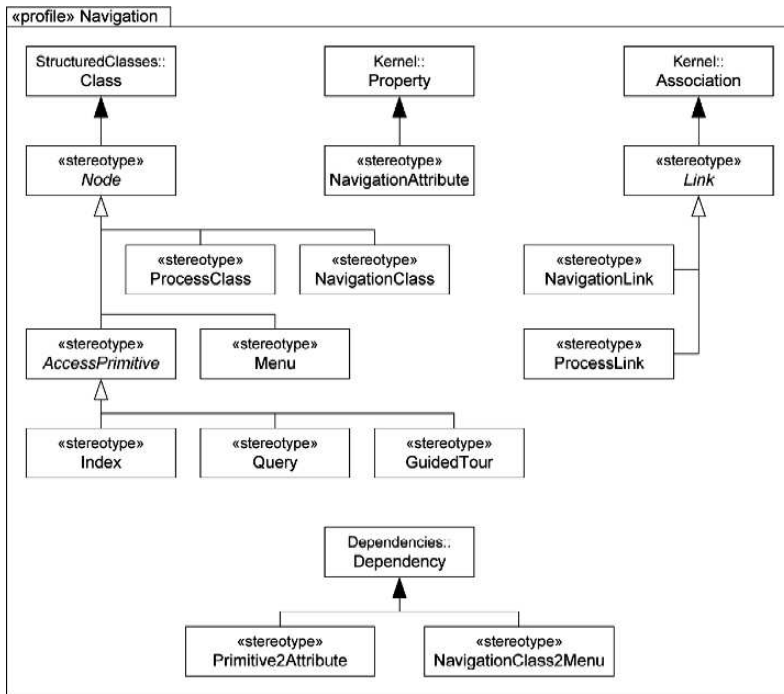
Figure 7.17. UWE navigation profile.

The associations of the UWE navigation meta-model, 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 the UML meta-model 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.

7.3.1 Consistency Rules

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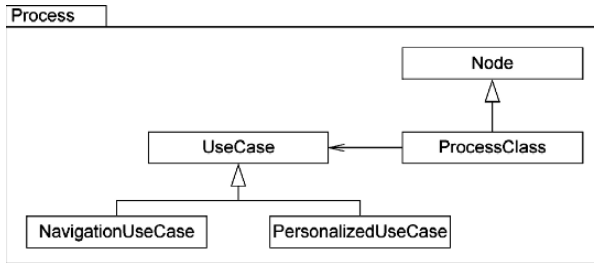

Figure 7.18. UWE process meta-model.

```
context ProcessClass
inv: not self.useCase.oclIsKindOf(NavigationUseCase) and
     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 meta-model. 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 semiautomatic development of Web systems, as detailed in Chapter 12 by Moreno et al. on model-driven Web Engineering. 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 Figure 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).
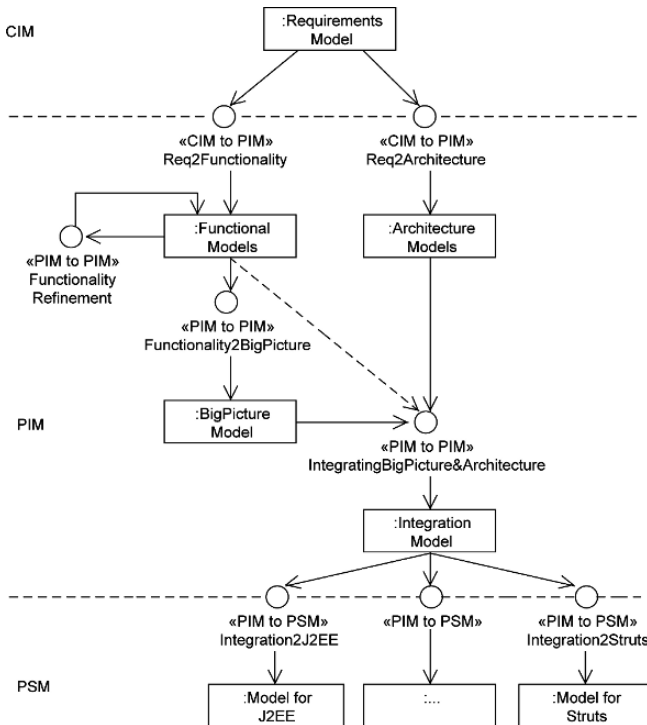

Figure 7.19. Overview of model transformations in the UWE process.

The process starts with the business model, which MDA calls the 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 Figure 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 of 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 providing input for the construction of models in the other development phases (see Figure 7.1, Schwinger and Koch, 2006, and Section 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 functional models. Transformation rules are defined therefore as mappings from the requirements meta-model package to the content, navigation, presentation, process, and adaptivity packages of the meta-model. How these packages depend on each other is shown in Figure 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 a source node and a target node. An action of type search indicates the need for a query in the navigation model in order to allow for user input of a term, and the system responds with a resulting set matching this term (see Section 7.2.1).

Figure 7.20 shows the Search2Query transformation rule specified in QVT's graphical notation (QVT-Merge Group, 2004). The source and target of the transformation are the UWE meta-model defined as *checkonly* and *enforce*, respectively (identified with a "c" and "e" in Figure 7.20). For each search with content 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 meta-model 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 meta-model of the nonfunctional requirements for Web applications and mappings of nonfunctional requirements to architectural model elements are subject to future work.
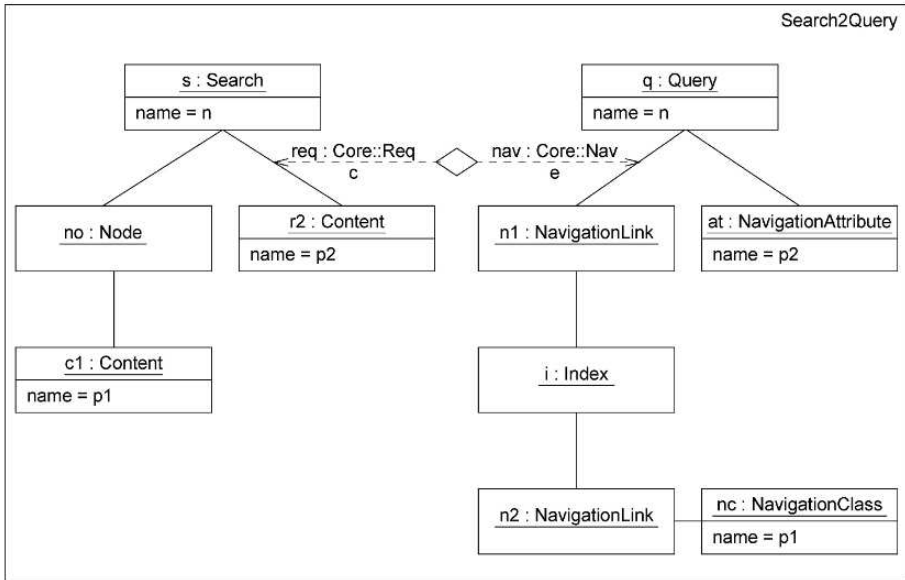
Figure 7.20. Transformation rule Search2Query.

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 Section 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 that 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 make 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 Section 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 nonfunctional 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 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 Figure 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 merging 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 meta-models.
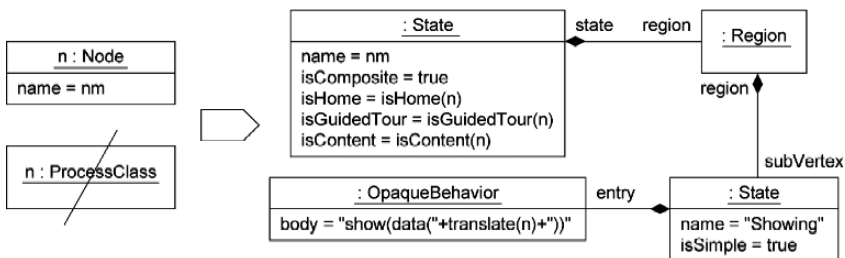

Figure 7.21. Transformation rule Node2State.

7.4.4 Generation of Models and Code for Specific Platforms

In order to transform PIMs into PSMs, additional information about the platform is required. It can be provided as an additional model or it can be 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, F1set.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`). We decided to extend
ArgoUML as it is a feature-rich, open source tool and offers a plug-in
architecture. The drawback of this decision is that the UWE meta-model
cannot 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 meta-
model along the same lines as sketched in Section 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 semiautomatic model transformations defined in the
UWE development process. As these model transformations are based on the
UWE meta-model, 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 meta-model
(Koch and Kraus, 2003), 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 critiques of ArgoUML and can also be checked upon
request (see Section 7.5.2).

Working with ArgoUWE is intuitive for ArgoUML users, as ArgoUWE makes use of ArgoUML's graphical interface. In particular, the UML model
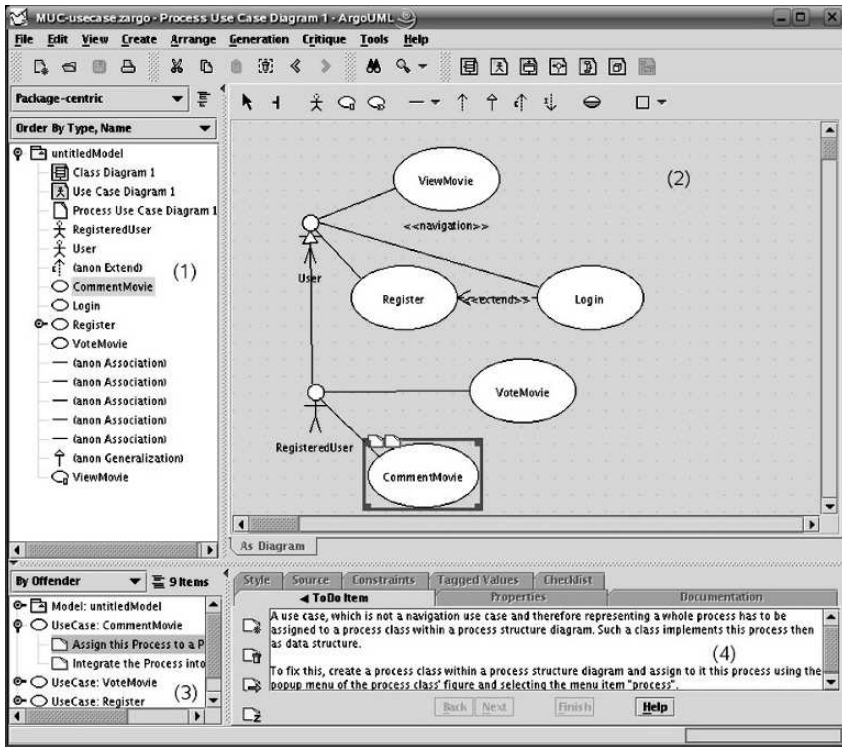

Figure 7.22. MUC case study: ArgoUWE screenshot of a fragment of the use case model.

elements and diagrams are structured in a tree view in the *explorer* [(1) in Figure 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 documentation of the currently selected model as well as automatically generated code skeletons are shown in the *details pane* (4).

7.5.1 Model Transformations

ArgoUWE implements some of the aforementioned model transformations as semiautomatic 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 semiautomatically by using frameworks for the implementation of Web applications, such as Struts (`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 critiques provides an excellent starting point for the implementation of the non-interruptive warnings for UWE models.

The "cognitive design critiques" 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 critiques not only warn the user that her 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 critiques also suggest the use of certain design patterns (Gamma et al., 1995). The issues of design critiques 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 critiques from ArgoUML. In fact, all well-formedness constraints of UWE have been fully integrated and are continuously checked by ArgoUWE in the background at run time. In Figure 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 meta-model constraints shown in Section 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 personalization of Web systems; adaptation to new technologies, e.g. 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 the meta-model level) of (nonfunctional) 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 such as 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 a development language that benefits from the advantages of the different methods. Obviously, there is a trend toward 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 an MOF-based meta-model. 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."

ACKNOWLEDGEMENTS

REFERENCES

Baresi, L., Garzotto, F., Mainetti, L., and Paolini, P., 2002, Meta-modeling techniques meet Web application design tools. In R.-D. Kutsche and H. Weber, eds., *Proceedings Fifth International Conference on Fundamental Approaches to Software Engineering* (FASE'02), pp. 294–307.

Baumeister, H., Knapp, A., Koch, N., and Zhang, G., 2005, Modelling adaptivity with aspects. In D. Lowe and M Gaedke, eds., *Proceedings Fifth International Conference on Web Engineering* (ICWE'05), pp. 406–416.

Baumeister, H., Koch, N., and Mandel, L., 1999, Towards a UML extension for hypermedia design. In R. France and B. Rumpe, eds., *Proceedings Second International Conference on Unified Modeling Language* (UML'99), pp. 614–629. .

de Troyer, O., and Leune, C.J., 1998, WSDM: A user centered design method for Web sites. *Computer Networks*, **30**(1–7): 85–94.

