Engineering Adaptive Web Applications

Von Fakultät für Elektrotechnik und Informatik der
Universität Hannover
zur Erlangung des Grades

Doktor der Naturwissenschaften

Dr. rer. nat.

genehmigte Dissertation von

Peter Dolog

geboren am
28. August 1976 in Brezno, Slowakische Republik

2006

Peter Dolog

Engineering Adaptive Web Applications

Dr. rer. nat. Dissertation

2006

Zusammenfassung: Nutzer von Web-Anwendungen stammen sich aus verschiedensten Nutzergruppen, die sich in ihrem Hintergrund, ihrer technischen Ausstattung, ihrem politischen und sozialen Kontext, ihren Interessen und Zielen, usw. unterscheiden. Dementsprechend haben unterschiedliche Nutzertypen auch unterschiedliche Anforderungen an Web-Anwendungen. Diese Anforderungen können durch das Angebot mehrerer Varianten dieser Anwendungen erfüllt werden. Die Implementierung und Wartung dieser Varianten macht die Entwicklung solcher Anwendungen allerdings komplexer, und erfordert daher die Einführung spezieller Ansätze, um trotz der zusätzlichen Komplexität effizienten Entwurf und Entwicklung zu gewährleisten. Die bisher vorgeschlagenen Methoden zur Entwicklung von Web- Anwendungen berücksichtigen die Problematik der Nutzerheterogenität und der Notwendigkeit, darauf angepasste Anwendungsvarianten parallel zu entwickeln, nicht.

In dieser Arbeit schlagen wir ein neues Framework zum Entwurf solcher Anwendungen vor, das den Entwicklungsprozess um spezielle Techniken für den Entwurf adaptiver, individuell anpassbarer Web- Anwendungen erweitert. Die Adaptivität und Anpassbarkeit wird as Auswahl aus Feature-Varianten entsprechend eines Nutzerprofiles aufgefasst, die als Teil der Web-Anwendung angeboten werden.

Das Framework basiert auf dem Ansatz des Domain Engineering. Für die Anwendung im Bereich adaptiver Web-Anwendungen wurde diese Methode wurde um zusätzliche Entwurfsabstraktionen erweitert, die es ermöglichen, die von der Anwendung angebotenen Informationen, die Art der Informationsausgabe sowie Benutzereigenschaften vollständig getrennt voneinander zu modellieren.

Die Adaption basiert auf der Anwendung von Feature-Modellen, wobei generelle Features die allgemeinen Teile der Anwendung repräsentieren, und variable Features diejenigen Teile der Anwendung, die zur Design- oder Laufzeit an die Benutzer adaptiert werden. Für die Modellierung der adaptiven Benutzerführung und der Adaption des präsentierten Contents stellen wir Modellierungsrichtlinien vor. Als Modelle für die adaptive Navigation werden dabei endliche Zustandsautomaten verwendet. Die Feature-Konfiguration wird durch Constraints an den Zuständen und Zustandsübergängen spezifiziert. Abhängig von im Benutzerprofil modellierten Eigenschaften bestimmen diese Constraints, welche Links aktiv sind, welche Zuständsübergänge möglich sind, usw. Über als Seiteneffekte spezifizierte Aktionen wird festgelegt, wie sich das Benutzerprofil verändert.

Die adaptive Konfiguration der Content-Bestandteile wird mit Hilfe von Kollaborationsdiagrammen spezifiziert. Durch die Kollaboration der Features, die für die Beschreibung des Contents verwendet werden, mit denen, die für Spezifikation der Content-Umgebung verwendet werden, wird festgelegt, aus welchen Content-Bestandteile ein Hypertextknoten zusammengesetzt wird. Constraints auf den Kollaborations-Nachrichten bieten eine Abstraktion, um Content-Bestandteile auf Basis des Benutzerprofils gezielt ein- oder auszublenden.

Die Anwendbarkeit dieses neuen Ansatzes wird an mehreren Anwendungen gezeigt. Ein Generator für adaptive Benutzerführung durch elektronische Lerneinheiten wurde auf der Grundlage dieses Verfahrens entworfen und implementiert. Die vorgeschlagenen Entwurfsabstraktionen können als Ergänzung zu bestehenden Methoden eingesetzt werden; dies wird durch Integration mit dem WebML-Ansatz belegt. Weiterhin wird diskutiert, wie die durch Feature-Modelle ermöglichte Variabilität im Zusammenhang eines Reasoning-Ansatzes verwendet werden kann, um adaptive Annotationen, Benutzerführung und automatische Anfragen nach bestimmten Informationen in einer verteilten Umgebung zu generieren. In diesem Zusammenhang wird auch die serviceorientierte Architektur dargestellt, innerhalb derer diese Modelle basierend auf Semantic-Web-Beschreibungssprachen eingesetzt werden.

Schlagworte: Domain Engineering, Adaptive Web Applications, Conceptual Modeling

Abstract: Applications on the Web are accessible for users with different background, technical environment used, political, social environment where they reside, interests, goals and so on. The different user types have slightly different requirements for features which such a Web application should have. The different requirements might be satisfied by different variants of features maintained and provided by Web applications. An adaptive Web application can be seen as a family of Web applications where application instances are those generated for particular user based on his characteristics relevant for a domain.

In this thesis, we propose a new framework which extends a development process of Web applications with techniques required when designing such adaptive customizable Web applications.

The framework is based on domain engineering. The domain engineering approaches proposed so far have been applied to product family engineering with variability resolution at the application design time. We propose a domain engineering approach for adaptive Web applications where some variability is resolved also at the run-time taking a user profile into account. The framework is provided with design abstractions which deal separately with information served by the Web application, environment used to deliver the information, and user characteristics which are observed and used to constrain the information and the environment selection.

Our customization approach is based on the feature models. Common features are provided to all users of the Web application and the variable features represent different variants which are selected according to a user profile either at the design or at the run-time. We propose guidelines for modeling adaptive navigation and adaptive content configuration for Web applications. The adaptive navigation is modeled by the state machines. The resolution of variable features and variation points is specified by constraints on states and transitions where characteristics of a user are checked. The constraints determine whether particular link is accessible, which state should be taken as a target state, which state can be entered and so on. Furthermore, side-effect actions specify changes of user profiles; i.e., evolution of user profiles.

The adaptive configuration of content fragments is specified by the collaboration diagrams. The features, used for a content description and environment to deliver the content, collaborate among one another to provide information fragment requested within a hypertext node. Some of the collaboration messages may be constrained. This provides abstraction for enabling/suppressing some content fragments according to information about a user.

The use of the approach is demonstrated on a generator provided for the adaptive guide through an electronic course. Complementariness of the proposed design abstractions to the other methods is demonstrated on the case of integration with the WebML platform. The variability in feature models is also discussed in the context of a reasoning approach over the domain, the resource, the navigation, and the user models to generate adaptive annotations, navigation support, and queries for information in distributed environments. The service oriented architecture, where those models are used, is also introduced utilizing semantic Web description formats for information exchange.

Keywords: Domain Engineering, Adaptive Web Applications, Conceptual Modeling

Acknowledgements

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The World Wide Web has had an extremely significant impact on access to information and information services. Web-based applications are influencing many domains such as business, commerce, banking and learning. The Web has simplified access to information and information services, enabling a variety of users with different backgrounds, social situations, and so on to participate.

The increasing complexity of such applications calls for engineering methods for developing efficient high quality software applications with all the appropriate performance, features and services which are required. The methods are required to handle the application development process in an efficient manner to meet time, budget and resources criteria. Web engineering is an area whose purpose is to answer the questions connected with the Web application engineering problems mentioned.

Web engineering is a multi-disciplinary area influenced by several communities such as multimedia, hypertext/hypermedia, human-computer interaction, software engineering and, information engineering. Information on the Web is disseminated by means of media (i.e. text items, video or audio sequences, images, and their combinations) actively interconnected by links. Usability issues, navigation and interaction support are other characteristics which are to be considered in Web-based systems. Each of these aspects can be engineered separately as an orthogonal activity in systematic methodologies.

The most significant feature of information intensive applications on the Web is the possibility of non sequential navigation among items of information or different documents; i.e. there is no definite order that determines a sequence in which the text is read. The applications provide rather exploratory user interfaces which are often adapted to different audiences or different devices according to the 'anywhere, anytime and anybody' paradigm. The applications for the Web are developed for multiple platforms with several accessing approaches to information bases. Information content constitutes not only regularly structured data but often unstructured (multimedia) items in the applications. The development of such applications has to take into account also aesthetic and cognitive aspects as well that traditional software engineering environments do not support [NN95].

The new conditions introduced by the Web applications have meant that the "one size fits all" approach is not suitable for building such applications any more. The applications should reflect the different requirements of users with different back-

grounds, technical, political and social environments, interests, goals and so on. As a result, a Web engineering method should meet following requirements:

Support for common and variable features of Web applications. A method for engineering Web applications should be able to handle common and variable features which are requested by current and potential users and shareholders of Web applications at design time.

Support for personalization. It is not always possible to determine which features are needed and which are not needed at design time. Sometimes, this decision has to be postponed to the final application (to its runtime). The decision is usually made according to maintained user features, thus personalization should be addressed in development methods as well.

1.1   Problems Addressed in this Thesis

The central goal of this thesis is to develop a methodology capable of supporting a design which takes account of common features and variabilities in a Web system family as well as personalization.

Customization and personalization can be seen as a selection over variants of Web application features. The variants can be selected according to shareholders' requirements or dynamic and evolving user profiles. The Web applications are usually a kind of information serving systems where the information items are served according to a hypertext paradigm possibly utilizing multimedia. To support better orientation of a user, information is usually served by using an environment. The environment can for example support users by providing information about a level or structural unit he is currently dealing with. It can indicate the current position of a user within information space and environment level. It can provide further functions and links, simplifying navigation to other levels, backtracking and so on. Environments for delivering the same information can differ either by employed features or their configuration. The customization design considers in addition variability in binding information to environment features as well as the way in which selection depends on a user interaction.

Customizable and adaptive Web applications can be seen as members of a Web application family in a particular domain (serving content from the domain) where all of them share a common part (common features) and they differ in variable parts and/or feature selection mechanisms left for the dynamic runtime environment.

Domain engineering based methods proposed for software systems such as the product line practices [Wit96, BFK$^+$99, PBvdL05] or generative programming [CE00] deal with product families. Based on the above discussion, the Web software applications bring new dimensions to the development and system design.

Web solutions have often been used for achieving availability of a (large amount of) information from several distributed sources, serving users who might be distributed as well. Recently, Web applications connecting/orchestrating several Web services have emerged as well.