Escalona, M.J., and Koch, N., 2006, Metamodeling the requirements of Web systems. *Proceedings Second International Conference on Web Information Systems and Technologies* (WebIST'06), Setubal, Portugal.

Filman, R.E., Elrad, T., Clarke, S., and Aksit, M., eds., 2004, *Aspect-Oriented Software Development*, Addison-Wesley, Reading, MA.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J., 1995, *Design Patterns*, Addison-Wesley, Reading, MA.

Garrett, J.J., 2005, Ajax: A New Approach to Web Applications. `http://www.adaptivepath.com/publications/essays/archives/000385.php`.

Gómez, J., Cachero, C., and Pastor, O., 2001, Conceptual modeling of device-independent Web applications. *IEEE Multimedia*, **8**(2): 26–39.

Hennicker, R., and Koch, N., 2001, Systematic design of Web applications with UML. In K. Siau and T.A. Halpin, eds., *Unified Modeling Language: Systems Analysis, Design and Development Issues*, Idea Group, Hershey, PA, pp. 1–20.

Isakowitz, T., Stohr, E.A., and Balasubramanian, P., 1995, MM: A methodology for structuring hypermedia design. *Communications of the ACM*, **38**(8): 34–44.

Jouault, F., and Kurtev, I., 2005, Transforming models with ATL. In J.-M. Bruel, ed., *Revised Selection of Papers on Satellite Events at the MoDELS 2005 Conference*, pp. 128–138.

Knapp, A., Koch, N., Moser, F., and Zhang, G., 2003, ArgoUWE: A CASE tool for Web applications. *Proceedings First International Workshop on Engineering Methods to Support Information Systems Evolution* (EMSISE'03), Geneva.

Knapp, A., Merz, S., and Rauh, C., 2002, Model checking timed UML state machines and collaborations. In W. Damm Werner and E.R. Olderog, eds., *Proceedings Seventh International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems,* pp. 395–416.

Knapp, A., and Zhang, G., 2006, Model transformations for integrating and validating Web application models. In H.C. Mayr and R. Breu, eds., *Proceedings Modellierung 2006* (MOD'06), pp. 115–128.

Koch, N., 2001, Software engineering for adaptive hypermedia systems: Reference model, modeling techniques and development process. PhD thesis, Ludwig-Maximilians-Universität, München.

Koch, N., and Kraus, A., 2002, The expressive power of UML-based Web engineering. In D. Schwabe, O. Pastor, G. Rossi, and L. Olsina, eds., *Proceedings Second Internatioanl Workshop on Web-Oriented Software Technology* (IWWOST'02), pp. 105–119.

Koch, N., and Kraus, A., 2003, Towards a common metamodel for the development of Web applications. In J.M.C. Lovelle, B.M.G. Rodríguez, L.J. Aguilar, J.E.L. Gayo, and M. del

Puerto Paule Ruiz, eds., *Proceedings Third International Conference on Web Engineering* (ICWE'03), pp. 495–506.

Koch, N., Kraus, A., and Hennicker, R., 2001, The authoring process of the UML-based Web engineering approach. In D. Schwabe, ed., *Proceedings First International Workshop on Web-Oriented Software Technology* (IWWOST'01). `http://www.dsic.upv.es/ ~west2001/iwwost01/`.

Koch, N., Zhang, G., and Escalona, M.J., 2006, Model transformations from requirements to Web system design. In D. Wolber, N. Calder, C. Brooks, and A. Ginige, eds., *Proceedings Sixth International Conference on Web Engineering* (ICWE'06), pp. 281–288.

Lowe, D., and Gaedke, M., eds., 2005, *Proceedings Fifth International Conference on Web Engineering* (ICWE'05).

Meliá, S., Kraus, A., and Koch, N., 2005, MDA transformations applied to Web application development. In D. Lowe and M. Gaedke, eds., *Proceedings Fifth International Conference on Web Engineering* (ICWE'05), pp. 465–471.

Object Management Group (2005). Unified Modeling Language. 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, R., 2005, *Software Engineering—A Practitioner's Approach,* 6th edition, McGraw-Hill, Boston.

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, version 1.1. `http://qvtp.org/downloads/1.1/qvtpartners1.1.pdf`.

Robbins, J.E., 1999, Cognitive support features for software development tools. PhD thesis, University of California, Irvine.

Schwabe, D., and Rossi, G., 1995, The object-oriented hypermedia design model. *Communications of the ACM*, **38**(8): 45–46.

Schwinger, W., and Koch, N., 2006, Modeling Web applications. In G. Kappel, B. Pröll, S. Reich, and W. Retschitzegger, eds., *Web Engineering: Systematic Development of Web Applications*, John Wiley, Hoboken, NJ, pp. 39–64.

Vilain, P., Schwabe, D., and de Souza, C.S., 2000, A diagrammatic tool for representing user interaction in UML. In A. Evans, S. Kent, and B. Selic, eds., *Proceedings Third International Conference on Unified Modeling Language* (UML'00), pp. 133–147.

Wirsing, M., Koch, N., Rossi, G., Garrido, A., Mandel, L., Helmerich, A., and Olsina, L., 1999, Hyper-UML: Specification and modeling of multimedia and hypermedia applications in distributed systems. In *Proceedings Second Workshop on German-Argentinian Bilateral Programme for Scientific and Technological Cooperation*, Königswinter, Germany.

Zhang, G., 2005, Towards aspect-oriented class diagrams. In *Proceedings 12th Asia Pacific Software Engineering Conference* (APSEC'05), pp. 763–768.

Chapter 9

DESIGNING WEB APPLICATIONS WITH WEBML AND WEBRATIO

Marco Brambilla, Sara Comai, Piero Fraternali, Maristella Matera
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Pizza L. da Vinci 32, 20133, Milan, Italy

9.1 INTRODUCTION

The Web Modeling Language (WebML) is a third-generation Web design methodology, conceived in 1998 in the wake of the early hypermedia models and the pioneering works on hypermedia and Web design, like HDM (Garzotto et al., 1993) and RMM (Isakowitz et al., 1995). The original goal of WebML was to support the design and implementation of so-called data-intensive Web applications (Ceri et al., 2002), defined as Web sites for accessing and maintaining large amounts of structured data, typically stored as records in a database management system, like online trading and e-commerce applications, institutional Web sites of private and public organizations, digital libraries, corporate portals, and community sites.

To achieve this goal, WebML reused existing conceptual data models and proposed an original notation for expressing the navigation and composition features of hypertext interfaces. WebML's hypertext model took an approach quite different from previous proposals: Instead of offering a high number of primitives for representing all the possible ways to organize a hypertext interface that may occur in data-intensive Web applications, the focus was on inventing a minimal number of concepts, which could be composed in well-defined ways to obtain an arbitrary number of application configurations.

This initial design choice deeply influenced the definition of the language and its evolution toward more complex classes of applications. Four major versions of WebML characterize the progression of the language:

- **WebML 1**: The original version comprised only a fixed set of primitives for representing read-only data-intensive Web sites; the focus was on the modular organization of the interface, navigation definition, and content extraction and publication in the interface.
- **WebML 2**: It added support for representing business actions (called operations) triggered by the navigation of the user; in this way, the expressive power was extended to support features like content management, authentication, and authorization.
- **WebML 3**: The introduction of the concept of model plug-ins transformed WebML into an open language, extensible by designers with their own conceptual-level primitives, as to widen the expressive power to cover the requirements of new application domains. This transition emphasized the role of component-based modeling and was the base of all subsequent extensions.
- **WebML 4**: The notion of a model plug-in was exploited to add orthogonal extensions to the core of WebML, covering sectors and applications not previously associated with model-driven development. For example, Web service interaction and workflow modeling primitives were added as plug-in components, to enable the modeling and implementation of distributed applications for multi-actor workflow enactment (Manolescu et al., 2005; Brambilla et al., 2006); other extensions pointed in the direction of multichannel and context-aware Web applications (Ceri et al., 2007).

A distinctive trait of the WebML experience is the presence of an industrial line of development running in parallel to the academic research. One of the original design principles of WebML was implementability, with the ultimate goal of bringing model-driven development (MDD) to the community of "real" developers. To achieve this objective, Politecnico di Milano spun off a company (called Web Models) in 2001, with the mission of implementing and commercializing methods and tools for model-driven development of Web applications, based on WebML. Even before then, WebML had been used for modeling and automatically implementing an industrial project, the Acer-Euro system (http://www.acer-euro.com), comprising the multilingual B2B and B2E content publishing and management applications of Acer, the number 4 PC vendor in the world.

The major result of the industrial R&D is WebRatio (WebModels, 2006), an integrated development environment supporting the modeling of applications with WebML and their implementation with model-driven code generators. Today WebRatio is a consolidated industrial reality: More than 100 applications have been developed by WebModels' customers, over 4,000 trial copies are downloaded per year, and many universities and institutions worldwide use the tool in their Web Engineering courses. In retrospect, the most fruitful and challenging aspect of the interplay of academic and industrial activity has been the continuous relationship between researchers and "real–world," "traditional" developers, which produced essential feedback on the definition of a truly usable and effective model-driven development methodology, which is (hopefully) reflected in the current status of WebML and its accompanying tools.

In this chapter we will overview the core features of WebML and some of its extensions and briefly comment on the usage experience. The chapter is organized as follows: Section 9.2 presents an overview of the WebML methodology and, in particular, introduces the WebML notations for the definition of conceptual schemas. Section 9.3 describes the implementation of the methodology and the architecture of the development tool supporting it. Section 9.4 presents extensions of WebML for supporting Web service composition and publication, workflow-driven Web applications, and context-aware Web applications. Section 9.5 shortly summarizes some of the lessons learned in the application of model-driven development with WebML in industrial projects. Finally, Section 9.6 presents the ongoing and future work and draws the conclusions.

9.2 THE WEBML METHODOLOGY

WebML is a visual language for specifying the content structure of a Web application and the organization and presentation of such content in a hypertext (Ceri et al., 2000, 2002).
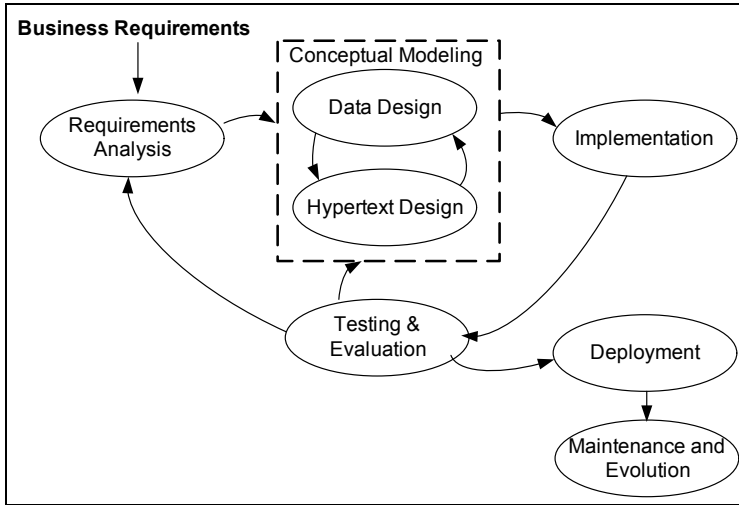
Figure 9.1. Phases in the WebML development process.

As reported in Figure 9.1, the WebML approach to the development of Web applications consists of different phases. Inspired by Boehm's spiral model (Boehm, 1988) and in line with modern methods for Web and software applications development (Beck, 1999; Booch et al., 1999; Conallen, 2000), the WebML process is applied in an iterative and incremental manner in which the various phases are repeated and refined until results meet the application requirements. The product life cycle therefore undergoes several cycles, each producing a prototype or a partial version of the application. At each iteration, the current version of the application is tested and evaluated and then extended or modified to cope with the previously collected requirements as well as the newly emerged requirements. Such an iterative and incremental life cycle appears particularly appropriate for the Web context, where applications must be deployed quickly (in "Internet time") and requirements are likely to change during development.

Out of the entire process illustrated in Figure 9.1, the "upper" phases of analysis and conceptual modeling are those most influenced by the adoption of a conceptual model. The rest of this section will introduce the WebML notations for the definition of conceptual schemas. It will then illustrate the different activities in the WebML development process, with special emphasis on conceptual modeling activities. Some issues about implementation through automatic code generation will be discussed in Section 9.3, by showing how conceptual schemas defined during the design phases can be translated into a running application using WebRatio.

9.2.1 Requirements Analysis

Requirements analysis focuses on collecting information about the application domain and the expected functions and on specifying them through easy-to-understand descriptions. The input to this activity is the set of business requirements that motivate the application development. The main results of this phase are

- the identification of the **groups of users** addressed by the application. Each group represents users having the same characteristics or playing the same role within a business process, i.e., performing the same activities with the same access rights over the same objects. The same individual user may play different roles, thus belonging to different groups.
- the specification of **functional requirements** that address the functions to be provided to users. For each group of users, the relevant activities to be performed are identified and specified.
- the identification of **core information objects**, i.e., the main information assets to be accessed, exchanged, and/or manipulated by users.
- the decomposition of the Web application into **site views**, i.e., different hypertexts designed to meet a well-defined set of functional and user requirements. Each user group will be provided with at least one site view supporting the functions identified for the group.

Analysts are expected to use their favorite format for requirements specification; for instance, tabular formats can be used for capturing the informal requirements such as group or site view descriptions; UML use case diagrams and activity diagrams can also be used as standard representations of usage scenarios and activity synchronization. In particular, functional requirements might be captured by activity flow, showing sequence, and parallelism and synchronization among the activities to be performed by different user groups.

9.2.2 Conceptual Modeling

Conceptual modeling consists of defining conceptual schemas, which express the organization of the application at a high level of abstraction, independently from implementation details. According to the WebML approach, conceptual modeling consists of data design and hypertext design.

Data design corresponds to organizing core information objects previously identified during requirements analysis into a comprehensive and coherent data schema, possibly enriched through derived objects.

Hypertext design then produces site view schemas on top of the data schema previously defined. Site views express the composition of the content and services within hypertext pages, as well as the navigation and the interconnection of components. For applications where different user groups perform multiple activities, or for multichannel applications, in which users can adopt different access devices, hypertext design requires the definition of multiple site views, addressing the user groups involved and their access requirements.

The models provided by the WebML language for data and hypertext design are briefly described in the following. A broader illustration of the language and its formal definition can be found in Ceri et al. (2000, 2002) and at http://www.webml.org.

9.2.2.1 WebML Data Model

Data design is one of the most traditional and consolidated disciplines of information technology, for which well-established modeling languages and guidelines exist. For this reason, WebML does not propose yet another data modeling language; rather, it exploits the entity-relationship data model, or the equivalent subset of UML class diagram primitives. The fundamental elements of the WebML data model are therefore entities, defined as containers of data elements, and relationships, defined as semantic connections between entities. Entities have named properties, called *attributes*, with an associated type. Entities can be organized in generalization hierarchies and relationships can be restricted by means of cardinality constraints.

In the design of Web applications it is often required to calculate the value of some attributes or relationships of an entity from the value of some other elements of the schema. Attributes and relationships so obtained are called *derived*. Derived attributes and relationships can be denoted by adding a slash character (/) in front of their name, and their computation rule can be specified as a logical expression added to the declaration of the attribute or relationship, as is customary in UML class diagrams (Booch et al., 1999). Derivation expressions can be written using declarative languages like OQL or OCL.
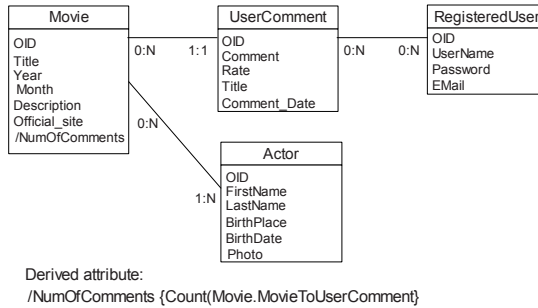
Derived attribute:
/NumOfComments {Count(Movie.MovieToUserComment}

Figure 9.2. A fragment of data schema of the Movie database Web application.

Figure 9.2 shows a small fragment of the data schema of the Movie database example, containing the entities `Movie`, `UserComment`, `RegisteredUser`, `Actor`, and their relationships. The entity `Movie` contains one derived attribute `/NumOfComments`, which is computed as the value of the expression `Count(Movie.MovieToUserComment)`. This expression counts the number of comments associated with a movie according to the `MovieToUserComment` relationship role between the entities `Movie` and `UserComment`.

9.2.2.2 WebML Hypertext Model

The hypertext model enables the definition of the front-end interface, which is shown to a user in the browser. It enables the definition of pages and their internal organization in terms of components (called *content units*) for displaying content. It also supports the definition of links between pages and content units that support information location and browsing. Components can also specify operations, such as content management or user's login/logout procedures. These are called *operation units*.

The modular structure of an application front end is defined in terms of site views, areas, pages, and content units. A *site view* is a particular hypertext, designed to address a specific set of requirements. It consists of *areas*, which are the main sections of the hypertext, and comprises recursively other subareas or pages. *Pages* are the actual containers of information delivered to the user.

Several site views can be defined on top of the same data schema, for serving the needs of different user communities or for arranging content as requested by different access devices like PDAs, smart phones, and similar appliances.
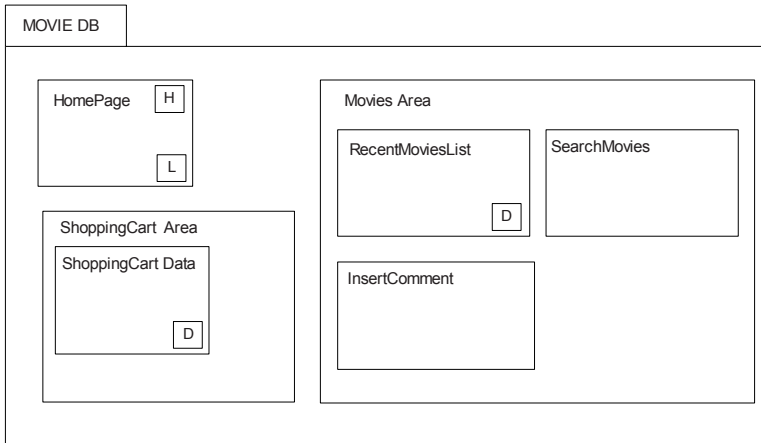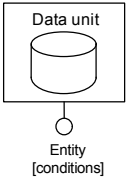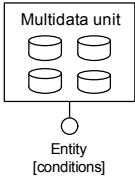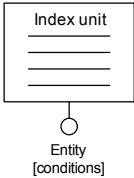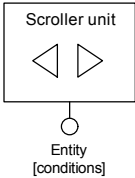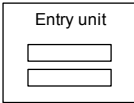
Figure 9.3. Example of site view modularization based on areas and pages.

Figure 9.3 gives an example of the organization of pages and areas in a site view, considering a fragment of the Movie database Web application. The site view is composed of a *home page*, which is the first page accessed when the user enters the application. The site view also comprises two areas: the `Shopping Cart` area, including only one page through which the user manages his current shopping cart; and the `Movies` area, including three pages that show the list of recent movies, support the search of movies, and allow the user to enter comments.

Pages and areas are characterized by some relevance properties, which highlight their "importance" in the Web site. In particular, pages inside an area or site view can be of three types:

- The **home page** (denoted with a small "h" inside the page icon) is the page at the default address of the site view, or the one presented after the user logs into the application; it must be unique.
- The **default page** (denoted with a small "d" inside the page icon) is the one presented by default when its enclosing area is accessed; it must be unique within an area. In the example in Figure 9.3, the `Shopping Cart Data` page and the `Recent Movies List` page are default pages for their enclosing areas. This implies that the two pages are entry points for the two areas.
- A **landmark page** (denoted with a small "l" inside the page icon) is reachable from all the other pages or areas within its enclosing module. For example, in Figure 9.3 the home page is also a landmark page, meaning that a link to it will be available from any other page of the site view.

Table 9.1. The Five Predefined Content Units in WebML

Data Unit	Multidata Unit	Index Unit	Scroller Unit	Entry Unit

Data unit	Multidata unit	Index unit	Scroller unit	Entry unit
Entity [conditions]	Entity [conditions]	Entity [conditions]	Entity [conditions]	

Page composition. Pages are made of *content units*, which are the elementary pieces of information, possibly extracted from data sources, published within pages. Table 9.1 reports the five WebML predefined content units, representing the elementary information elements that may appear in the hypertext pages.

Units represent one or more instances of entities of the structural schema, typically selected by means of queries over the entity attributes or over relationships. In particular, *data units* represent some of the attributes of a given entity instance; *multidata units* represent some of the attributes of a set of entity instances; *index units* present a list of descriptive keys of a set of entity instances and enable the selection of one of them; *scroller units* enable the browsing of an ordered set of objects. Finally, *entry units* do not draw content from the elements of the data schema, but publish a form for collecting input values from the user.

Data, multidata, index, and scroller units include a *source* and a *selector*. The source is the name of the entity from which the unit's content is retrieved. The selector is a predicate, used for determining the actual objects of the source entity that contribute to the unit's content. The previous collection of units is sufficient to logically represent arbitrary content on a Web interface (Ceri et al., 2002). However, some extensions are also available, for example, the *multichoice* and the *hierarchical* indexes reported in Table 9.2. These are two variants of the index unit that allow one to choose multiple objects and organize a list of index entries defined over multiple entities hierarchically.

Link definition. Units and pages are interconnected by links, thus forming a hypertext. Links between units are called *contextual*, because they carry some information from the *source unit* to the *destination unit*. In contrast, links between pages are called *noncontextual*.

Table 9.2. Two Index Unit Variants

In contextual links, the binding between the source unit and the destination unit of the link is formally represented by link parameters, associated with the link, and by parametric selectors, defined in the destination unit. A *link parameter* is a value associated with a link between units, which is transported as an effect of the link navigation, from the source unit to the destination unit. A *parametric selector* is, instead, a unit selector whose condition contains one or more parameters.
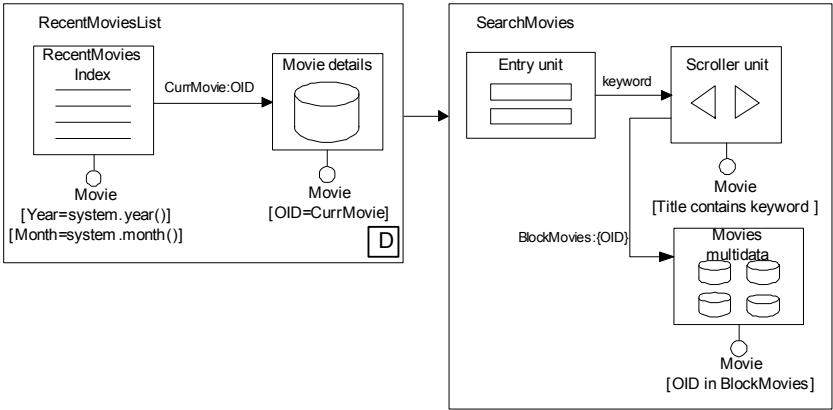

Figure 9.4. Example of contextual and noncontextual navigation.

As an example of page composition and unit linking, Figure 9.4 reports a simple hypertext, containing two pages of the **Movies** Area. The page **Recent Movies List** contains an index unit defined over the **Movie** entity, which shows the list of movies shown in the last month, and a data unit also

defined over the `Movie` entity, which displays the details of the movie selected from the index. Two selectors (`[Year=system.year()]`, `[Month=system.month()]`) are defined to restrict the selection only to the movies of the current month and year. The arrow between the two units is a contextual link, carrying the parameter `CurrMovie`, containing the object identifier (OID) of the selected item. The data unit includes a parametric selector (`[OID=CurrMovie]`), which uses the input OID parameter to retrieve the data of the specific movie.

OIDs of the objects displayed or chosen from the source unit are considered the default context associated with the link. Therefore, OID parameters over links and parametric selectors testing for OID values can be omitted and simply inferred from the diagram.

An example of a noncontextual link is shown from the `Recent Movies List` page to the `Search Movies` page: This link does not carry any parameter, because the content of the destination page does not depend on the content of the source page.

The page `Search Movies` shows an interesting hypertext pattern; it contains three units: an entry unit denoting a form for inserting the keyword of the title to be searched, a scroller unit defined over the `Movie` entity and having a selector for retrieving only the movies containing that keyword in their titles (`[Title contains keyword]`), and a multidata unit displaying a scrollable block of search results. Through the scroller unit it is possible to move to the first, previous, next, and last blocks of results.

Automatic and transport links. In some applications, it may be necessary to differentiate a specific link behavior, whereby the content of some units is displayed as soon as the page is accessed, even if the user has not navigated its incoming link. This effect can be achieved by using automatic links. An *automatic link*, graphically represented by putting a label "A" over the link, is "navigated" in the absence of a user's interaction when the page that contains the source unit of the link is accessed.

Also, there are cases in which a link is used only for passing contextual information from one unit to another and thus is not rendered as an anchor. This type of link is called a *transport link*, to highlight that the link enables only parameter passing and not interaction. Transport links are graphically represented as dashed arrows.
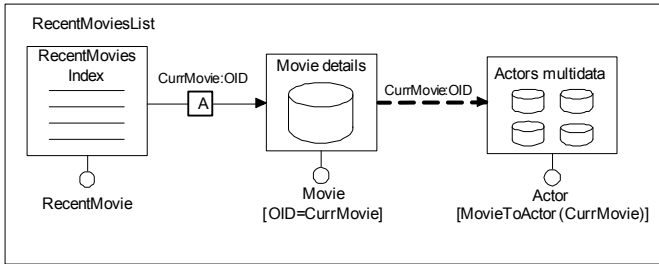
Figure 9.5. Example of automatic and transport links.