In order to cover the peculiar characteristics of Web applications, the generic software processes need to be refined and augmented with activities to create particu-
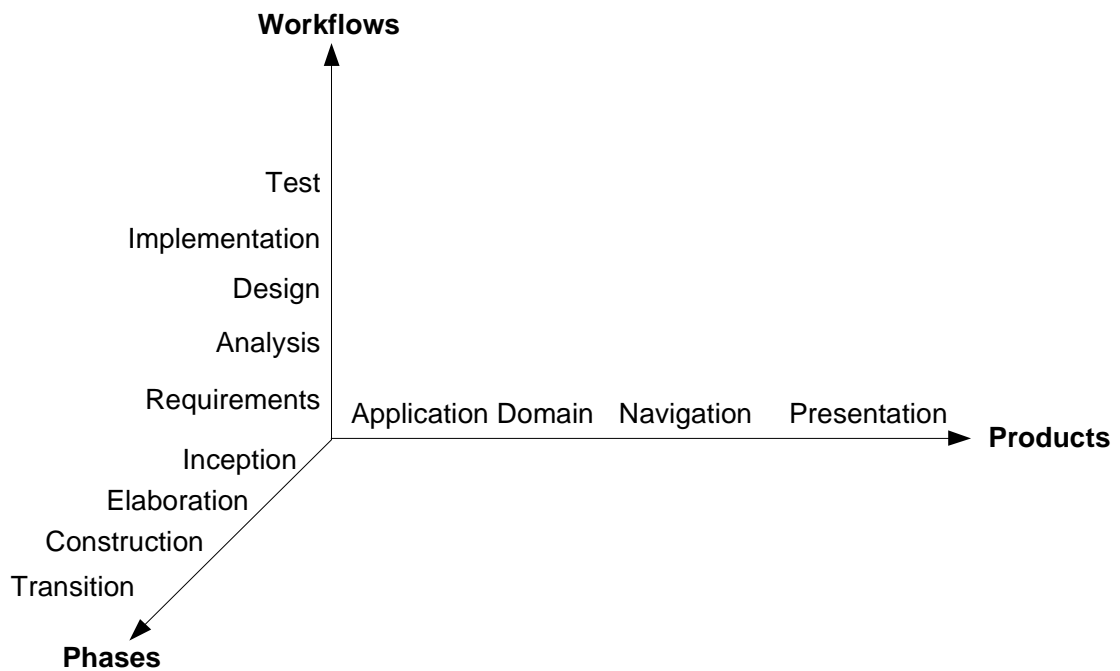
Figure 1.1: Phases, Workflows and Products in Web Development Process

lar components/views/layers. There have been several proposals for Web development methods, describing specific activities for Web application development, such as OOHDM [SR98], WebML [CFM02], and UML-based Web Engineering [HK00], or reference models like the IMPACT-A method [LBW99]. All these methods have in common three high level activities, referring to the engineering of the *application domain*, *navigation*, and *presentation*.

Figure 1.1 depicts the dimensions of the Unified Process [JBR99] supplemented with the dimensions reflecting the Web development activities:

- *Application Domain Engineering* deals with analysis, design, implementation and authoring of concepts which are related to the information content to be made accessible through the Web application, and functions to process, access and guide through;

- *Navigation Engineering* deals with activities related to analysis, design, implementation and testing of the modality through which users will navigate through the available information and services. Navigation engineering is concerned with grouping information fragments and functions into navigation nodes (hypertext nodes, contexts, views) and interconnecting them by means of links;

- *Presentation Engineering* is concerned with analysis, design, implementation and testing of the way in which information fragments, functions and their results appear to a user. The presentation model defines spatial layout and content of information fragments related to the user interface. It also defines presentation classes or objects, spatial relationships among them and content associated with them.

The above mentioned concerns are developed through all the workflows of the software process. The application domain is analyzed and structured according to one or several conceptual models. The conceptual models are refined to use cases and workflows (how to work with particular information from the application domain). A database can be designed and implemented to store the information described by the conceptual models or an API can be developed to access external information sources to be analyzed. Navigation over information is analyzed based on workflows and use cases connected with the information. An analysis model may serve as a tool to structure navigation requirements. The navigation is designed and implemented based on the available environment, e.g., ordinary Web pages accessed from a web server, or on the fly generated pages from a database. Presentation is analyzed, designed and implemented in a similar fashion based on requirements posed by information to be served by a Web application, users, navigation and the environment used to deliver particular information.

An application domain model is usually mapped into the content. If we look on the content from the variability point of view, several concepts from the application domain model can appear or describe different content fragments (information nodes). Moreover, the same content can be represented by different media and this content can evolve in time. The content can be also presented in different formats, e.g., as a book, lecture, or an article. Also overall access to the content can be managed through different patterns such as a digital library, an e-course (virtual university), on-line help, etc.

The Web application can be used by different types of users. Each user or user group may require different information to browse, a different composition of presented information (local navigation), and differing order and interconnections among information fragments (global navigation). Different navigation styles can also be determined by the target environment where the information is served to a user.

Similarly, different target groups may require information fragments to have differing appearance, layout, presentation and organization of the information to be viewed. The target environment can also restrict possible presentation variants. Thus it is important to capture this kind of variability as well.

The current Web application development methods conform to a single system development approach, i.e., the focus is put on the development of single systems rather than models for classes or families of systems where a certain variation among the members of a family can occur. Current methods can handle the requested variability in customized and/or adapted applications to a certain extent at the conceptual level as supported by known frameworks like structural techniques for UML, E-R models, or views where query language is used to query conceptual models for a particular subset of entities, components, use cases and so on.

Variability can also occur at the implementation level. It requires the existence of versions of a system's components and system releases. Version control in the hypermedia domain has been studied at the document level in several works [Nel, SRS00]. All mentioned works provide a model of versions and a model of configuration, which define how the versions contribute to the final configuration.

As the variability in adaptive Web application can occur at several levels, this aspect has to be studied from the point of view of what is actually required and contin-

uously refined from the anylysis and design stages through to implementation and maintenance. The variability should determine the development, deployment and run time of the adaptive Web applications. For this reason, the current Web application engineering methods should be extended by means of methods for variability engineering.

According to the above mentioned requirements, the following problems have been identified:

Problem 1 How to extend the techniques for modeling Web applications by means of commonality and variability modeling?

To be able to define explicitly variable and common features in different design views, we need to extend the modeling techniques used in Web engineering methods with additional modeling views and concepts used in the views. This increases the expressiveness of the modeling language and provides further information in the model. On the other hand, additional information in the model may raise its complexity. The extension should be separated into separate, possibly orthogonal models in order to be able to plug modules in or exclude them according to the requirements of a project.

Problem 2 How to model user centered personalization of Web based applications?

The static variability modeling considered in the previous problem would support just different customers of the Web application. Internal diversity in requirements of one customer may result in installation of extended set of system features and the decision about which feature will be presented to particular user should be given to the application runtime. Thus, there is a great need for a modeling technique which will be able to describe the dynamic decision based on user and/or system behavior.

Problem 3 Can the models used for systematic descriptions of the Web based applications be utilized in the implementation for some of the personalization decisions or designer support?

Additional information in respect of personalization and common and variable features of a system and/or domain where the Web-based application will be used may be beneficial for reasoning and decision making. The designer might be able to make use of existing software components described by the design models and search for them according to the information in those models. Web-based application may utilize the models according to rules implemented in the application.

Problem 4 Is it possible to support the extensions using industry standards or their extensions at the model and at the implementation level?

The success of the World Wide Web has been supported by standard protocols and formats. Current trends follow W3C standard recommendations such as XML, RDF, or OWL. Systems design practices adopted by industry are also widely supported by standards. The methodology would benefit if current technology standards either at the modeling level (UML) or at the implementation level (XML, RDF, OWL) can be supported.

1.2   Contributions

The main contributions of this thesis are fourfold. First, a domain engineering method for adaptive Web-based information intensive applications is introduced. The method uses additional feature models in information modeling which are based on the idea of an information product line and are inspired by domain engineering approaches for building software systems like generative programming [CE00] or product line practices [Wit96] where feature models play an important role. Our method is also based on the idea that an item of information is usually communicated to a user using several concepts. We use the UML collaboration diagrams to model such a design view where information features combine to fulfill a main information goal. Our approach is based on:

- Two views on information: application domain and environment;

- User domain view which determines which user features are considered for adaptation decisions;

- Feature models which model common and variable features of concepts and variation points in three views;

- Collaboration diagrams which refine and integrate feature models of both information views and constrain the collaboration messages with evaluation of features from the user's viewpoint.

The feature and collaboration models allow us to specify conditions in which features from two separate models can be used. The conditions determine which combinations of features provide us with meaningful information. Different configurations of features and roles represent possible Web-based applications which form an application family. Such an approach provides us with following advantages:

- The separation of the application domain and environment models allow us to reason about application domain concepts independently of the environment used to deliver application domain content;

- The separation allows for connecting the content to different environments when instantiating a particular application;

- Mandatory and optional features together with variation points in both models allow us to maintain information which reflects different successful implementations of Web-based application;

- Instance roles and their collaborations allow us to maintain information as to how domain features in information serving environments installed at a customer site are combined.

Secondly, the thesis introduces the UML state diagram based approach for modeling of personalized navigation guides where user features are considered and modeled in a connected user domain design class diagram. The state diagrams provide facilities to express variability which will be resolved at run time by means of transition branches, splits and joins, and guards used to specify constraints based on user

features. Side effect actions provide us with a means to to update user profiles dynamically.

Thirdly, we explain how the models (UML state diagram models for navigation and class diagrams for user modeling) can be utilized for generation of adaptive navigation sequences. The method utilizes the availability of XML based standards for storing UML models — XMI. This enables us to take full advantage of XML technology which is very popular amongst designers of Web based applications. The generator is studied in the context of two platforms: Web pages and the WebML platform.

Fourthly, a reasoning method is introduced which uses the domain, resource, navigation, and user models to generate adaptive annotations, navigation support, and queries for information in a distributed environment. The service oriented architecture, where those models are used, is also introduced based on Semantic Web description formats.

1.3   Thesis Structure

The thesis contains three parts. Part I is concerned with the design of Web-based applications. Chapter 2 discusses several prototypes of adaptive Web-based systems from the design and conceptual point of view. Chapter 3 surveys modeling techniques and methods used in present day Web engineering. Chapter 4 contrasts the state of the art in Web application development with the future demands which are emerging on the design and development of adaptive applications.