Consider the example in Figure 9.5, extending the content of the page `Recent Movies List` shown in Figure 9.4. The link between the index and the data unit has been defined as *automatic*: When the page is accessed, the details of the first movie appearing in the index will be shown to the user, without the need for her interaction. A multidata unit has been added to show the names of the actors playing in the selected movie. A *transport* link is used to pass the OID of the current movie to the multidata unit. This OID is used by the multidata unit in a parametric selector associated with the `MovieToActor` relationship defined between the entities `Movie` and `Actor` to retrieve only the actors associated with the current movie. Note that the automatic link admits the user's interaction for selecting a different movie and is thus rendered as an anchor; conversely, the output link of the data unit does not enable any selection and thus is defined as transport and is not rendered as an anchor.

Global parameters. In some cases, contextual information is not transferred point to point during navigation but can be set as globally available to all the pages of the site view. This is possible through *global parameters*, which abstract the implementation-level notion of session-persistent data.

Parameters can be set through the *Set unit* and consumed within a page through a *Get unit*. The visual representation of such two units is reported in Table 9.3. An example of use of the get unit will be shown in the next subsection.

Operations. In addition to the specification of read-only Web sites, where user interaction is limited to information browsing, WebML also supports the specification of services and content management operations requiring write access over the information hosted in a site (e.g., the filling of a shopping trolley or an update of the users' personal information). WebML offers additional primitives for expressing built-in update operations, such as creating, deleting, or modifying an instance of an entity (represented through the *create*, *delete,* and *modify* units, respectively) or adding or dropping a

relationship between two instances (represented through the *connect* and *disconnect* unit, respectively). The visual representation of such units is reported in Table 9.4.
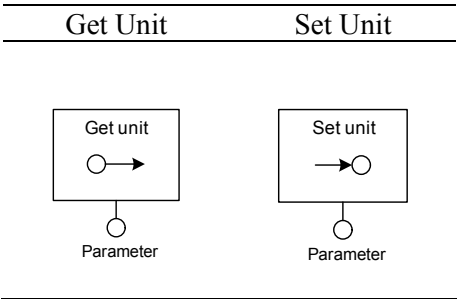
Table 9.3. The WebML Global Parameter Units

Get Unit	Set Unit
Get unit ○—▶	Set unit —▶○
Parameter	Parameter

Table 9.4. The WebML Operation Units

Create Unit	Modify Unit	Delete Unit	Connect Unit	Disconnect Unit
Create ⊕	Modify ±	Delete ⊖	Connect ⊕	Disconnect ⊖
Entity <param := value>	Entity [Conditions] <param := value>	Entity [conditions]	Relationship	Relationship
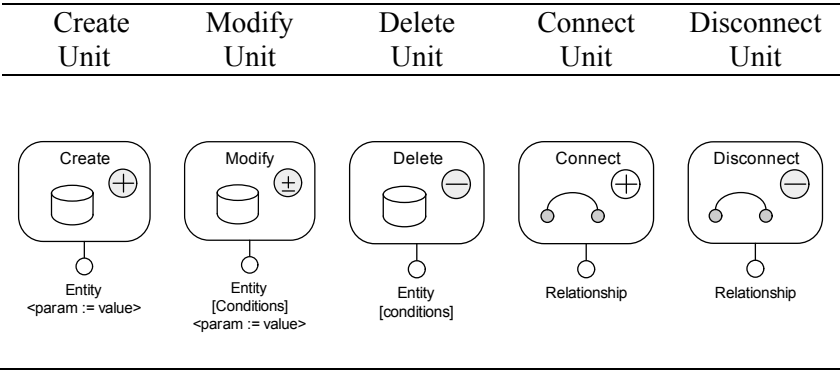
Other utility operations extend the previous set. For example, *login* and *logout* units (see Table 9.5) are respectively used (1) for managing access control and verifying the identity of a user accessing the application site views and (2) for closing the session of a logged user.

Operation units do not publish the content to be displayed to the user but execute some processing as a side effect of the navigation of a link. Like content units, operations may have a source object (either an entity or a relationship) and selectors, may receive parameters from their input links, and may provide values to be used as parameters of their output links. The result of executing an operation can be displayed in a page by using an appropriate content unit, for example, a data or multidata unit, defined over the objects updated by the operation.

Table 9.5. Login and Logout Operations, Supporting Site View Access Control

Login Unit	Logout Unit
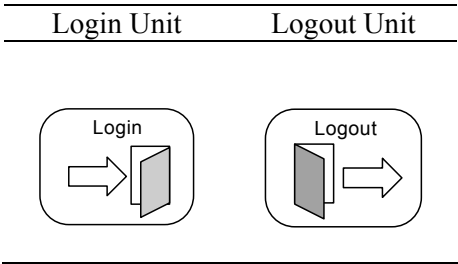
Regardless of their type, WebML operations may have multiple incoming contextual links, which provide the parameters necessary for executing the operation. One of the incoming links is the activating link (the one followed by the user for triggering the operation), while the others just transport contextual information and parameters, for example, the identifiers of some objects involved in the operation.

Two or more operations can be linked to form a chain, which is activated by firing the first operation. Each operation can have two types of output links: one *OK link* and one *KO link*. The former is followed when the operation succeeds; the latter when the operation fails. The selection of the link to follow (OK or KO) is based on the outcome of the operation execution and is under the responsibility of the operation implementation.
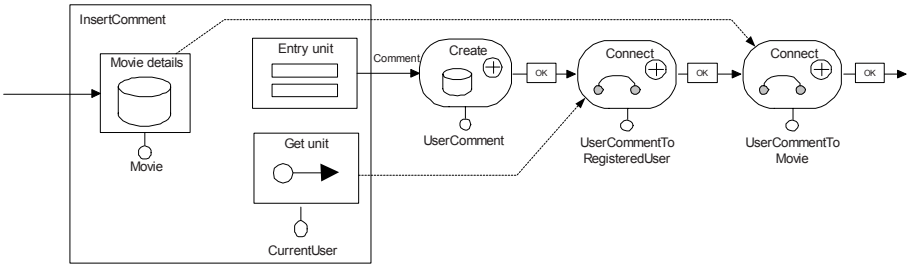
Figure 9.6. Example of content management.

The example in Figure 9.6 shows the content of the `Insert Comment` page in the `Movies` area. Through the entry unit the user can insert a comment for the movie currently displayed by the `Movie details` data unit. A get unit is defined to retrieve the data of the currently logged user, which have been stored in a global parameter after the login. When the user submits a comment, a chain of operations is triggered and executed: First, a new comment instance is created in the `UserComment` entity, containing the text inserted by the user; then, the new comment is associated to the current user (by creating a new

instance of the relationship `UserCommentToRegisteredUser`) and to the current movie (relationship `UserCommentToMovie`). In the example, KO links are not explicitly drawn: By default, they lead the user to the page from which the operation chain has been triggered.

9.2.3 Other Development Phases

The phases following conceptual modeling consist of implementing the application, testing and evaluating it in order to improve its internal and external quality, deploying it on top of a selected architecture, and maintaining and possibly evolving the application once it has been deployed.

As described in more details in Section 9.3, the WebRatio development environment (WebModels, 2006) largely assists the implementation phase. First of all, it offers a visual environment for drawing the data and hypertext conceptual schemas. Such visual specifications are then stored as XML documents, which are the inputs for the WebML code generator, which then produces the data and hypertext implementation.

For space reasons, the remaining phases of the application life cycle are only hinted at in this chapter, but they are nonetheless well supported by WebML and WebRatio. In particular:

- The model-driven approach benefits the systematic testing of applications, thanks to the availability of the conceptual model and the model transformation approach to code generation (Baresi et al., 2005). With respect to the traditional testing of applications, the focus shifts from verifying individual Web applications to assessing the correctness of the code generator. The intuition is that if one could ensure that the code generator produces a correct implementation for all legal and meaningful conceptual schemas (i.e., combinations of modeling constructs), then testing Web applications would reduce to the more treatable problem of validating the conceptual schema. The research work conducted in this area has shown that it is possible to quantitatively evaluate the confidence in the correctness of a model-driven code generator, by formally measuring the coverage of a given test set (that is, of a set of sample conceptual schemas) with respect to the entire universe of syntactically admissible schemas. Different notions of coverage have been proposed, and heuristic rules have been derived for minimizing the number of test cases necessary to reach the desired coverage level of the testing process.
- Model-driven development also fosters innovative techniques for quality assessment. The research in this area has led to a framework for the model-driven and automatic evaluation of Web application quality (Fraternali et al., 2004; Lanzi et al., 2004; Meo and Matera, 2006). The

framework supports the *static* (i.e., compile-time) analysis of conceptual schemas and the *dynamic* (i.e., run-time) collection of Web usage data to be automatically analyzed and compared with the navigation dictated by the conceptual schema. The static analysis is based on the discovery in the conceptual schema of design patterns and on their automatic evaluation against quality attributes encoded as rules. Conversely, usage analysis consists of the automatic examination and mining of enriched Web logs, called *conceptual logs* (Fraternali et al., 2003), which correlate common HTTP logs with additional data about (1) the units and link paths accessed by the users, and (2) the database objects published within the viewed pages.

- In a model-driven process, maintenance and evolution also benefit from the existence of a conceptual model of the application. Requests for changes can in fact be turned into changes at the conceptual level, either to the data model or to the hypertext model. Then, changes at the conceptual level are propagated to the implementation. This approach smoothly incorporates change management into the mainstream production life cycle and greatly reduces the risk of breaking the software engineering process due to the application of changes solely at the implementation level.

9.3 IMPLEMENTATION

Application development with WebML is assisted by WebRatio (WebModels, 2006), a commercial tool for designing and implementing Web applications. The architecture of WebRatio (shown in Figure 9.7) consists of two layers: a *design layer*, providing functions for the visual editing of specifications, and a *run-time layer*, implementing the basic services for executing WebML units on top of a standard Web application framework.

The design layer includes a graphical user interface (shown in Figure 9.8) for data and hypertext design, which produces an internal representation in XML of the WebML models. A data mapping module, called Database Synchronizer, maps the entities and relationships of the conceptual data schema to one or more physical data sources, which can be either created by the tool or pre-existing. The Database Synchronizer can forward- and reverse-engineer the logical schema of an existing data source, propagate the changes from the conceptual data model to the physical data sources, and vice versa.
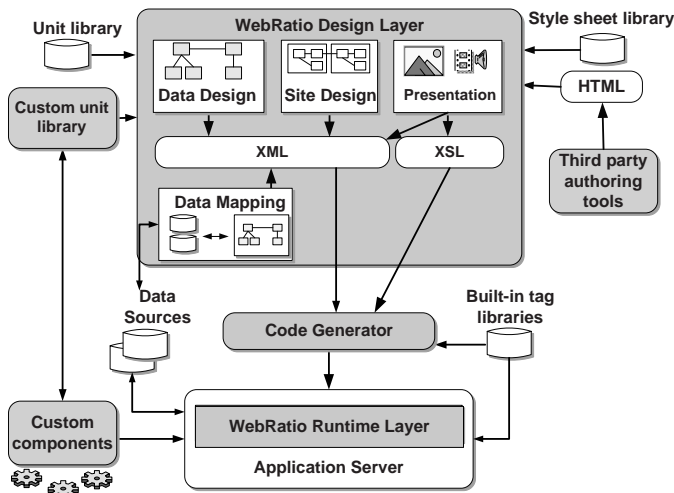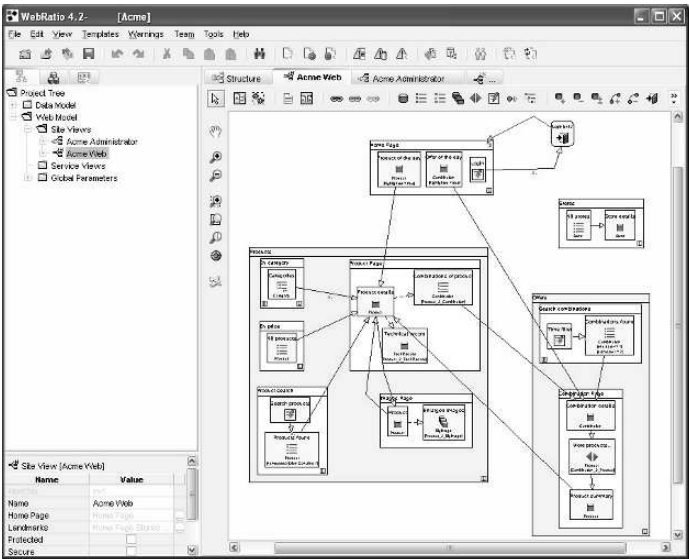
Figure 9.7. The WebRatio architecture.

Figure 9.8. WebRatio's graphical user interface.

A third module (called *EasyStyler Presentation Designer*) offers functionality for defining the presentation style of the application, allowing the designer to create XSL stylesheets from XHTML mock-ups, associate XSL styles with WebML pages, and organize page layout, by arranging the relative position of content units in each page.

The design layer is connected to the run-time layer by the WebRatio code generator, which exploits XSL transformations to translate the XML specifications visually edited in the design layer into application code executable within the run-time layer, built on top of the Java2EE platform. In particular, a set of XSL translators produces a set of *dynamic page templates* and *unit descriptors*, which enable the execution of the application in the run-time layer. A dynamic page template (e.g., a JSP file) expresses the content and markup of a page in the markup language of choice (e.g., in HTML, WML, etc.). A unit descriptor is an XML file that expresses the dependencies of a WebML unit from the data layer (e.g., the name of the database and the code of the SQL query computing the population of an index unit).

The design layer, code generator, and run-time layer have a plug-in architecture: New software components can be wrapped with XML descriptors and made available to the design layer as custom WebML units, the code generator can be extended with additional XSL rules to produce the code needed for wrapping user-defined components, and the components themselves can be deployed in the run-time application framework. As described in the following section, such a plug-in architecture has been exploited to extend WebRatio to support new WebML constructs that have been recently defined for covering advanced modeling requirements.

9.4 ADVANCED FEATURES

The core concepts of WebML have been extended to enable the specification of complex applications, where Web services can be invoked, the navigation of the user is driven by process model specifications, and page content and navigation may be adapted (like in a multichannel, mobile environment). In the next subsections we briefly present the extensions that have been integrated in the WebML model for designing service-enabled, process-enabled, and context-aware Web applications.

9.4.1 Service-Enabled Web Applications

Web services have emerged as essential ingredients of modern Web applications: They are used in a variety of contexts, including Web portals for collecting information from geographically distributed providers or B2B applications for the integration of enterprise business processes.

To describe Web services interactions, WebML has been extended with Web service units (Manolescu et al., 2005), implementing the WSDL (W3C, 2002) classes of Web service operations.

We start by recalling some basic aspects of WSDL, providing the foundation of the proposed WebML extensions. A *WSDL operation* is the basic unit of interaction with a service and is performed by exchanging messages.

Two categories of operations are initiated by the client:

- *One-way* operations consist of a message sent by the client to the service.
- *Request-response* operations consist of one request message sent by the client and one response message built by the service and sent back to the client.

Two other operation categories are initiated by the service:

- *Notification operations* consist of messages sent to the service.
- *Solicit* and *response* operations are devised for receiving request messages sent to the service and providing messages as responses to the client.

WebML supports all four categories of operations. In particular, we interpret the operations initiated by the service as a means for *Web services publishing*. Therefore, we assume that these operations will not be used within the traditional hypertext schemas representing the Web site, but within appropriate *Service views*, which contain the definition of published services. The operations initiated by the client are instead integrated within the specification of the Web application. In the following subsections we will see how they can be specified in WebML and present some examples applied to the Movie database running case.

9.4.1.1 Modeling Web Applications Integrated with Web Services

The specification of Web service invocation from within a Web application exploits the request-response and one-way operations. Here we show an example of a request-response operation. Suppose we want to extend the Movie database Web application with the possibility of retrieving books related to a particular movie from a remote Web service (e.g., the Amazon

Web service). Assume that the request-response operation `SearchBooks` allows one to obtain a list of books meeting search criteria provided as input to the service (e.g., keywords contained in the title). The remote Web service responds with the list of books meeting the given search criteria.

The WSDL request-response operation is modeled through the request-response unit, whose graphical notation is shown in Figure 9.9. This operation involves two messages: the message sent to the service and the message received from the service. The corresponding unit is labeled with the Web service operation name and includes two arrows that represent the two messages. This operation is triggered when the user navigates one of its input links; from the parameters transferred by these links, a message is composed and then sent to a remote service as a request. The user waits until the arrival of the response message from the invoked service; then she can resume navigation from the page reached by the output link of the Web service operation unit.
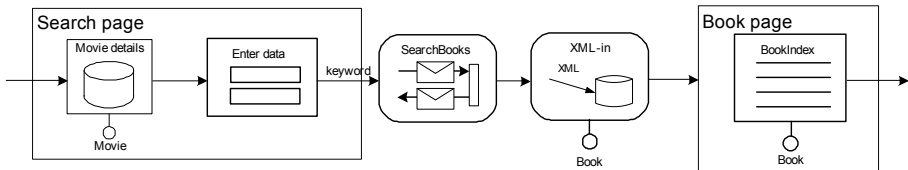

Figure 9.9. Example of usage of the request-response operation.

In the example in Figure 9.9, the user can browse to the `Search page`, where an entry unit permits the input of search criteria, preloaded from the currently selected movie. From this information, a request message is composed and sent to the `SearchBooks` operation of the Web service exposed by the service provider. The user then waits for the response message, containing a list of books satisfying the search criteria. From these options, a set of instances of the `Book` entity is created through the XML-in operation unit (which receives as input XML data and transforms them into relational data) and displayed to the user by means of the `Book Index` unit; the user may continue browsing, e.g., by choosing one of the displayed books. Further details about data transformations and about the storage of data retrieved from Web services can be found in recent publications (Manolescu et al., 2005).

One-way operations are modeled in a similar way: The main difference is that the service will not provide any response. Therefore, once the message is sent to the service, the user continues navigation without waiting for the response.

9.4.1.2 Modeling Web Services Publishing

WebML also supports the publication of Web services that can be invoked by third-party applications. From the application point of view, no user interaction is required in a published Web service. The actions to be performed when the notification or the solicit-response operations are triggered are not specified through pages, but as a chain of operations (e.g., for storing or retrieving data, or for executing generic operations such as sending emails). Therefore, the publishing of Web services can be specified separately from the site view of a Web application. We introduce the following concepts:

- *Service view*: a collection of ports that expose the functionality of a Web service through WSDL operations
- *Port*: the individual service, composed by a set of WSDL operations; each individual WSDL operation is modeled through a chain of WebML operations starting with a solicit-response and/or notification operation

Therefore, the business logic of a WSDL operation is described by a chain of WebML operations, specifying the actions to be performed as a consequence of the invocation of the service, and possibly building the response message to be sent back to the invoker. Each WSDL operation starts with a *solicit unit*, which triggers the service, and possibly ends with the *response unit*, which provides a message back to the service. Here we show an example of a solicit-response operation.

Suppose we want to extend the Movie database application with the publication of a service providing the list of movies satisfying search criteria. The WSDL operation is modeled through a chain of WebML operations starting with the solicit unit (**SearchSolicit**), shown in Figure 9.10. The solicit unit receives the SOAP message from the requester and decodes the search keywords, passing them as parameters to the next WebML operation in the sequence. This is a so-called XML-out (Manolescu et al., 2005) operation unit, which extracts from the database the list of movies that correspond to the specified conditions and formats it as an XML document. After the XML-out operation, the composition of the response message is performed through the *response unit* (**SearchResponse**).
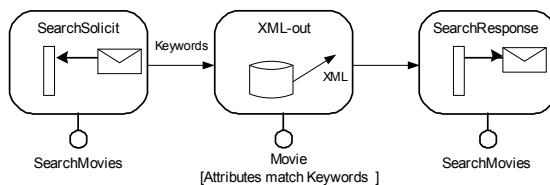

Figure 9.10. Example of usage of the solicit-response operation.

Notice that the schema of Figure 9.10 can be seen as the dual specification of the *SearchBooks* service invocation pattern, represented in Figure 9.9.

In addition to the above-mentioned examples, WebML also supports the exchange of asynchronous messages (Brambilla et al., 2004) and complex Web service conversations (Manolescu et al., 2005).

From the implementation standpoint, the deployment and publishing of Web services required the extension of the run-time WebRatio with a SOAP listener able to accept SOAP requests.

9.4.2 Process-Enabled Web Applications

Today the mission of Web applications is evolving from the support of online content browsing to the management of full-fledged collaborative workflow-based applications, spanning multiple individuals and organizations. WebML has been extended for supporting lightweight Web-enabled workflows (Brambilla, 2003; Brambilla et al., 2003, 2007), thus transferring the benefits of high-level conceptual modeling and automatic code generation also to this class of Web applications.

Integrating hypertexts with workflows means delivering Web interfaces that permit the execution of business activities and embodying constraints that drive the navigation of users. The required extensions to the WebML language are the following:

- *Business process model:* A new design dimension is introduced in the methodology. It consists of a workflow diagram representing the business process to be executed, in terms of its activities, the precedence constraints, and the actors/roles in charge of executing each activity.
- *Data model:* The data model representing the domain information is extended with a set of objects (namely, entities and relationships) describing the meta-data necessary for tracking the execution of the business process, both for logging and for constraints evaluation purposes.
- *Hypertext model:* The hypertext model is extended by specifying the business activity boundaries and the workflow-dependent navigation links.

Besides the main models, the proposed extension affects the following aspects of the WebML methodology:

- *Development process:* Some new phases are introduced in the development process, to allow the specification of business processes and their integration in the conceptual models (see Figure 9.11).

- *Design tools:* A new view shall be introduced for supporting the design of the workflow models within the WebML methodology.
- *Automatic generation tools:* A new transformer is needed for translating workflow diagrams into draft WebML specifications of the Web applications implementing the process specification.
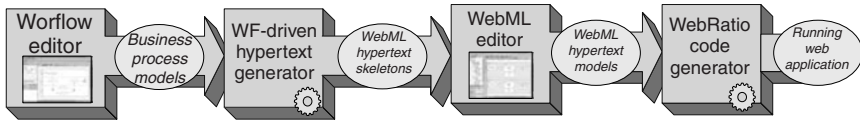

Figure 9.11. Steps of the proposed methodology: Square boxes represent the design steps and the involved tools; bubbles represent the expected results of each step.

The following sections present the details of the process-related extensions, by referring to a specific aspect of the Internet movie database case study, namely the subscription process. Details will be provided about the new features of the development process, the business process modeling, and the data and hypertext modeling.

9.4.2.1 Extensions to the Development Process

The development process is enriched by a set of new design tasks and automatic transformations that addresses the workflow aspects of the application. Figure 9.11 shows the expected steps of the development, the results of each steps, and the involved tools: Through a visual workflow editor, the analyst specifies the business process model to be implemented; the designed workflow model can be processed by an automatic transformation that generates a set of hypertext skeletons implementing the specified behavior; the produced skeletons can be modified by designers by means of CASE tools for conceptual Web application modeling; the resulting models can be processed by automatic code generators that produce the running Web application.

9.4.2.2 Workflow Model and Design Tool

Many standard notations have been proposed to express the structure of business processes. For our purposes, we adopt the Business Process Management Notation (BPMN), which covers the basic concepts required by WfMC (Workflow Management Coalition) and is compatible with Web service choreography languages (e.g., BPEL4WS) and standard business process specification languages (e.g., XPDL). A visual design tool for business processes has been implemented for covering this design phase. The tool is an Eclipse plug-in and allows one to specify BPMN diagrams.

Figure 9.12 shows a subscription process that could apply to the Movie database scenario (the case study has been extended to avoid a simplistic example): The user specifies whether he is a private customer or a company, then he alternatively submits the company or his own personal information, and finally a user manager accepts the subscription.
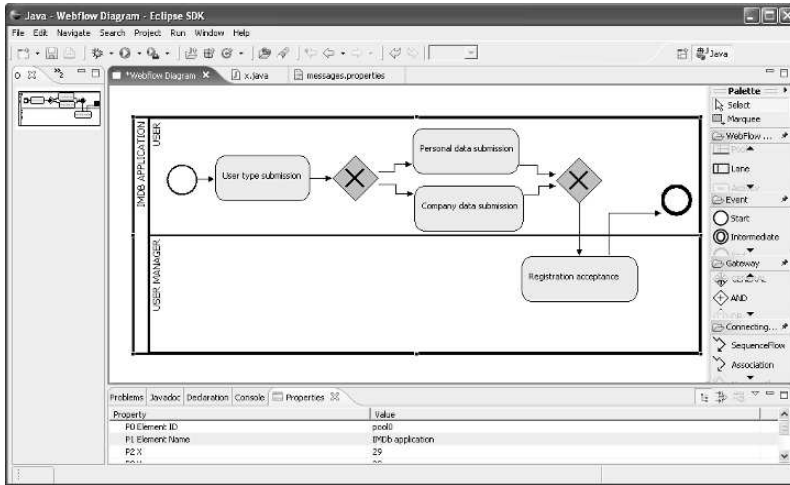

Figure 9.12. Subscription process represented in BPMN in the BP design tool.