Part II discusses the domain engineering conceptual framework for adaptive Web based applications. Chapters 5 and 6 deal with aspects of the *problem 1* identified in this thesis. Chapter 5 discusses activities of the domain engineering framework for adaptive Web applications and relates it to generic domain engineering approaches. Chapter 6 covers domain analysis activity of the proposed framework, which consists of conceptual modeling and feature modeling. The extension of the UML, which is de facto industry standard for modeling software systemsm, is used for feature modeling. Chapter 6 concentrates on application, environments and user domains. At the end other domains relevant for Web applications are taken into consideration. We introduce the conceptual and the feature models first. The feature models reflect the common features and variabilities at design time which are required in Web-based application. Then we explain how to use them to model two aspects of Web-based applications — the application domain and the respective environment which are concerned with content and its (re-) presentation. Finally, the user domain view describes features which parameterize adaptation.

Chapter 7 discusses domain design activity in the proposed framework contributing to the *problem 2* identified in this thesis. First it defines domain design models. Then, it discusses the variability at run time. The decisions as to which of variable features to choose is made according to operational models and restriction rules based on user profiles. Two techniques are discussed in this: the state diagram approach and the collaboration diagrams. The chapter also discusses how the state diagram approach for adaptive navigation modeling can be conceptually integrated with other methods on the WebML example. Chapter 8 summarizes the contributions of the part II.

Part III discusses several experiments performed with the proposed conceptual framework. The whole part contributes to solving the *problems 3 and 4* identified in the thesis. Chapter 9 discusses how the UML state machines can be used to generate adaptive navigation guides in Web based application utilizing its XML representation and XSLT. Chapter 10 discusses the integration of adaptive navigation guides into model driven frameworks for Web applications based on the WebML [CFB00]. The generator again benefits from standards for Web technologies and helps designers to prototype adaptive Web applications. Chapter 11 discusses synergies between conceptual models used in the design of Web application and their Semantic Web representations. Moreover, the models transformed to the Semantic Web representation formats are directly utilized for personalization decisions performed by automatic reasoning. This chapter provides a comprehensive study for the most of the products of the proposed framework. Chapter 12 summarizes applications of the conceptual models.

Part IV ends with a summary and remarks on open issues still to be resolved.

Part I

Adaptive Web-Based Application Design

Chapter 2

Adaptive Web-Based Applications

The purpose of the research being undertaken in adaptive Web based systems is to find answers to questions relating to the heterogeneous needs of many different Web users. Applications and information which are provided on the Web according to the 'one size fits all' approach are not appropriate to such a heterogeneous environment. Several researchers have tried to address personalization issues in their research and have developed a number of adaptive Web-based applications.

Adaptive Web-based applications are an alternative to the traditional "one-size-fits-all" static approach in the development of applications [Bru01] and aim to leave some of the features of such applications at the design stage in the form of variables which are dependent on several criteria. Thus a distinctive feature of such an adaptive application is its ability to adapt itself to certain conditions. The most commonly used criteria to determine which features of an application will be used are user centered criteria. User-centered adaptive applications utilize user features to determine appropriate information presentation and navigation sequences for exploring a sufficiently complete set of information. They update a user model in accordance with user interaction and the information which he or she has provided.

Adaptive Web-based applications match conceptual descriptions of adaptive parts of the applications with related descriptions of a user. In some cases, there is an exact match between information description and user features but mostly some similarity measures are applied in addition. Based on the matching results, some further processing is applied, usually at the presentation level. Such processing is classified into several techniques.

The first group of techniques is intended to improve a user's local navigation and orientation in the currently presented page or fragment. For example, such adaptive systems can provide different text or media variants which serve information at different levels of detail to users with varying levels of knowledge or expertise in some field. They can switch between different media types according to differing user preferences or learning styles. They are able to hide or appropriately annotate certain parts of presented information items based on values of user features maintained by a system. These techniques are called *adaptive presentation techniques* [Bru01].

The subsequent group of techniques have the purpose of enhancing the user's global orientation in hyperspace. That is to say, they are designed to provide a user with support in exploring required information. This includes techniques such as

enabling, disabling, showing, hiding, annotating or removing links when it is appropriate and applying priorities to them according to different user features. The techniques also deal with generating appropriate information subsequent to the one currently being presented which then provides a further aid to guide a user. These techniques are called *adaptive navigation techniques* [Bru01].

2.1 User Adapted Web-Based Hypermedia

User adapted applications can be found for example in the domain of educational hypermedia where learning instructions and navigation are provided adaptively based on learning related features and the preferences of a user.
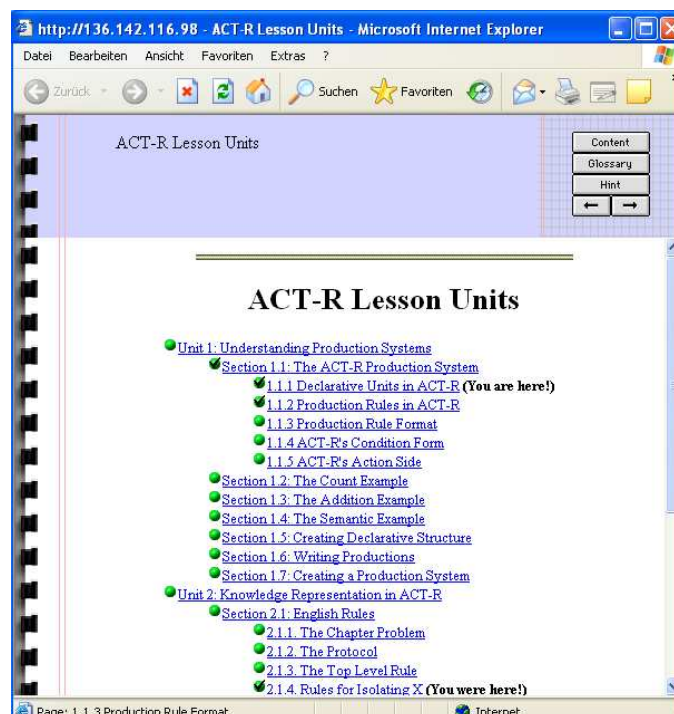
Figure 2.1: An adaptive annotation of links to lesson units in the Interbook system

Examples of user adapted Web based hypermedia systems in the educational domain are the Interbook [BES98] and the ELM-ART [WB01]. Both can adapt course presentation and navigation according to the user's background knowledge. The adaptation is made possible by formal representation of knowledge about the learning objects, users, and domains being taught. Learning objects are structured into smaller pieces, e.g., units, sections and chapters. Each can be indexed by concepts from an application domain model. The application domain model is a graph of related concepts linked by dependency relations. Each concept used for indexing can play a certain role in a learning object. In case of Interbook, it is either an outcome or a prerequisite. An outcome is usually a concept being presented to a user and a prerequisite is a concept required to understand the material being presented.

There are several guidance and personalization techniques which are supported in the Interbook. Figure 2.1 depicts a personalized recommendation of links to units of an ACT-R lesson in a form of outline. The traffic light metaphor is employed to cluster links which are recommended (green color), not recommended (red color), and other colors which inform about recommendation levels in between. The color is determined according to prerequisites required by particular units. The background is represented as a list of knowledge items needed in order to be able to grasp particular units. The background knowledge items required by a unit are compared with the user overlay model and if it matches fully then a green color is suggested. Other colors are suggested if there is something missing in the user background and if the color shown is red then there is no particular match. In addition, the user history within the course is maintained and visualized by means of checked items. An indication as to where a user is in his lesson is also supported.
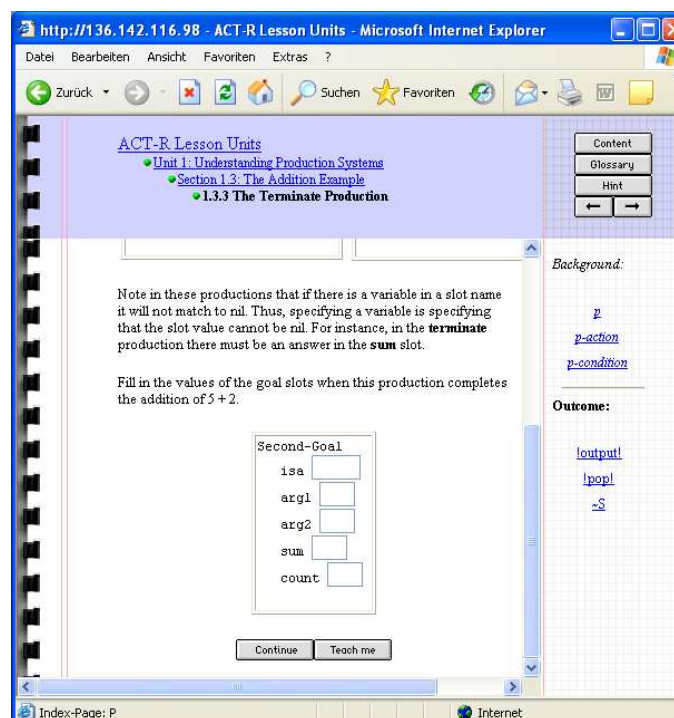
Figure 2.2: An adaptive presentation and annotation of content and links within a lesson unit in the Interbook system

Figure 2.2 depicts an example of Interbook unit presentation. Several guidance techniques are supported here. First of all, a user knows which background he should have and what the expected knowledge outcomes of the unit are by examining the list of knowledge concepts listed under those categories. The position of the user within the unit is highlighted as well. Similarly to the lesson outline, traffic light annotations are used to indicate parts of the unit which are recommended for reading or not. The system is able to recommend an assessment item to test user knowledge when he or she is at the end of the unit. Furthermore, the system is able to organize a learning path through a unit based on the structure of a course and a user's background

knowledge after pressing the "teach me" button. The user then follows the suggested path step by step. The path also contains the items which are missing in the user background if they are required by a certain section of a unit.

2.2 User Adapted eCommerce Systems

Personalization has also been implemented in industry applications using eCommerce. Systems like amazon recommend products, items and information related to customer interests and buying history, price preferences and availability.

In [AG00], knowledge about products or items, user preferences, user background knowledge about the products and presentation preferences are maintained according to prescribed structures. The product features are matched with features maintained about a user, user groups, and user stereotypes. Similarity measures are employed to obtain an order of product items. In addition, presentation styles are sometimes taken into account, for example to reduce complex information presented by applying natural language processing.

Record1:
Name: Facile;
Code: 100025;
Features:
Price: LIT. 108000
Color: grey, black;
. . .
Properties:
Quality: high;
Ease of use: high;
Cost: medium;
Design: high;
. . .

Figure 2.3: An example of a product record from a SETA system

An information record about products in the SETA [AG00] has a prescribed structure which is split into three main categories: basic information, features and properties. The basic information refers to the name and code of the product. The features of the product reflect the product's color, price, etc. The properties refer to qualitative characteristics like quality, ease of use, cost, design, etc. The properties can usually take on a stereotypical value selected from a set {high, medium, low }. An example of an information record about a product is depicted in fig. 2.3. The product record is about a product called Facile. The Facile is available in gray and black colors and for a price of 108000 LIT. The quality, ease of use, and design properties are set to high and the cost to medium. The SETA system also maintains information about users in precisely defined structures.