9.4.2.3 Data Model Extensions: Workflow Meta-Data

The extensions to the data model include some standard entities for recording activities instances and process cases, thus allowing one to store the state of the business process execution and enacting it accordingly. The adopted meta-model is very simple (see Figure 9.13): The `Case` entity stores the information about each instantiation of the process, while the `Activity` entity stores the status of each activity instance executed in the system. Each activity belongs to a single case. Connections to user and application data can be added, for the purpose of associating domain information to the process execution. Typical requirements are the assignment of application objects to activity instances and the tracking of the relation between an activity and its executor (a user).

Notice that the proposed meta-model is just a guideline. The designer can adopt more sophisticated meta-data schemas or even integrate with underlying workflow engines through appropriate APIs (e.g., Web services) for tracking and advancing the process instance.
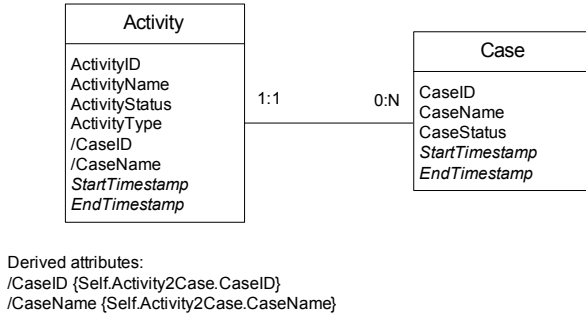
Derived attributes:
/CaseID {Self.Activity2Case.CaseID}
/CaseName {Self.Activity2Case.CaseName}

Figure 9.13. Workflow meta-data added to the data model.

9.4.2.4 Hypertext Model Extensions: Activities and Workflow Links

The hypertext model is extended with two new primitives:

- *Activity:* An activity is represented by an area tagged with a marker "A."
  The whole hypertext contained in the area is the implementation of the
  activity.
- *Workflow link:* Workflow links are links that traverse the boundary of
  any activity area. They are used for hypertext navigation, but their
  behavior includes workflow logic, which is not explicitly visible in the
  hypertext. Every link entering an activity represents the start of the
  execution of the activity; every outgoing link represents the end of the
  activity. The actual behavior of the workflow links is specified by a
  category associated with the link.

Incoming links can be classified as *Start link*, allowing an existing
activity to start from scratch; *Start case link,* allowing one to create a new
case and a new activity and to start them; *Create link,* allowing one to create
a new activity and start it; *Resume link*, allowing one to resume the
execution of an activity once it has been suspended.

Outgoing links can be classified as *Complete link*, which closes the
activity and sets its status to completed; *Complete case link*, which closes the
activity and the whole case, setting their status to completed; *Suspend link,*
which suspends the execution of an activity (that can be resumed later
through a resume link); *Terminate link,* which closes the activity and sets its
status to terminated (e.g., for exception management).

Notice that *if* and *switch* units can be used to express navigation
conditions. Moreover, a specific approach has been studied for managing
exceptions within workflow-based Web applications (Brambilla et al., 2005;
Brambilla and Tziviskou, 2005), but it is not discussed here for the sake of

brevity. Moreover, by combining workflows and Web services extensions, the design of distributed processes can be obtained (Brambilla et al., 2006).

9.4.2.5 Mapping Workflow Schemas to Hypertext Models

Workflow activities are realized in the hypertext model by suitable configurations of pages and units, enclosed within an activity area. Workflow constraints must be turned into navigation constraints among the pages of the activities and into data queries on the workflow meta-data for checking the status of the process, thus ensuring that the data shown by the application and user navigation respect the constraints described by the process specification. The description of how the precedence and synchronization constraints between the activities can be expressed in the hypertext model is specified in Brambilla et al. (2003), which describes the mapping between each workflow pattern and the corresponding hypertext.

A flexible transformation, depending on several tuning and style parameters, has been included in the methodology for transforming workflow models into skeletons of WebML hypertext diagrams.

The produced WebML model consists of an application data model, workflow meta-data, and hypertext diagrams. The transformation supports all the main WfMC precedence constraints, which include sequences of activities, AND-, OR-, XOR- splits and joins, and basic loops.

Since no semantics is implied by the activity descriptions, the generated skeleton can only implement the empty structure of each activity and the hypertext and data queries that are needed for enforcing the workflow constraints. The designer remains in charge of implementing the interface and business logic of each activity. Additionally, it is possible to annotate the activities with a set of predefined labels (e.g., create, update, delete, browse), thus allowing the transformer tool to map the activity to a coarse hypertext that implements the specified behavior.

Once the transformation has been accomplished, the result can be edited with WebRatio (WebModels, 2006), thus allowing the designer to refine the generated hypertext and to implement the internal behaviour of each activity.

9.4.2.6 Workflow-Based Hypertext Example

Figure 9.14 shows the hypertext diagram for the `Personal Data Submission` activity, which is part of the example process depicted in Figure 9.12. Notice that the shown implementation is the final result of the two steps of automatic hypertext skeleton generation and of hypertext refinement by the designer. The link marked with the "…" label may come from any hypertext fragment in the site view.

Before starting the activity, a condition is checked for verifying that the `Company data submission activity` is not started yet, since it is defined as mutually exclusive with respect to the `Personal Data Submission` activity (a corresponding XOR-split decision gateway is shown in Figure 9.14). Hence, the condition to be checked before starting `Personal Data Submission` is that the instance of `Company data submission` activity within the current case has a status not yet *Active*. Notice that we assume an ordered set of possible values for the status (*Created < Inactive < Active < Suspended < Resumed < Completed*), and at most one instance of the activity `Company data submission` exists within a case, because of the construction rules of the instances of the workflow. Therefore, the condition extracts the activity of type `Company data submission` not yet started. If this instance exists, the *Start* link is followed and the `Personal Data Submission` activity is started (i.e., its status in the database is set to *Active*). The user submits his own information and the Modify unit updates the database, then the *Complete* link closes the activity and redirects the user to the home page.
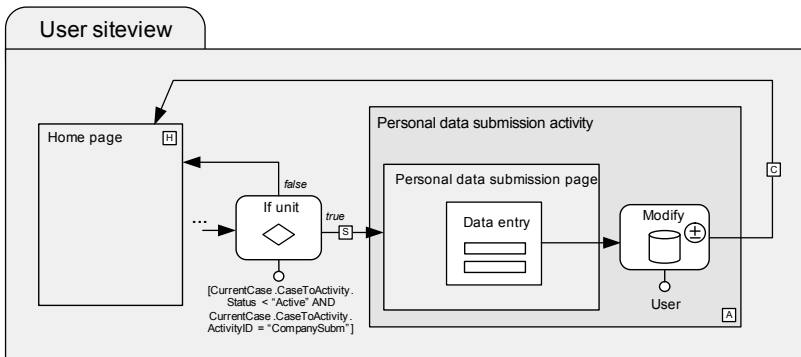
Figure 9.14. Example of hypertext representing the Personal data submission activity.

9.4.3 Context-Aware Web Applications

WebML has also been applied to the design of adaptive, context-aware Web applications (Ceri et al., 2003, 2006, 2007). The overall design process for context-aware applications follows the activity flow typically used for conventional Web applications. However, some new issues must be considered for modeling and exploiting the context at the data level and for modeling adaptive behaviors in the hypertext interface.

9.4.3.1 Modeling User and Context Data

During data design, the user and context requirements can be translated into three different subschemas complementing the application data (see Figure 9.15):

- The *User subschema*, which clusters data about users and their access rights to application data. In particular, the entity `User` provides a basic profile of the application's users, the entity `Group` allows access rights for a group of users to be managed, and the entity `SiteView` allows users (and user groups) to be associated with specific hypertexts. In the case of adaptive context-aware applications, users may require different interaction and navigation structures, according to the varying properties of the context.
- The *Personalization subschema*, which consists of entities from the application data associated with the `User` entity by means of relationships expressing user preferences for some entity instances, or the user's ownership of some entity instances. For example, the relationship between the entities `User` and `UserComment` in Figure 9.15 enables the selection and the presentation to the user of the comments she has posted. The relationship between the entities `User` and `Movie` represents the preferences of the user for specific movies. The role of this subschema is to support the customization of contents and services, which is one relevant facet of adaptive Web applications.
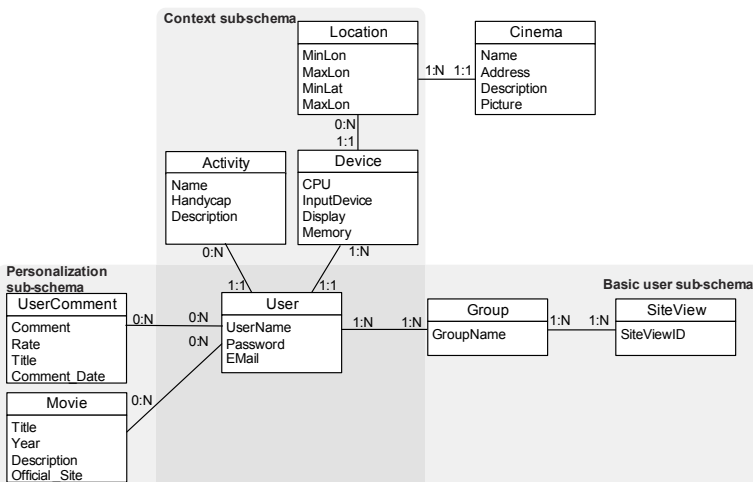

Figure 9.15. Three subschemas representing context data.

- The *Context subschema*, including entities such as `Device`, `Location,` and `Activity`, which describe context properties relevant for providing adaptivity. Context entities are connected to the entity `User` to associate each user with his (personal) context.

9.4.3.2 Identifying Context-Aware Pages

During hypertext design, adaptive requirements are considered to augment the application's front end with reactive capabilities. As illustrated in Figure 9.16, context-awareness in WebML can be associated with selected pages, and not necessarily with the whole application. Location-aware applications, for example, adapt "core" contents to the position of a user, but typical "access pages" (including links to the main application areas) might not be affected by the context of use.

We therefore tag adaptive pages with a *C* label (standing for "Context-aware") to distinguish them from conventional pages. This label indicates that some adaptivity actions must be associated with the page. During application execution, such actions must be evaluated prior to the computation of the page, since they can serve to customize the page content or to modify the navigation flow defined in the model.
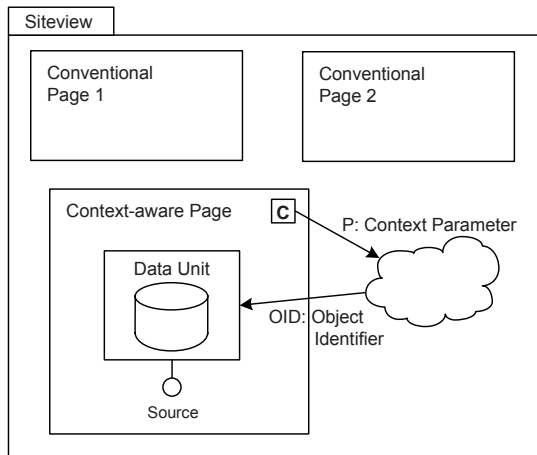
Figure 9.16. Hypertext schema highlighting context-aware pages. Context-aware pages are labeled with a "C" and are associated with a context cloud.

As shown in Figure 9.16, adaptivity actions are clustered within a *context cloud*. The cloud is external to the page, and the adaptivity actions that it clusters are kept separate from the page specification. Such a notation highlights the different roles played by pages and context clouds: The former

act as providers of content and services, the latter act as modifiers of such content and services.

In order to monitor the state of the context and execute adaptivity actions, C-pages must be provided with autonomous intervention capabilities. The standard HTTP protocol underlying most of today's Web applications implements a strict pull paradigm. In the absence of a proper push mechanism, reactive capabilities can therefore be achieved by periodically refreshing the viewed page and by triggering the execution of adaptivity actions before the computation of the page content. This polling mechanism "simulates" the active behavior necessary for making pages sensitive to the context changes.

9.4.3.3 Specifying Adaptivity Actions in Context Clouds

Context clouds contain adaptivity actions expressed as sequences of WebML operations and are associated with a page by means of a directed arrow, i.e., a link, exiting the C label. This link ensures communication between the page logic and the cloud logic, since it can transport parameters derived from the content of the page, useful for computing the actions specified within the cloud. Vice versa, a link from the cloud to the page can transport parameters computed by the adaptivity actions, which might affect the page contents with respect to a new context.

The specification of adaptivity actions relies both on the use of the standard WebML primitives and on a few novel constructs, related to the acquisition and use of context data:

1. *Acquisition and management of context data*. This may consist of the retrieval of context data from the context model stored within the data source, or of the acquisition of fresh context data provided by device- or client-side-generated URL parameters, which are then stored in the application data source. These are the first actions executed every time a C-page is accessed, for gathering an updated picture of the current context.
2. *Condition evaluation*. The execution of some adaptivity actions may depend on some conditions, e.g., evaluating whether the context has changed and hence triggering some adaptivity actions.
3. *Page content adaptivity*. Parameters produced by context data acquisition actions and by condition evaluation can be used for page computation. They are sent back to the page by means of a link exiting the context cloud and going to the page. The result is the display of a page where the content is adapted to the current context.
4. *Navigation adaptivity*. The effect of executing the adaptivity actions within the context cloud can be the redirection to a different page. The

specification of context-triggered navigation just requires a link exiting the context cloud to be connected to pages other than the cloud's source page.

5. *Adaptivity of the hypertext structure.* To deal with coarse-grained adaptivity requirements, e.g., the change of device, role, or activity, the adaptivity actions may lead to the redirection toward a completely different site view.

6. *Adaptivity of presentation properties.* To support finer-grained adjustments of the interface, the adaptivity actions may induce the run-time modification of the presentation properties (look and feel, content position and visibility, and so on).
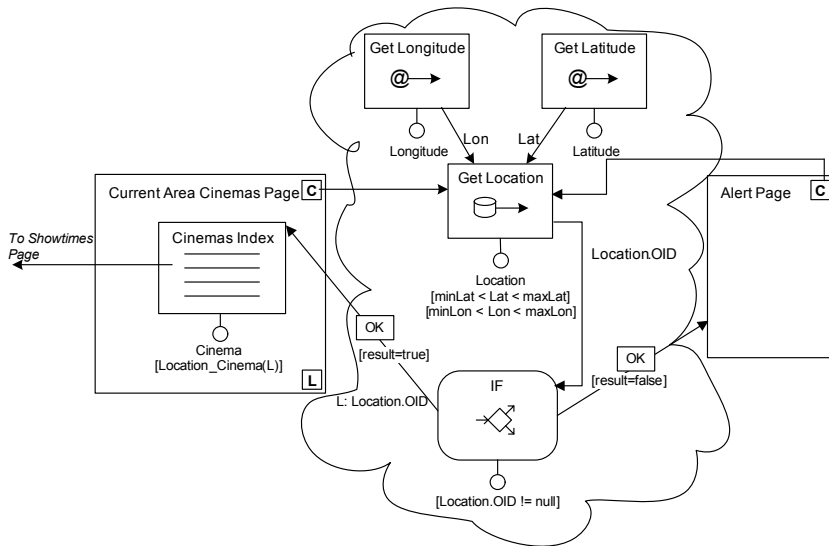
Figure 9.17. The WebML specification of adaptivity actions providing users with context-aware information about cinemas.

Figure 9.17 illustrates an example of adaptivity actions, applied to the `Current Area Cinemas` page. Upon page access, some adaptivity actions in the cloud are executed, which may change the content of the page based on the geographical position of the user. Specifically, the user's `Latitude` and `Longitude` are retrieved by the `Get Longitude` and `Get Latitude` units, which are examples of the *GetClientParameter* operation unit, introduced in WebML to access context data sensed at the client side. In the example, the two parameters `Longitude` and `Latitude` represent the position coordinates sensed through a user's device equipped with a GPS module. The retrieved position values are used by the `Get Location` unit to identify a (possible)

location stored in the database for the current user's position. `Get Location` is a *Get Data* unit, a content unit for retrieving values (both scalars and sets) from an entity of the data model without displaying them on a page. The location OID is evaluated through an *If* unit: If it is not null (i.e., the sensed coordinates fall into a location stored in the application data source), the list of cinemas in that location is visualized in the `Current Area Cinemas` page; otherwise, the user is automatically redirected to the `Alert` page, where a message notifies of the absence of information about cinemas in the current area.

Figure 9.17 also models the `Alert` page as context-aware; in particular, this page shares its adaptivity actions with the `Current Area Cinemas` page. Therefore, as soon as an automatic refresh of the `Alert` page occurs, the shared actions are newly triggered and the application is adapted to the user's new position.

More details on the WebML extensions for adaptivity and context-awareness and on their implementation in WebRatio can be found in Ceri et al. (2003, 2006, 2007).

9.5 INDUSTRIAL EXPERIENCE

We conclude the illustration of WebML with an overview of the most significant aspects of transferring model-driven development to industrial users. The reported activities are based on WebML and WebRatio, but we deem that the achieved results demonstrate the effectiveness and economic sustainability of MDD in a more general sense. As a case study, we focus on the applications developed by Acer EMEA, the Europe, Middle East, and South Africa branch of Acer, for which five years of experience and data are available. In particular, we will review some of the realized projects, highlighting their functional and nonfunctional requirements, their dimensional parameters, and the key aspects of their development, deployment, evolution, and economic evaluation. The experience started with the first version of the Acer-Euro application (http://www.acer-euro.com), which aimed at establishing a software infrastructure for managing and Web-deploying the marketing and communication content of an initial group of 14 countries out of the 31 European Acer national subsidiaries. The content of Acer-Euro 1.0 included the following main areas: *About Acer*, *Products*, *News*, *Service & Support*, *Partner Area*, and *Where to buy*.
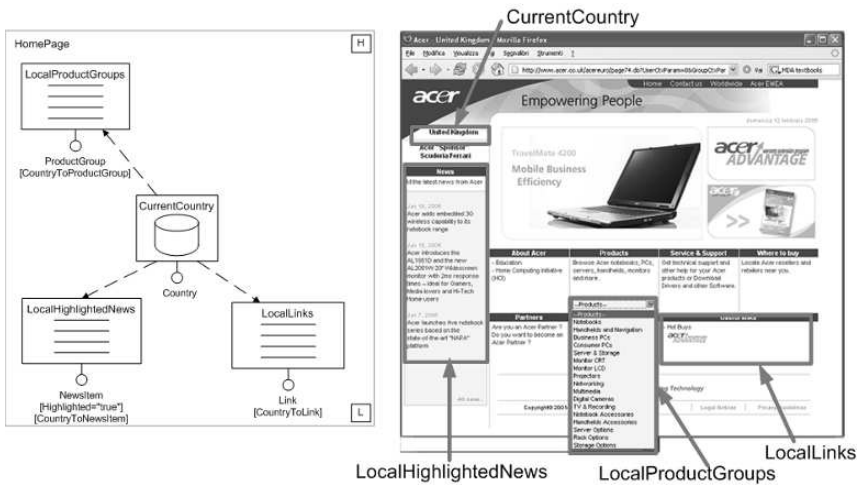
Figure 9.18. The WebML specification of the home page of a national site of Acer-Euro (left) and its rendition in HTML (right).

Figure 9.18 shows the home page of a national site of Acer-Euro (left) and its rendition in HTML generated by WebRatio. The Acer-Euro 1.0 system supported two main functions:

1. *Content publishing:* comprising the architecture, tools, and processes to make content about the Acer European Web sites available on the Web to the users of the target countries.
2. *Content management:* comprising the architecture, tools, and processes needed to gather, store, update, and distribute to the destination countries the content related to the Acer European Web sites.

Figure 9.19 shows the schedule and milestones of the Acer-Euro 1.0 project. Only 7 weeks elapsed from the approval of the new site map and visual identity to the publishing of the 14 national Web sites and to the delivery of the CMS to Acer employees. In this period, two distinct prototypes were formally approved by the management: Prototype 1, with 50% of functionality, was delivered at the end of week 2; prototype 2, with 90% of functionality, at week 5. Overall, nine prototypes were constructed in six weeks: two formal, seven for internal assessment.

The development team consisted of four persons: one business expert and one junior developer from Acer, and one analyst and one Java developer from Politecnico di Milano.
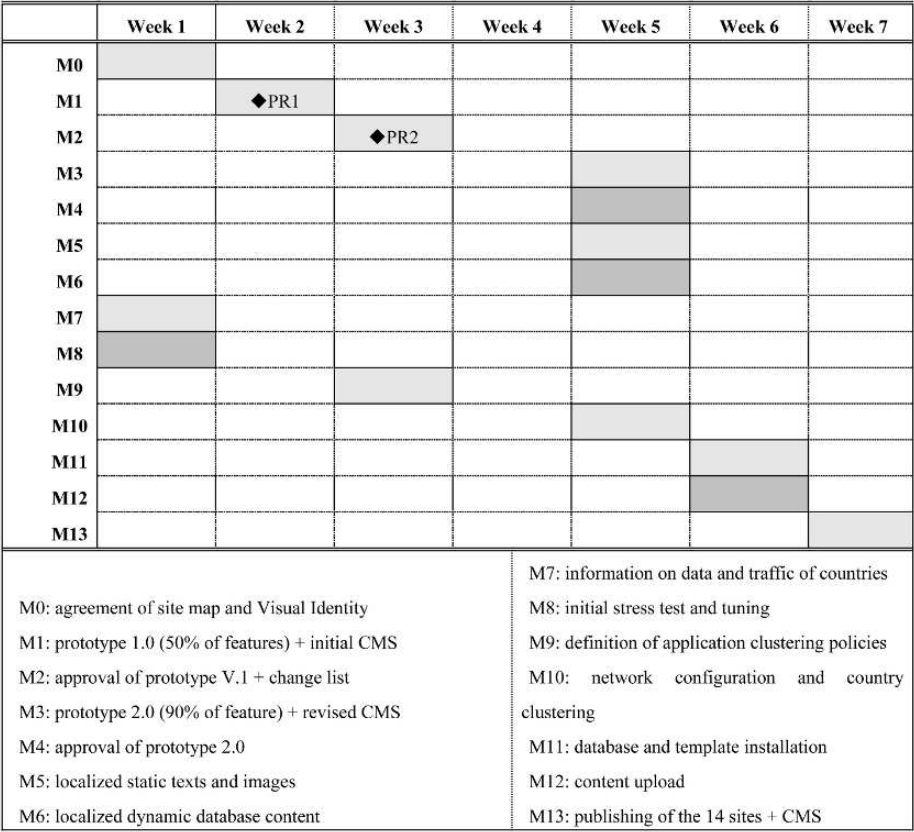
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
M0	▓						
M1		◆PR1					
M2			◆PR2				
M3					▓		
M4					▓		
M5					▓		
M6					▓		
M7	▓						
M8	▓						
M9			▓				
M10					▓		
M11						▓	
M12						▓	
M13							▓

M0: agreement of site map and Visual Identity

M1: prototype 1.0 (50% of features) + initial CMS

M2: approval of prototype V.1 + change list

M3: prototype 2.0 (90% of feature) + revised CMS

M4: approval of prototype 2.0

M5: localized static texts and images

M6: localized dynamic database content

M7: information on data and traffic of countries

M8: initial stress test and tuning

M9: definition of application clustering policies

M10: network configuration and country clustering

M11: database and template installation

M12: content upload

M13: publishing of the 14 sites + CMS

Figure 9.19. The schedule and milestones of the Acer-Euro 1.0 project.

Figure 9.19 shows the most relevant figures of the project: only six weeks of development plus one week of testing were sufficient for analyzing, designing, implementing, verifying, documenting, and deploying a set of midsized, functionally complex, multilingual Web applications. As illustrated by the dimensional and economic parameters reported in Table 9.6, such result has to be ascribed to

1. The high degree of automation brought to the process by the use of the model-driven approach: More than 90% of the application and database code were synthesized automatically by the WebRatio development environment from the WebML models of the applications, without the need to manually intervene on the produced code.
2. The overall productivity of the development process: The productivity value is obtained by counting the number of function points (FPs) of the project and dividing this value by the number of staff-months

employed in the development. The result is an average productivity rate of 131.5 FP/staff month, which is 30% greater than the maximum value expected for traditional programming languages in the Software Productivity Research Tables (SPR, 2006). This latter result is a consequence of the former: High automation implies a substantial reduction of the manually written repetitive code and a high reuse of design patterns.

Table 9.6 Main Dimensional and Economic Parameters of the Acer-Euro Project

Class	Dimension	Value
Time & effort	Number of elapsed workdays	49
	Number of development staff-months (analysts and developers)	6 staff-months (6 weeks × 4 persons)
	Total number of prototypes	9
	Average elapsed man days between consecutive prototypes	5,4
	Average number of development man days per prototype	15,5
Size	Number of localized B2C Web sites	14
	Number of localized CMS applications	4 (Admin, News, Product, Other)
	Number of supported languages	12 for B2C Web sites, 5 for CMS
	Number of data entry masks	39
	Number of automatically generated database tables	46
	Number of automatically generated database views	82
	Number of automatically generated database queries	279 for extraction, 89 for update
	Number of automatically generated JSP page templates	48
	Number of automatically generated or reused Java classes	250
	Number of automatically generated Java lines of code	12,500 Noncommented lines of code
Degree of automation	Number of manually written SQL statements	17 (SQL constraints)
	Percentage of automatically generated SQL code	96%
	Number of manually written/adapted Java classes /JSP templates	10% JSP templates manually adapted
	Percentage of automatically generated Java and JSP code	90% JSP templates, 100% Java classes
Productivity	Number of function points	177 (B2C web site) + 612 (CMS) = 789
	Average number of FP delivered per staff-month	131.5

Another critical success factor has been the velocity in focusing the requirements, thanks to the rapid production of realistic prototypes. At the end of week 2, the top management could already evaluate an advanced

prototype, which incorporated 50% of the requested functionality, and this initial round of requirement validation proved essential to the delivery of a compliant solution in such a limited time. With respect to traditional prototyping, which exploits a simplified architecture, WebRatio generates code directly for the actual delivery platform; in this way, stress test and architecture tuning could already start at week 1 on the very first prototype, greatly improving the parallelism of work and further reducing time to market.