Figure 2.4 depicts an example of such a user record. The record includes an identification subrecord, personal data and user features such as characteristics, knowl-

Identification:
First Name: Paul;
Family Name: Smith;
Personal data:
Age: 55–64;
Gender: male;
Job: employee;
Education Level: high_school;
User features:
Characteristics:
Receptivity:
Values: low: 0.5; medium: 0.25; high: 0.5;
Sight:
Values: low: 0.3; medium: 0.4; high: 0.3;
…
Knowledge:
Expertise:
Values: low: 0.5; medium: 0.3; high: 0.2;
Interests:
Technical Interest:
Values: low: 0; medium: 0.4; high: 0.3;
Aesthetic Interest:
Values: low: 0.3; medium: 0.3; high: 0.4;
…
Preferences:
Quality:
Importance: 0.8;
Values: low: 0; medium: 0.4; high: 0.6;
Ease of Use:
Importance: 1;
Values: low: 0; medium: 0.25; high: 0.75;
Cost:
Importance: 0.7;
Values: low: 0.5; medium: 0.3; high: 0.2;
Design:
Importance: 1;
Values: low: 0; medium: 0.8; high: 0.2;
…
User classification:
life style:
Values: yuppie: 0.2; average: 0.6; modest: 0.2;
domain expertise:
Values: ...

Figure 2.4: An example of a user record from a SETA system

edge and interests. Characteristics in a user record are for example receptivity for recording user's ability to absorb large amount of information, and sight for recording user's ability to read small text. The user's knowledge is represented as a stereotypical expertise where a propensity for expertise is distributed over three linguistic values: low, medium or high . Other user properties such as technical and aesthetic interests, preferences such as quality, ease of use, cost and design, and user classification to stereotypes are measured similarly. The structures of user properties are extensible and can be defined by a store designer.

To overcome the "cold start problem", the SETA system maintains a stereotypical knowledge base. The knowledge base clusters user properties of homogeneous customer groups into hierarchically structured stereotypes. The stereotypes are described from two viewpoints: (a) the properties belonging to individuals and (b) predictions; i.e., the properties which describe user features and preferences which are

relevant to a particular stereotype. The population can be clustered into stereotypes according to multiple viewpoints. An example of a stereotype profile for a novice user is depicted in the fig. 2.5. The stereotypes are used to predict user preferences for users when they first enter the system. The users fill in an entry questionnaire. The information from the questionnaire is used to classify the new user to several stereotypes. His own user profile is updated with a prediction based on features and likelihoods merged from several stereotypes. The merging is calculated based on matching degrees of the user profile and the stereotypes.

NOVICE USER
Profile:
Age:
Values: less_than_24: 0.1; 25-34: 0.05; 35-44: 0.05;; 45-54: 0.15;
Job:
Values: employee: 0.3; retired: 0.3; student: 0.1; teacher: 0.05; ...;
…
Prediction Part:
User features:
Domain Expertise:
Values: low: 0.8; medium: 0.15; high: 0.05;
Technical Interest:
Values: low: 0.5; medium: 0.4; high: 0.1;
…
Preferences:
Ease of Use:
Importance: 1;
Values: low: 0; medium: 0.3; high: 0.7;
…

Figure 2.5: An example of a user stereotype from a SETA system

The recommendation process is based on how closely product features match user preferences. The matching degree is used to determine in which order particular products should be presented and how the information pertaining to the products should be arranged. The SETA system generates different kinds of pages to be shown to different users which contain information content specific to the respective user. SETA also provides and orders links to related products. The degree of matching between product features and user features is utilized for link ordering. User receptivity and user expertise contribute to decisions about how complex information should be presented, i.e. how many product features to present and in which way. Similarly to product information, information about the structure of pages is maintained as well. Product features (the technical features, functional features, aesthetic features and generic features) are evaluated based on user interests: each feature is characterized by its importance in the product taxonomy and importance in the user profile. This information is used to order information to be presented to a user. The user receptivity is used to evaluate how many information items can be presented at a user interface simultaneously and what should be generated as a next step in a dialog accessible via more links (steps in navigation). Linguistic descriptions are generated based on user expertise. Presentation layouts are maintained in different configurations and a specific one is offered based on matching with user profile features like life-style and color preferences.

2.3   User Adapted Search and Exploration Systems

Search system prototypes can also benefit from personalization. The most commonly used search prototypes concentrate on searching for documents which match to a user query and order them according to user preferences. An example of a personalized search system on the Web is the ifWeb [AT97] system. It is concerned with stored knowledge about documents. The knowledge about the documents is represented by interconnected concepts which the document is describing. The ifWeb also considers a knowledge about a user where a user preferences are modeled as a graph of concepts.

There are two personalization decisions performed by the ifWeb system. The first one is concerned with the recommendation of documents which best match preferences in a user profile. Graphs of documents and user preferences are compared. Similarity measures between user preferences and documents determine whether documents are recommended, not recommended or provided with additional annotations. The second personalization decision facilitates the list ordering of documents based on links between them and a degree of likelihood that a user will follow a particular link. The degree of likelihood is analyzed again by means of matching with a learner preference graph. Additional annotation is computed from the history of paths which lead to the document being analyzed. Threshold values are set to classify the documents into several classes like interesting, indifferent, not interesting, or a class to record that analysis was terminated from a particular document. The documents are organized in a hierarchy based on the links chosen as promising and with annotation based on document classification to enable a user to explore them.

Another example of a search and exploration system is the INTRIGUE [AGP$^+$03] system for personalized tourist guides. The system dynamically generates multilingual tourist attraction catalogues and recommends tours which suit the preferences of various user groups. As with the ifWeb system, the INTRIGUE personalization decisions are based on knowledge about tourist items and knowledge about users and their classification into groups. The recommendation is based on fuzzy evaluation functions for ranking the tourist attractions. The knowledge about the items is split into the global domain (meta-) knowledge about tourist attractions and concrete knowledge chunks about single tourist items. Each item is characterized by features. Each feature is represented in structures prescribed by a conceptual model and contains a name, type, domain and measure unit. The type refers to which class a feature belongs to. The type distinguishes between geographic, essential, basic features, specific characteristics and properties. The classification reflects which purpose the features in any particular category have, e.g., properties are used for recommendations, geographical features for selection criteria and so on. Similarly, the user group descriptions are maintained as features which represent preferences and values which reflect to what extent a group of users considers a particular feature to be important.

2.4   Requirements for Design Methodology

The applications mentioned above share several features from the design point of view. The adaptation is usually a decision for a particular information item or func-

tion based on knowledge about the items being recommended. The knowledge about items usually contains what particular information items or functions have in common and where they differ. The properties which differ from item to item determine the source for adaptation. The selection of an appropriate item is based on results from matching to user properties. Different users have different properties or different property values associated with them. The differences determine a selection of different information items or functions which best fit to particular user features according to a chosen selection strategy. As the user's behavior pattern evolves, recommended items may change. This is ensured by continuous updating and evolution of a user's profile based on his or her behavior as traced by the application. In this way, the user always receives up to date information items or functions matching the current state of his profile.

Analyses of the applications described above resulted in the following requirements for engineering methodology:

- *Support for common and variable features* — A designer should determine which application parts are common to all users and which are variable (variants suitable for different users). To be able to take decisions about the common and variable features, an appropriate modeling technique should be provided. The technique should support the notion of common and variable features and a notion of feature dependencies when a selection of one feature influences a selection of another one.

- *Support for several domain models* — A Web application usually serves information from several domains. The term "user model" stands for another domain considered especially in the context of adaptive Web-based applications. Also an environment for information delivery and arrangement of the environment sometimes differ. Separation of the features according to the domains to which they belong helps the designer to focus on features which are important for a particular domain.

- *Support for dynamic connectors between several domains* — The separation of several domains allows for better decision making about features in a particular domain. When designing particular (instance of) a Web application, the domain features have to be connected (configured) in a certain manner suited to the context of the (instantiated) Web application. Appropriate design technique which supports connectors and collaborations between domains should be provided. Such a technique will help to reason about connections between domains which might encourage their reuse. The connections can be further constrained where the constraints are to be evaluated at run time. The technique should also support the specification of composition of information features into meaningful items of information suitable for presentation.

- *Support for navigation design in connected domains* — the composed information fragments should be further linked to form possible navigation paths. The navigation paths can be further constrained where constraints are to be evaluated at run time. The constraints can be of different kinds but the focus is especially directed to constraints which evaluate user features.

Chapter 3

Web Application Development

We have identified several requirements for a methodology which should support the adaptive Web-based application engineering in section 2.4. In this section we will look at current proposals for Web-based application design available in research community and will analyze how they support the requirements.

It is widely accepted that models constitute important mechanism in the process of a system development. Models help us understand the system by omitting some details. The choice of *what* to model has a significant effect on understanding a problem and suggesting the solution. Significant influence on the solution has also the matter *when* particular model is created. According to answers on "what" and "when" questions, particular notation is chosen. A notation determines the level of formality of particular model. Notation is also determined by a decision about whether structural or behavioral model is developed.

The notation and technique determine how we can express certain features of a system. Regarding to our requirements for the engineering methodology for adaptive Web-based systems, it is important that a notation and technique provide a possibility to express the variable and common features in a domain information which will be provided within a Web application. Also the technique should provide a possibility to model several domains separately as the Web application is usually build on top of several information and/or serving domains. In addition, it should provide notions for expressing variability in connections between the different domains and in navigation in the information space determined by the domains.

The questions "when", "how", and "what" provide dimensions for analysis of existing techniques used in development of Web applications:

- *Phase dimension* follows the question "when". It reflects the fact that a modeling technique is applied in particular phase in development of a system (i.e., inception, elaboration, construction, or transition). Not all techniques can be applied in each phase;

- *Workflow dimension* follows the question "when" from the point of view of workflows in software engineering. It reflects the fact that a modeling technique is applied in particular workflow of development (i.e., requirements, analysis, design, or implementation). Not all techniques can be applied in each workflow;

Workflows

Notation/Technique

Test

Implementation   ...