The benefits of MDD were manifested not only in the development of the first version, but were even more sensible in the maintenance and evolution phase. Figure 9.20 shows the timeline of the additional releases and spin-off projects of Acer-Euro. Four major releases of Acer-Euro were delivered between 2001 and 2006, and the number of applications grew from the initial 5 to 13 intranet and Internet applications, serving more corporate roles and supporting more sophisticated workflow rules.
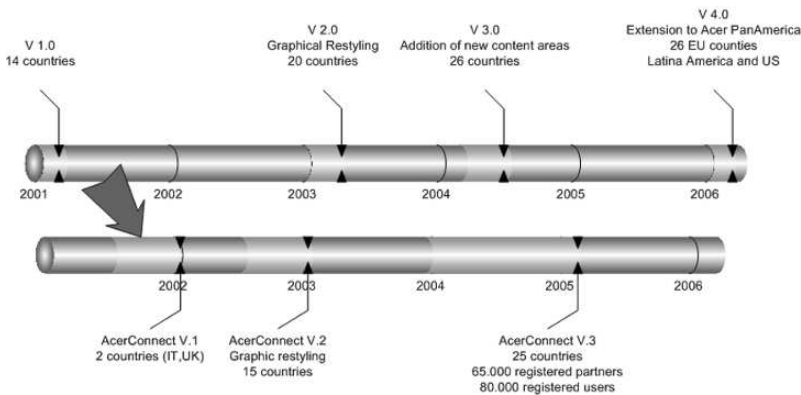

Figure 9.20. The evolution of the Acer-Euro project in five years.

At the end of 2005, Acer-Euro was rolled out in 26 European countries and extended also to the Acer Pan-American subsidiaries, including Latin America and the United States. As early as June 2001, an extension of the Acer-Euro platform was scheduled, to address the delivery and management of content for the channel operators (Acer partners). This spin-off project, called Acer Connect, is a multi-actor extranet application targeted to Acer partners, characterized by the following features:

1. the segmentation of the users accessing the site into a hierarchy of groups corresponding to both Acer's and partners' business functions
2. the definition of different access privileges and information visibility levels to groups

3. the provision of an Acer European administration role, able to dynamically perform via the Web all administrative and monitoring tasks
4. the provision of an arbitrary number of nation-based and partner-based administration roles, with responsibility for local content creation and publishing, and local user administration
5. a number of group-tailored Web applications (e.g., sales, marketing) targeting content to corporate-specific or partner-specific user communities
6. the management of administrative and business functions in multiple languages flexibly set by administrators and users
7. a security model storing group and individual access rights into a centrally managed database, to enforce global control over a largely distributed application
8. content personalization based on group-specific or user-specific characteristics, for ensuring one-to-one relationships with partners
9. advanced communication and monitoring functions for the effective tracking of partners' activity and of Acer's quality of services

The first version of Acer Connect was deployed in Italy and the UK in December 2001, after only seven months of development and with an effort of 24 staff-months. Today, Acer Connect is rolled out in 25 countries and hosts 65,000 registered partners, delivering content and services to a community of over 80,000 users. Acer Connect and Acer-Euro share part of the marketing and communication content, and therefore the former project was realized as an evolution of the latter; starting from the data model of Acer-Euro, the specific functions of Acer Connect were added, and new applications were modeled and automatically generated. The model-driven approach greatly reduced the complexity of integration, because the high-level models of the two systems were an effective tool for reasoning about the functionality to reuse and develop.

Besides Acer Connect, several other projects were spun off, to exploit the customer and partner communities gathered around these two portals. Figure 9.21 overviews the delivered B2C projects, which collectively total over 10,800,000 visits per month.

As a remark on the long-term sustainability of MDD, we note that, despite their complexity and multinational reach, both Acer-Euro and Acer Connect are maintained and evolved by one junior developer each, working on the project at part time. In total, only 5 junior developers, allocated to the projects at part time, maintain the 56 mission-critical Web applications implemented by Acer with WebML.
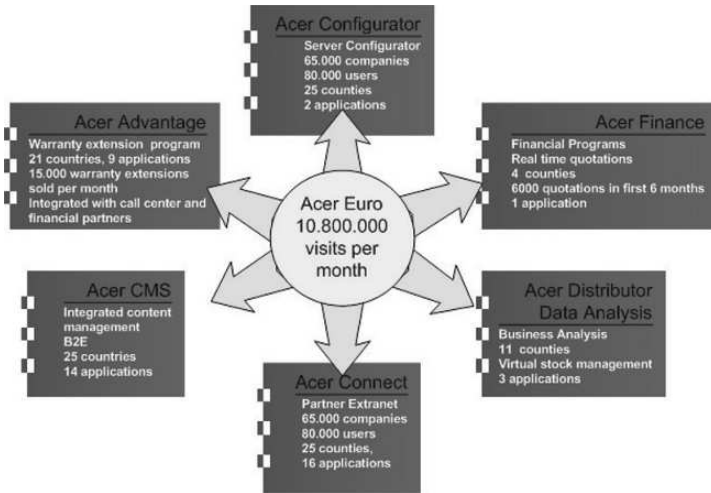
Figure 9.21. The main applications developed in Acer with WebML.

On the negative side of MDD, the initial training and switching costs have been reported as the most relevant barrier. MDD requires nontechnical knowledge on the modeling of software solutions, which must be acquired with a mix of conventional and on-the-job training. Furthermore, developers have their own previous consolidated skills and professional history, and switching to a completely new development paradigm is felt to be a potential risk. Acer estimates that it takes developers from 4 to 6 months to become fully acquainted and productive with MDD, WebML, and WebRatio. However, Acer's figures demonstrate that the initial investment in human capital required by MDD pays off in the mid-term. The number of applications developed and maintained per unit of development personnel increases with the developers' expertise and exceeds 10 fully operational, complex, and distributed Web applications per developer.

9.6 CONCLUDING REMARKS

In this chapter we have described the Web Modeling Language, a conceptual notation for specifying the design of complex, distributed, multi-actor, and adaptive applications deployed on the Web and on service-oriented architectures using Web services. WebML was born in academia but soon spun off to the industrial battlefield, where it faced the development of complex systems with requirements often exceeding the expressive power of the language. This fruitful interplay of academic design and industrial experience made the language evolve from a closed notation for data-centric

Web applications to an open and extensible framework for generalized component-based development. The core capability of WebML is expressing application interfaces as a network of collaborating components, which sit on top of the core business objects. WebML incorporates a number of built-in, off-the-shelf components for data-centric, process-centric, and Web service-centric applications and lets developers define their own components, by wrapping existing software artifacts and reverse-engineering them. In other words, the essence of WebML boils down to a standard way of describing components, their interconnection and passage of parameters, their exposition in a user interface, and the rules for generating code from their platform-independent model.

This flexibility allowed several extensions of the language, in the direction of covering both new application requirements and deployment architectures. The ongoing work is pursuing a number of complementary objectives:

1. Extending the model-driven approach to all the phases of the application life cycle: WebML is being used as a vehicle to investigate the impact of MDD on development activities like business requirement elicitation and reengineering, cost and effort estimation, testing, quality evaluation, and maintenance.
2. Extending the capability of the user interface beyond classical hypertexts: The expressive power of WebML is presently inadequate to express Rich Internet Applications and classical client-server applications; research is ongoing to identify the minimal set of concepts needed to capture the Web interfaces of the future.
3. Broadening the range of deployment platforms: WebML and WebRatio are being extended to target code generation for nonconventional infrastructures. A version of WebRatio for digital television has been already built, and experimentation is ongoing for deploying applications on top of embedded systems and mobile appliances for the DVB-H standard.

REFERENCES

Baresi, L., Fraternali, P., Tisi, M., and Morasca, S., 2005, Towards model-driven testing of a Web application generator. *Proceedings 5th International Conference on Web Engineering* (ICWE'05), Sydney, Australia, pp. 75–86.

Beck, K., 1999, Embracing change with extreme programming. *IEEE Computer*, **32**(10): 70–77.

Boehm, B., 1988, A spiral model of software development and enhancement. *IEEE Computer*, **21**(5): 61–72.

Booch, G., Rumbaugh, J., and Jacobson, I., 1999, *The Unified Modeling Language User Guide (Object Technology Series),* Addison-Wesley, Reading, MA.

Brambilla, M., 2003, Extending hypertext conceptual models with process-oriented primitives. *Proceedings Conceptual Modeling* (ER 2003), Chicago, IL, pp. 246–262.

Brambilla, M., Ceri, S., Comai, S., Fraternali, P., and Manolescu, I., 2003, Specification and design of workflow-driven hypertexts. *Journal of Web Engineering*, **1**(2): 163–182.

Brambilla, M., Ceri, S., Comai, S., and Tziviskou, C., 2005, Exception handling in workflow-driven Web applications. *Proceedings World Wide Web International Conference* (WWW'05), Chiba, Japan, May 10–13, pp. 170–179.

Brambilla, M., Ceri, S., Fraternali, P., and Manolescu, I., 2007, Process modeling in Web applications. *ACM Transactions on Software Engineering and Methodology*. In print.

Brambilla, M., Ceri, S., Passamani, M., and Riccio, A., 2004, Managing asynchronous Web services interactions. *Proceedings ICWS 2004*, pp. 80–87.

Brambilla, M., and Tziviskou, C., 2005, Fundamentals of exception handling within workflow-based Web applications. *Journal of Web Engineering*, **4**(1): 38–56.

Ceri, S., Daniel, F., Facca, F., Matera, M., and the MAIS Consortium, 2006, Front-end methods and tools for the development of adaptive applications. In *Mobile Information Systems. Infrastructure and Design for Flexibility and Adaptivity*, B. Pernici, ed., Springer-Verlag, pp. 209–246.

Ceri, S., Daniel, F., and Matera, M., 2003, Extending WebML for modelling multi-channel context-aware Web applications. *Proceedings WISE '03 Workshops*, IEEE Press, pp. 225–233.

Ceri, S., Daniel, F., Matera, M., and Facca, F., 2007, Model-driven development of context-aware Web applications. *ACM Transactions on Internet Technology*, **7**(1), Article No. 2.

Ceri, S., Fraternali, P., and Bongio, A., 2000, Web Modeling Language (WebML): A modeling language for designing Web sites. *Computer Networks*, **3**(1–6): 137–157.

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., and Matera, M., 2002, *Designing Data-Intensive Web Applications,* Morgan Kaufmann, San Francisco.

Conallen, J., 2000, *Building Web Applications with UML (Object Technology Series),* Addison-Wesley, Reading, MA.

Fraternali, P., Lanzi, P.L., Matera, M., and Maurino, A., 2004, Model-driven Web usage analysis for the evaluation of Web application quality. *Journal of Web Engineering*, **3**(2): 124–152.

Fraternali, P., Matera, M., and Maurino, A., 2003, Conceptual-level log analysis for the evaluation of Web application quality. *Proceedings LA-WEB 2003*, IEEE Press, pp. 46–57.

Garzotto, F., Paolini, P., and Schwabe, D., 1993, HDM—A model-based approach to hypertext application design. *ACM Transactions on Information Systems*, **11**(1): 1–26.

Kruchten, P., 1999, *The Rational Unified Process: An Introduction*, Addison-Wesley, Reading, MA.

Isakowitz, T., Sthor, E.A., and Balasubranian, P., 1995, RMM: A methodology for structured hypermedia design. *Communications of the ACM*, **38**(8): 34–44.

Lanzi, P.L., Matera, M., and Maurino, A., 2004, A framework for exploiting conceptual modeling in the evaluation of Web application quality. *Proceedings ICWE 2004*, Springer-Verlag, pp. 50–54.

Manolescu, I., Brambilla, M., Ceri, S., Comai, S., and Fraternali, P., 2005, Model-driven design and deployment of service-enabled Web applications. *ACM Transactions on Internet Technology,* **5**(3): 439–479.

Meo, R., and Matera, M., 2006, Designing and mining Web applications: A conceptual modeling approach. In *Web Data Management Practices: Emerging Techniques and Technologies*, A. Vakali and G. Pallis, eds., Idea Group Publishing, Hershey, PA.

SPR (Software Productivity Research), 2006, SPR Programming Language Table—Version PLT2005a. Retrieved February 2006 from http://www.spr.com.

WebModels, 2006. WebRatio Tool Suite. Retrieved October 2006 from http://www.webratio.com.

W3C, 2006, WSDL Web Service Description Language. Retrieved October 2006 from https://www.w3.org/2002/ws/desc.