Design   State Diagrams

Analysis   Activity graphs

Requirements   CRC Cards

Inception   Class Diagrams

Elaboration   Use Cases

Construction

Transition   Application Domain Navigation Presentation **Products**

Phases
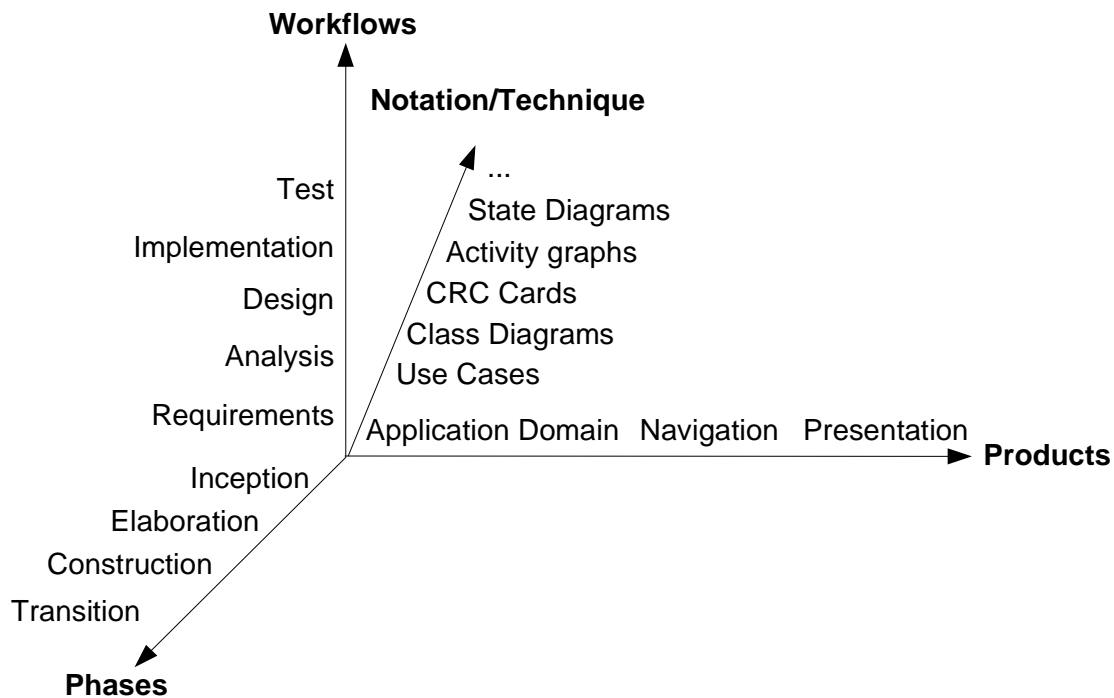
Figure 3.1: Web application development space with notation axis

- *Notation/Technique dimension* emphasizes modeling technique and corresponding notation used. The techniques can be grouped into two aspects: structural and behavioral. Structural models describe structure of a system (e.g., subsystems, packages) and structural relationships between them (e.g., inheritance, dependency and associations). Behavioral models describe reactions of a system to external or internal events such as collaborations or message passing in the system or the system's reactions to user interactions.

- *Web Application Products dimension* is concerned with the question "what" to model. Tiers, which are commonly discussed in the Web-based application perspective, are application domain, navigation and presentation. Not all techniques are useful for description of particular product;

Figure 3.1 depicts an extended space for Web application development from fig. 1.1 with the technique dimension. As pointed above, not all techniques are useful in each phase, workflow or for each product.

As discussed in chapter 1, the Web-based application development is characterized usually as an integrated set of activities producing three products of a Web application: *application domain models*, *navigation models*, and *presentation models*. In the following, the Products dimension is analyzed with respect to the modeling technique dimension. Several modeling techniques have been proposed to cover several views in requirements, design and implementation of software systems (see for example [Wie98, Myl98] for survey). Here we will concentrate on some of those which are used in the most prominent proposals for Web application development.

Mentioned perspectives or their categories can be further refined. For example, behavioral techniques can be divided according to semantics of a technique. There are three types of semantics: *axiomatic, denotational,* and *operational* [Goo94]. Axiomatic semantics map programs directly to properties, which characterize their behaviors. Denotational semantics map programs onto the functions, from which input-output behavior can be derived. Operational semantics allow that behavior can be derived from the sequence of transitions a program may perform. Formal models can be classified according to formal model or formal theory used as a base for modeling language. The basic categories are *set theory, logic, algebras, graphs* and *automatons.*

3.1 Application Domain Model

Application domain model comprises abstract concepts, which are provided as information in a Web-based system. Moreover, the application domain model could serve as a model for indexing content items. The content items can be considered as instances of particular concepts, or as sets of structural features of concepts describing them. Application domain model may also involve models of tasks, which may serve as a base for later navigation and predefined interaction between a user and the target system.

Application domain modeling as a component of application domain engineering is discussed in [CE00]. Simplified view of the application domain model is that it is a consistent set of requirements for the system in particular application domain[1]. This holds in Web-based application engineering too.

Table B provides a summary of notations and primitives used for application domain modeling.

Concepts' static structure of an application domain is mostly modeled by two approaches: (i) application of semiformal techniques such as class diagrams or entity-relationships diagrams, or (ii) application of graph formalism for semantic nets. Semiformal techniques are based on the notion concepts as classes or objects with their attributes. Concepts are interconnected by several types of relationships (association, subtyping or generalization/specialization and containment).

The Relationship Management Methodology (RMM) adopts entity-relationship diagrams (ERD) [IKK97]. Hypermedia Design Method (HDM) [GP93] employs customized ERD. Hypermedia application often involves unstructured information. HDM introduces entities, which derive its content from its components. Components can be for example sections of documents. Entities represent objects in the real word (exemplar perspective of a concept). Entities can be grouped into entity types. The distinction between entities in classical ERD and entity types in HDM is that HDM entity types have more complex inner structure formed by entities, their components and perspectives. Web modeling Language (WebML) [CFB00] is also based on ER model. The generators from WebML to Java Server Pages and Microsoft Active Server Pages templates have been implemented.

Methods such as Object-Oriented HDM (OOHDM) [SR98], UML-based Web Engi-

---

[1]Czarnecki refers to them as feature models.

Method/Model	Notation	Primitives
OOHDM	Class Diagram	Class, Association, Aggregation, Generalization/Specialization
SOHDM	CRC, Scenario Diagram	CRC Card, Relationship, Task, Flow
RMM	ERD	Entity, Relationship
HDM	Customized ERD	Entity, Component, Perspective, Collection, Relationship
W2000	Class Diagram, State Diagram	Class, Association, Aggregation, Generalization/Specialization, State, Transition
HDM-lite	Customized ERD	Entity, Component, Perspective, Collection, Relationship
UWE	Class Diagram	Class, Association, Aggregation, Generalization/Specialization
WebML	Customized ERD	Entity, Relationship
WSDM	Class diagram	Class, Association, Aggregation, Generalization/Specialization
WCML	Component Diagram	Component, Relationship, Aggregation, Inheritance

Table 3.1: Summary of techniques for application domain modeling.

neering (UWE) [HK00], W2000 [BGP01] (successor of HDM)[2] employ class diagrams.

Web Composition Markup Language (WCML) [GG99] is based on component oriented modeling. Main element in the WCML language is a component with properties and declarations of behavioral features. The components can inherit from other components or aggregate other components. Special property for the component content is introduced too. WCML model can be compiled into target environments [GSG00]. The mappings from OOHDM designs to WCML were introduced [SG00].

One of the first approaches for a hypertext application modeling – the NodeCards system – is graph based. NodeCards system developed in Xerox PARC [HMT87] allows to model application domain by so called NodeCards. The notion of file boxes was adopted by the NoteCards. The file box is intended to be used as a container for node cards. Each NodeCard in the system has to be associated to at least one file box. The file boxes can contain other file boxes. Information is embedded in the NodeCard. NodeCards together with file boxes form directed acyclic graph. Another graph-based model with domain semantics is proposed in [WR98]. The domain is modeled as a graph with associated domain labels for nodes and links. Besides the ordinatry nodes which model information items, there are also the grouping nodes which are used to model encapsulation of related information itme nodes. Nested Context Model (NCM) [SRS00] models structure of document as a tree.

Behavioral models of an application domain mostly express activities. They are represented by activity diagrams or flow charts. These models describe processes, activities or scenarios in particular domain. The navigation can be derived from these models, or they can serve as organizing framework for incorporation of Web-based application into an organization. Scenario-Based Object-Oriented Hypermedia De-

---

[2]In W2000 the hyperbase information design is counterpart of conceptual modeling.

sign Methodology (SOHDM) [LLY99] employs scenario diagrams (similar to activity graphs in the UML) for task and activity specification.

3.2   Navigation Model

Navigation models usually describe possible navigation paths and navigation support through information space determined by the application domain model. The navigation in Web-based application is considered at two levels: *local* and *global*. This categorization reflects the principles identified for example in [THH95, Kra97]. In both cases, the navigation support needs to build on a large local (currently presented information) and global (all information) coherence and a low cognitive overhead with additional tasks performed at the user interface when navigating large information spaces.

Two aspects are commonly employed in modeling the navigation. The former is based on structuring information into contexts, which represent coherent information chunks/nodes interconnected by meaningful links — static structure of navigation. The latter is based on operational semantics, i.e., to which navigation node the system will get when particular user or system generated event will raise a transition — behavioral aspect of navigation (interactions). Table 3.2 provides a summary of analyzed techniques for navigation modeling.

Method/Model	Notation	Primitives
OOHDM	Class Diagram	View, Navigation Class, Navigation Context, Link, Index, Guided Tour
SOHDM	Class Diagram, Graph	Object Oriented View, Access Structure Node, Link
RMM	M-Slices	Slices, Links
HDM	Graph	Node, Structural Link, Application Link, Perspective Link, Collection, Component
W2000	Traversals, Collections	Collection, Traversal
HDM-lite	Traversals, Collections	Collection, Traversal
UWE	Class Diagram	Navigation Class, Index, Guided Tour, Link
WebML	Compositions, Link models	Unit, Page, Site View, Link, Selector
WSDM	Component Diagram	Component, Link, Navigation Track
Trellis	Petri Net	Place, Transition
XHMBS/M	Extended State Diagram, Class Diagram	State, Transition, Navigation Class, Navigation Context, Link, Index, Guided Tour

Table 3.2: Summary of techniques for navigation modeling.

Structural techniques are used for navigation modeling in most of the methods for Web application engineering. The navigation in OOHDM [SR98] is specified by two different views: navigation class schema and navigation context schema. In the former, the object oriented view mechanism is employed [AB91]. The navigation classes are views over structural models of an application domain model. They also incorporate derived attributes and additional attributes relevant for navigation, which do not appear in the application domain model. Navigation context schema is intended to

group navigation classes into contexts and access structures such as indexes, guided tours and so on. Similar principle for specifying navigation is employed in UWE [HK00] and WebML [CFB00]. There is only a little difference in the notation and modeling language employed. WebML in addition introduces Web specific modeling elements. Object-oriented views are also used in SOHDM [LLY99]. The contexts are specified by access structure nodes, which are interconnected by links. RMM-based methodologies (see for example [IKK97, HBFV03]) use slices to denote the contexts. Slices group entities and their attributes from content model. In [HBFV03], the slices and relationships may be constrained with user features to provide adaptive navigation. HDM [GP93] provides perspectives for its components and entities. It means that the entity can have different contents in different contexts (perspectives).

Behavioral specification for navigation is based especially on the state-transition models and their equivalents, such as the petri nets. The notion of path is employed. Stotts and Furuta [SF89] proposed a model (called Trellis model) to specify so called *browsing semantics*. The binary petri nets proposed for such specification was later extended with colored petri nets to support collaboration specification [SF98] and context adaptation [NF00]. XHMBS (eXtended Hyperdocument Model B based on Statecharts) [PMdO99] have been proposed as a hypermedia related extension of the Harel's state machines. This approach integrates structural features of state machines (AND and OR composition of states) and behavioral features (events and transitions). The path can be reconstructed at several levels according to a composition tree of a state machine.

3.3   Presentation Model

Presentation model for a Web application describes visual characteristics of information presented by Web application, such as the layout configuration of information items presented and their appearance. Table 3.3 presents summary of mentioned techniques for presentation modeling.

Method/Model	Notation	Primitives
OOHDM	Abstract Data Views, ADVCharts	View, Object, State, Transition
SOHDM	User Interface design	UI objects (button, image, list...)
HDM	HMTL	slot, frame, generated HTML
W2000	HTML	Page Types, HTML elements
HDM-lite	HMTL	Page Types, HTML elements
UWE	Class diagram	Presentation class, Framesets
WebML	Pages, Style Sheets	Entity, Relationship
W3DT	Simplified HTML Metamodel	Pages, Diagrams, Forms, Layouts

Table 3.3: Summary of techniques for presentation modeling.

OOHDM [SR98] uses Advanced Data Views (ADV) for presentation modeling which allows to derive attributes from several navigation and/or domain classes. The configuration diagrams are used in [CHB92]. The attributes for expressing visual characteristics can extend the specification of such configuration objects. ADVCharts are used for expressing possible states of specific presentation objects (such as buttons) with definition of a message, which is sent to another object when the state is

changed. However, state-based modeling is used in OOHDM only for ADV behavior modeling. HDM-lite, as a successor HDM adopts structural modeling to specify the user interfaces on the Web. The presentation model is represented by three basic types: *component page- types, collection pages, traversal page-types* [FP00]. The UML class diagram is employed in [HK00] with stereotypes for presentation and navigation elements. The UML class is extended by frame sets as counterpart to HTML frame sets and presentation classes. The state charts are used only for specific objects similarly to OOHDM. WebML [CFB00] incorporates presentation modeling too. The models of presentation are represented by two types of style sheets: *untyped style sheets—models* and *typed style sheets*. The former are generic and independent from the content. The latter are intended for specific concepts (information items).

Traditional Dexter-based reference models such as Dexter hypertext model [HS94], Amsterdam hypermedia model [HBR94], and Adaptive hypermedia application model (AHAM) [BHW99], incorporate the runtime layer where presentation of a component is seen as an instance of the component. Two fundamental constructs are introduced—an instance and a session. The instance is used as a mechanism for instantiation and mapping components and links to anchors. The session is used to map the instance to a corresponding component. RMM [IKK97], HDM [GP93], or W2000 [BGP01] consider HTML as presentation modeling language.

The abstraction of HTML can be found in visual modeling language — World Wide Web Design Technique (W3DT) [BN96]. The approach is based on top-down principle of large scale Web site decomposition. The W3DT modeling language incorporates elements for modeling Web site, which may contain one or more diagrams. It means that the site is modeled by several diagrams (directory structure, linkage of pages, etc.). The diagrams are structured into pages, which can have optionally one or more layouts. Pages (form, index or menu) may be linked to other pages. The authors distinguish static and dynamic elements. Jim Conallen provides stereotypes for modeling Web application architectures using UML [Con99]. His approach is based on the architectural and programming elements supported by Web environment. Architecture model depends on whether the site is dynamic or static. Several stereotypes for relationships are provided such as link, target, builds, etc.

3.4   Dependencies between Models

The modeling techniques described above are used in the models, which are common for any Web-based application. We denote them as *core Web-based applications models*. Relationships between mentioned models (sometimes denoted as levels [RS00]) are very important. Models' relationships and possibilities of transformations were not studied extensively in case of Web-based modeling. High level dependencies between core models are depicted in Figure 3.2. We use the UML notation where dashed arrows represent dependencies and boxes represent packages. The arrows point from the dependent (source) package to another (target) package. Each package of one of the core model is split into two subpackages. These packages represent two aspects in each core model: structure and behavior.

The structural model of *concepts* expresses the dependencies between concepts. The concepts from application domain form the base of a Web-based application.
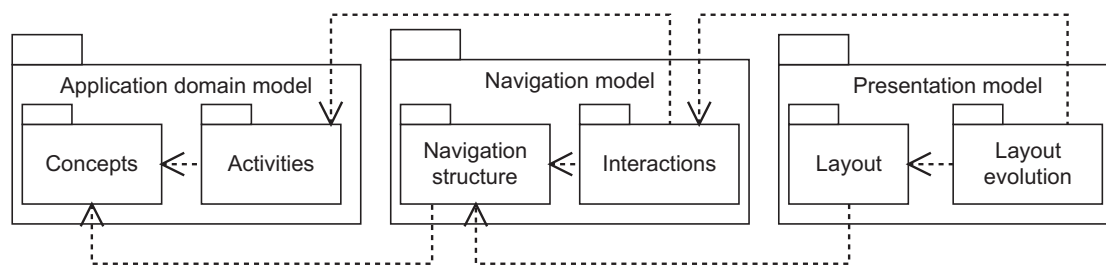
Figure 3.2: High-level dependencies between core Web-based application models.

Concepts index and describe the content of the Web-based application. The structural model of concepts serves besides modeling concepts per se also as an interface for accessing the information in the Web-based application.

The dependency between the model of *activities* and the structural model of concepts in an application domain means that the activities model references concepts, which describe the activities. The activities are linked usually by transitions. This allows to look at the application domain from different perspectives (the relationships in the model have different semantics).

The *navigation structure* model references and merges concepts from modeled in the conceptual model. The concepts are merged into navigation views, packages or higher level nodes. The navigation model also instantiates the relationships between concepts from the conceptual model. Higher level navigation nodes participate in different contexts according to specific conditions. The nodes and contexts are interconnected by relationships from conceptual model or by specific navigation shortcuts.

The model of *interactions* enriches the navigation structure model by events, which raise particular link (transition) between contexts and nodes. The nodes and contexts are transformed into superstates and substates. The interactions can be derived from higher level activities models. The navigation in the information space is performed according to the activities in the domain.

The navigation objects from the navigation structure model are mapped to several *layout* objects, which describe also the appearance of navigation objects. The symbolic alternatives of the presentation can be represented by several alternative substates of a user interface. The information (concept) presented in particular context in particular navigation view can change its layout and appearance. The *layout evolution* model models such a view. Layout evolution is performed according to the user interactions when he or she browses the information space of content provided by a Web application.

User modeling is very active research area especially in the field of adaptive Web-based systems. However, a little attention has been paid to the guidelines for modeling adaptive Web-based applications.

User models are very closed to the navigation models because the user model influences interactions within a system on the one hand while being updated dynamically according to the user interaction on the other hand.

The classes and associations are used for user modeling in [Koc98]. User model-

ing is based on the user model component of the Adaptive Hypermedia Application Model (AHAM) [BHW99]. Each user has a separate table in this model where a concept, the level of knowledge about the concept, whether it is read or whether the user is ready to read it are referenced. WebML [CFB00] introduces more specific modeling elements: a user and a group, which are directly applied in user access. User features are used to specify constraints on navigation guides in [HBFV03].

Bayesian networks are employed in [HN99]. The Bayesian network is used as an index to the conceptual network of content. Conditional probabilities determine to what extent a user is ready to read next concept presented in an information chunk. The probability is updated in accordance to what concept a user read. The user model evolution is modeled in [Koc98] by a state chart. The state of a user model can be changed according to an event generated from a user interface.

As pointed in chapter 1, *environment domain model* is another domain model useful for adaptive Web-applications to describe different environments which are used to deliver the content adaptively.
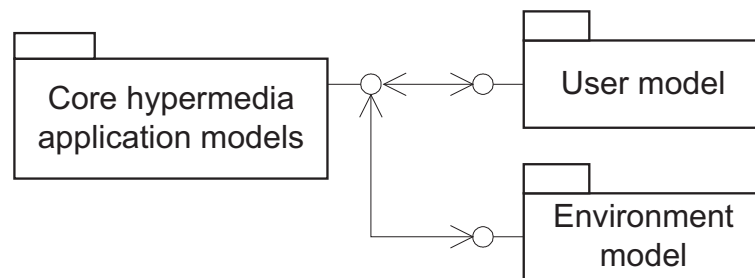
Figure 3.3: Extending core Web-based application models.

The possibility for extensions of core Web applications models can be seen in figure 3.3. The core Web-based application models can be connected to other models by means of conceptual elements in these models (concrete classes, their operations, events, transitions, etc.). The similar connection can be made in other models, which extend the core models (e.g., user model, and environment model). These conceptual models serve then as interfaces to the models. They can be referenced directly in the core models or can extend the models. The interfaces are represented by circle ended line (the UML notation for an interface). The bidirectional arrow between interfaces represents their usage by packages at both sides.

We can take the integration between the user model and the navigation model as an example. Interactions cause update of the user model. The user profile structure is updated according to the concepts visited during his browsing in a Web-based application. An update of the user profile structure is performed by methods and operations from the user profile structure model. The user profile can be in several states. The states and transitions between them form the profile evolution model (dependency between the user profile structure model and user profile evolution model).

Chapter 4

Summary

This section summarizes the reviewed techniques according to the notations used and the requirements posed on the engineering method for the adaptive Web applications.

4.1 Static Structure Models

Commonalities and Variabilities. The class diagram technique and the extended E-R models are the structural modeling techniques used usually to model several viewpoints on web applications. They are graphs where the nodes represent entities/classes of domain interest specified by their properties or attributes which are usually typed. Edges represent dependencies between entities which are usually of different kinds like association, generalization/ specialization, composition, aggregation, and so on. In web applications, they usually represent either information which is going to be served by a web application or an organization of a domain in an environment (e.g. courses and lectures in eLearning applications, books and chapters in eBooks, and pages, navigation and presentation classes in generic web applications).

The reviewed methods provide modeling techniques, which allow variability modeling partially at the conceptual level by means of generalization/specialization relationship or `xor` constraints between aggregation or association relationships. However, the specialization relationship is rather implementation oriented technique for variability modeling. Moreover, specialization explicitly prescribes that the particular concept instance may belong only to the one specialized sub concept. Similarly can be seen the `xor` constraint. The variability at the attribute level cannot be modeled by techniques employed in mentioned methods. Another technique where variability can be partially captured is views modeling.

Some methods borrowed the view mechanism known from databases. The views are mostly described by an object query language. Such a query language is used to select and constrain the concepts and attributes from the domain model to be presented at the navigation or the presentation level. The views are used usually to organize the domain information to chunks which can be presented at once in particular context. The context is either set as a presentation context or as a context given by associations among several concepts describing the information items. The variabil-

ity in views is considered sometimes at the navigation level where it is parameterize by attributes from a user profile. As the current methods do not consider several domains to be analyzed for the web application also the views are made just over the one domain model.

Multiple Domains and Connectors between them. Current methods for web application development distinguish between an information organization (environment) domain and a navigation domain. Some of them recognize a domain of a user to a certain extent. They suggest to connect the domains through a navigation domain model where the concepts from the information domain and the user domain models are used. This approach is very common but have several limitations:

- Strong coupling of the user parameters and the domain model with the navigation model makes it difficult to refactor the models for new customers;

- The fact that the constrains which are used to model adaptive decisions are specified just at the attribute level of navigation model makes it difficult for a designer to explore the adaptivity point of view in design and dependencies between the adaptive features;

- No explicit notion of common and variable parts is supported; therefore it is difficult for a designer to easily extract and reason about the variability which is embodied in one single model.

4.2 Behavior Models

Behavior models are considered to a limited extent as a supplement to navigation structure models in reviewed methods. Some methods replace traditional behavior models with their own extensions which are semantically equivalent to the existing behavioral techniques. While the event/ condition/ action paradigm of state-transition models and the notion of interaction employed in collaboration and sequence diagrams are naturally suited for user oriented run-time adaptation, they have not been employed by the reviewed methods. The behavior models are used to a limited extent for example to model evolution of user profiles or to model navigation in information chunks but without considering adaptation features and variability in the models.

Part II

Conceptual Modeling for User-Adapted Web Applications

Chapter 5

Domain Engineering and Adaptive Web Applications

The World Wide Web is becoming a more and more popular platform for providing applications to an ever wider spectrum of users. This development brings with it several challenges in terms of the handling requirements and preferences of the wider spectrum of users. The significance of personalization of the information provided by the application and navigation in the applications is of increasing importance as a means of responding to the needs of different users.

A Web application is an application which is accessible for users on the World Wide Web, usually through a Web browser. An adaptive Web application is one which adapts some of its features by taking knowledge about user and his requirements into account.

In this thesis, we view adaptation of Web applications from two perspectives:

- Adaptation by humans to the changed requirements of stakeholders;

- Dynamic system adaptation to the changed parameters of environment or context.

The main motivation for studying domain engineering principles as reported in the literature has been the reuse of modules in software development processes. The essential characteristic of domain engineering processes is "engineering for reuse", while that of application engineering processes is "engineering with reuse". Here, the emphasis is laid on the idea that components designed for reuse should be customizable for new software applications and extensible with new components developed as custom components for specific customers according to their specific requirements.

In our view, the engineering of adaptive (Web) applications should have this point in common — the application being ready for customization for different customers but still retaining some parts in common at all the installation sites. However, the customization idea should be taken beyond the static customization done by a development team. True adaptive Web applications, as reviewed in chapter 2, adapt to changed environments, user features and other parameters on the fly according to knowledge gathered from their "sensors". This situation and some specific features of Web application processes lead us to investigate how domain engineering principles

can be adapted to the new conditions which engineering of adaptive Web applications brings. In this chapter we define a domain engineering method for adaptive Web applications.

5.1   Domain Engineering vs. Application Engineering

Domain engineering concentrates on providing reusable solutions for families of systems. There are several definitions for domain engineering. According to [SEI00], domain engineering is an activity for building reusable components. For that activity to be successful, teams must understand common and variable features of the components used to build software applications. According to [BFK+99], domain engineering relies on the notion of the application domain to scope a reusable infrastructure. An application domain bounds all possible applications in that domain. Domain engineering relates closely to software product lines where it was adopted for software family engineering. According to [MA02], product line engineering supports systematic development of a set of similar software systems by understanding and controlling their common and distinguishing characteristics as inspired by concepts from the real-world domain of the software products which are used to tackle main reuse challenges. We have adopted a definition from [JGJ97, CE00].

Definition 1 (Domain Engineering) *According to [JGJ97], domain engineering is a systematic way of identifying a domain model, commonality and variability, potentially reusable assets and an architecture to enable their reuse.*

The idea behind this approach is that the reuse of components between applications occurs in one or more application domains. The components created during domain engineering activities are reused during a subsequent application system engineering phase. Several approaches to domain engineering for software systems have appeared, for example the Model Based System Engineering [Wit94] of SEI, which was later replaced by the Framework for Software Product Line Practice [Wit96]. Generative programming [CE00] also adopts domain engineering which supports a generative way of producing programs. Figure 5.1 summarizes software development based on domain engineering or product line software engineering.

We adopt a process referenced by [Wit94, CE00] where domain engineering consists of three main activities:

- *domain analysis* defines a set of reusable requirements for the systems in the domain;

- *domain design* establishes a common architecture for the systems in the domain;

- *domain implementation* implements reusable components, domain-specific languages, generators, and a reuse infrastructure.

There are several other approaches to software product line and domain engineering. Product Line Software Engineering (PuLSE) [BFK+99] characterizes processes in
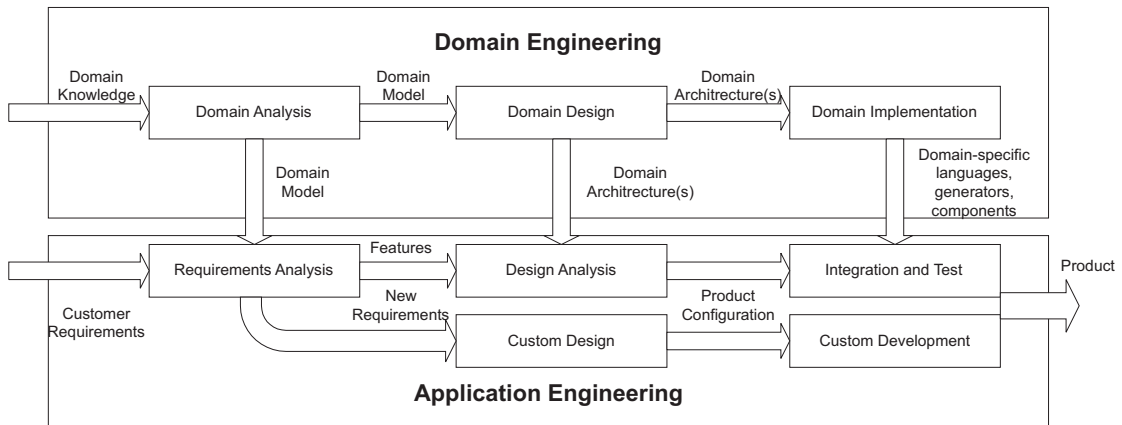
Figure 5.1: Software development based on domain engineering according to [Wit94].

four phases: initialization, infrastructure construction, infrastructure usage, evolution and management. The ooutcome of the initialization process is an enterprise baseline. Infrastructure construction defines, models and designs the product line infrastructure. Infrastructure usage implements the infrastructure to build product line members. Evolution and management focus on changes in the infrastructure. These activities are supported by technical components to operationalize the product line development such as customizing, scoping, modeling, architecting, and instantiating. Furthermore, PuLSE defines additional support components such as guidelines, project entry points, and maturity scales.

PuLSE is used and developed further in [MA02] where it provides for two processes: family engineering and application engineering. Family engineering results in domain models and product line infrastructure for building a software family in an application domain. Application engineering is a process which leads to instances of products in the family.

In [PBvdL05], domain analysis is described as "domain requirements engineering". In this way, the importance of requirements is stressed in respect of further development of a product line. The result of domain analysis is a domain model. This model represents common and variable properties of the systems in the domain and relationships between them. Domain analysis starts with a selection of the domain being analyzed. Concepts from the domain and their mutual relationships are analyzed and modeled. The domain concepts in the model may represent a domain vocabulary. Each concept is then extended by its common and variable features and the dependencies between them. This is the key concept of domain engineering. Variable features determine the configuration space for the systems family.

Domain design and domain implementation are closely related and are sometimes presented as one phase (e.g. product line infrastructure construction [BFK+99, MA02]). Domain realization is used to denote domain implementation in [PBvdL05].

The domain design produces generic abstract architecture for the family of systems according to commonly accepted architectural patterns (layered, model-view controller, etc.). Domain implementation implements the architecture by applying appropriate technology in specific environments. Sometimes, domain implementa-

tion is followed by domain testing in a process description (see e.g. [PBvdL05]). We see testing as an activity which cuts across all the activities of domain engineering; thus it is an integral part of the activities. We do not focus further attention on this aspect in this thesis.

In addition, several other domain engineering methods have been developed. This includes for example Organizational Domain Modeling [STA96], Domain Specific Software Architecture [TC92] and others. Please refer also to [PBvdL05] for more details on software product lines and domain engineering.

Definition 2 (Application Engineering) *According to [Wit94], application engineering develops software applications from software assets created by the domain engineering process.*

Similarly, in product line engineering, application engineering is a sub-process of software product line engineering in which the applications of the product line are built reusing domain artifacts and exploiting product line variability. Application engineering follows traditional approaches to single software application engineering with the requirements engineering, design and implementation, integration and test, but takes the further step of applying the product line infrastructure to select and instantiate reusable software assets.

5.2   Domain Engineering for Adaptive Web Applications

The basic principle of adaptation is to select appropriate variants of particular features or a combination of features (either by a human or a system) to satisfy user needs. Features which are adaptable are the ones which vary and thus to be considered as being the variable features of the application. The features which stay unadapted can be considered to be the common features of the application. This similarity to domain engineering methods lead us to study how they can be adapted to provide domain engineering guidelines for adaptive Web application families.

The domain engineering activities framework is general enough for any class of software applications. The main differences are to be found within the generic activities. There are three distinctive features of adaptive Web software applications which require more thorough examination of the domain engineering methods:

- The World Wide Web is document centric which means that any kind of application has to conform the to standardized HTML presentation styles when generating the user interface; This implies the existence of a specific solution domain which has to be adopted and considered in the engineering methods.

- The standardized browsers of WWW documents do not prescribe nor define any environment, displays or dialogs to be used in any application; This implies that for each application, an environment has to be defined. Therefore environment domain engineering should become an integral part of the engineering methods as well.

- Adaptation to a user requires that an adaptive application should be able to collect, maintain and work with knowledge about a user to be able to decide about variants provided within the application.

The first two features are related to all Web applications. The third one is related specifically to adaptive Web applications.

We propose a domain engineering framework for adaptive Web applications which adopts the above mentioned principles. The framework incorporates established Web based application modeling aspects into activities of domain engineering. Figure 5.2 depicts a framework for engineering adaptive Web-based information-intensive product lines [DB02b, DN04].

Domain analysis for Web-based applications involves *application, environment,* and *user domain conceptual* and *feature models* where:

- *Conceptual models* are used to model concepts and their mutual relationships in a particular domain and serve as vocabularies for later feature models and domain designs;

- *Feature models* are used to encode configuration knowledge; i.e., are used to maintain common and variable features of concepts and their dependencies such as a company's stored experience; While the configuration specification is user-regulated in application and environment domains, it is regulated by adaptation requirements in user domains.
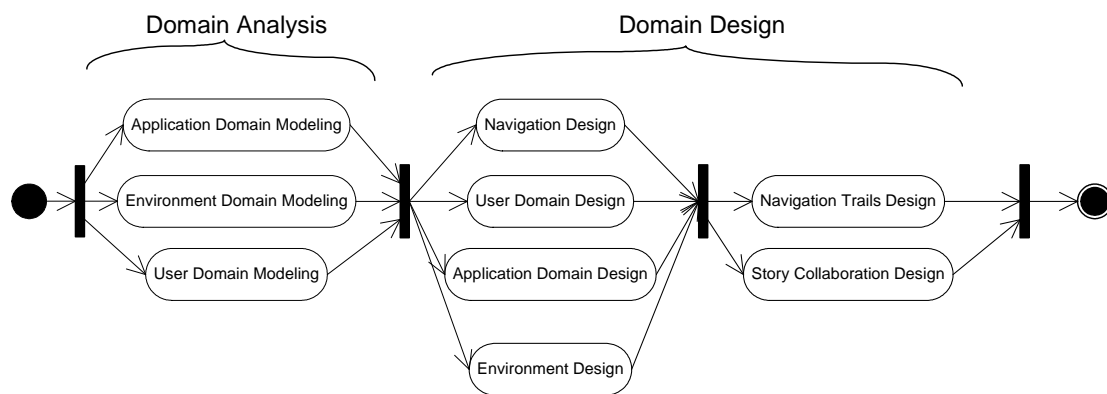
Figure 5.2: Domain engineering approach for adaptive Web-based application.

The purpose of the conceptual models is to document domain and environment vocabulary used in all other models. For example, the domain/content presented in a training suite in the case of a Java lecture implies that the domain conceptual model will refer to concepts from the Java programming language. As the lecture is accessible in a course environment, the course structure and some other concepts will be depicted in the environment conceptual model.

The purpose of the feature models is to document presentation relations of all concepts from the conceptual models to other concepts in that model in the case of the application domain and environment domain feature models; i.e., which other

concepts are used to articulate particular concept, e.g. on Java objects. The user domain feature models document configuration aspects according to the requirements of adaptation functionalities; i.e., which features and which combinations of features are required for certain adaptation strategies.

Domain design for Web-based applications involves *the navigation design, the user design, the application domain design,* and *the environment design.* The navigation domain design produces an architecture which enables the hypertext solution domain to generate HTML documents for particular environments with relevant content. The user domain design produces an architecture for user models to be used with applications in the domain. The application domain design produces an architecture to access content as an instance of application domain concepts and features. The architecture can be domain specific, i.e. based on vocabulary defined in an application domain conceptual model from domain analysis or it can be generic in order to access any content. The environment design defines an architecture for accessing and manipulating the environment.

Further refinements of domain design concentrate on content composition and navigation between content components. The domain design elements are used to bind the domains concerned together to produce reusable units of Web applications. The application domain and environment domains are bound together to create content and an environment *story collaboration models* as content components.

The domain design also incorporates mappings of the collaboration models to *navigation trails* which specify the sequences of content components to be presented to a user. The state diagram approach is adopted for this purpose. The access to content components and links between them is constrained by guards. They are further annotated by functions updating user profile, information presentation and access environment appearance. Further, the trails are bound to hypertext designs and presentation options to specify a concrete realization on the WWW hypertext platform.

Domain implementation includes construction of parameterized implementation components with their mutual dependencies. Parameterization of implementation components can be realized for example as HTML templates, active server templates, WAP templates or components in other implementation languages. Domain implementation should also incorporate domain specific languages such as query languages or languages for selecting and integrating components for a given application from the application family. All the mentioned models and their parameterization are transformed into these implementation components. We will demonstrate some aspects of the realization of components or services for adaptive Web applications in some case studies.

Application engineering is a subsequent activity following domain engineering activities when a particular contract for an application is made. The Web-based application is built according to the requirements which are specific to a particular application. Similar to software systems, the requirements are split into those which

- can be satisfied by the components from the application family framework created during the domain engineering process,

- should be satisfied by a custom development.

Results of the framework generation and custom development are integrated into the

final product. Some prototypes of adaptive applications built according to such principles will be shown in part III.

Chapter 6

Domain Analysis for Adaptive Web Applications

This chapter describes the domain analysis part of our method which is governed by conceptual and feature models. Those models are used to manage the common and variable features of the applications. The conceptual and feature modeling is explained by use of examples. Note that the examples used in this chapter are descriptive and not prescriptive, i.e., they merely illustrate the introduced modeling models for Web-based applications. Other teams may conclude with different models which are more suitable for their context.

Features which can be denoted as adaptable are the ones which vary and are thus regarded as the variable features of the application. The features which stay unadapted are common and are thus regarded as the common features of the application. The commonalities and variabilities of software families are the main concern in domain engineering methods.

The process of such information modeling can be summarized in the following steps [DN04]:

1. Define a conceptual model for an application, user and environment (e.g. concepts used to teach the Java programming language served in a course as an environment and concepts for a learner's learning performance for adaptation purposes);

2. Define a feature model for all concepts from the application, user and application domain conceptual models being used in the applications;

3. Update conceptual and feature models of domain and environment if new concepts and/or features have been developed.

6.1 Conceptual Modeling

Conceptual models represent abstracted knowledge of the real world. Entities of interests are mapped one to one to concepts in the conceptual models. Similarly, the relations of interest between the entities are mapped to associations (relations) in the conceptual models.

Conceptual models are used in many disciplines to provide an artificial reasoning framework for problem analysis and/or problem solving. Usually, they are used to represent knowledge about a certain area. In software engineering they are used to describe knowledge about software artifacts developed throughout a software process.

Conceptual models in Web applications are usually concerned with information or its representation being served including information chunks representation and software artifacts which help to support the access to information.

Information is usually comprised of one or more concepts from a domain where general conceptual models, taxonomies or ontologies may already exist. For example, a Java tutorial serves information which belongs to the domain of computer science. There are several taxonomies which are used for example to classify computer science literature (ACM CCS[1]) or to describe a body of computing knowledge and curricula [2]. Companies also use their own conceptual models to communicate terminology used in their information systems. Information delivery environments can also feature different concepts which are related to each other.

We employed the Unified Modeling Language (UML) to model the application domain and environment domain models. A concept is modeled by a class, which is stereotyped by the «`Concept`» stereotype. Concepts can be connected by one of the following relationship types: association, generalization/specialization or aggregation relationships in order to support the known abstractions of model domains.

Furthermore, some domains provide information which is more of a 'procedural knowledge' character. For those domains, we employed activity diagrams to model activity concepts with control flow relations between them.

Application Domain Modeling. We refer to an application domain as a domain of information (or content) which is to be served by a Web application. Conceptual modeling of an application domain models the domain in terms of concepts which are relevant to the content (are described by a content) served by a Web application.

Definition 3 (Application Domain Model) *An Application Domain Model is a set of structural and behavioral models. The structural domain models model concepts and relationships from a domain which depicts the content presented by a Web application. The behavioral models (usually activity models) represent procedural knowledge about the application domain. They model concepts which are activities from the application domain interconnected by control flow relationships. The control flows can be branched by decision symbols and the branches can be constrained by conditions.*

Figure 6.1 depicts an excerpt from such an *application domain conceptual model*, showing content of a page fragment on basic object-oriented programming (in JAVA). It is modeled by the UML class diagram with concepts annotated by the «`Concept`» stereotype and their mutual relationships. The figure expresses one possible view on relationships between `Object`, `Class`, object's `State` and `Behaviour`. `Methods` and `Variables` are used as additional concepts to describe those relationships, later realized by content fragments.

---

[1]http://www.acm.org/class/1998/
[2]http://www.computer.org/education/cc2001/
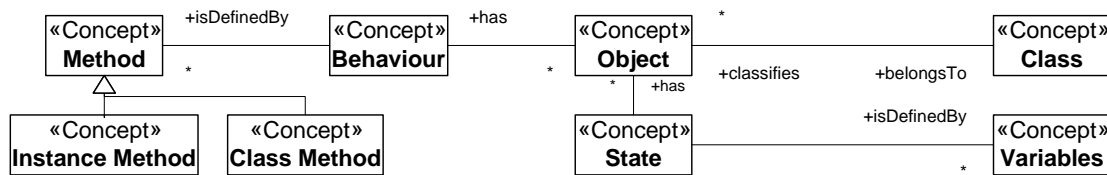
Figure 6.1: An excerpt from a conceptual application domain model which describes content on JAVA object oriented programming

The process of application domain modeling differs from application to application and from organization to organization. It may well be that the conceptual models are created according to content which already exists in some applications (having already been supplied by a particular organization). In this case, the models are created for the purpose of reuse and customization, to document what has been already developed. But on the other hand it may well be that an organization foresees that its applications will be partially reused and already uses models from the beginning of development process.

The following shows the main activities in application domain modeling:

- Collecting information sources on the application domain[3]

- Analyzing the information sources

- Extracting instances of concepts of interests from the information sources

- Classifying/categorizing the instances into concepts

- Creating relationships of interests between the concepts

- Refining the conceptual models

In some domains, information is rather of procedural or workflow character (provides procedural knowledge). In those cases, it is very useful to document the content and conceptual models by means of an activity model view.

Information/Environment Domain Modeling. We refer to an information or environment domain as a domain for representation and organization of information in the context of a delivery platform. An example of an environment is a course with its lectures or modules. Another example might be an e-Book with its chapters. In a customer support domain, an environment can be a problem ticket with its subproblems, activities, contracts and so on.

Definition 4 (Environment Domain Model) *An Environment Domain Model is a set of models with concepts interconnected by association, aggregation and generalization/specialization relationships. The concepts and the relationships are from a domain which is used to organize the content either for presentation, delivery or management purposes in a Web application.*

---

[3]this may involve the existing content or references to be used as a source for a new content
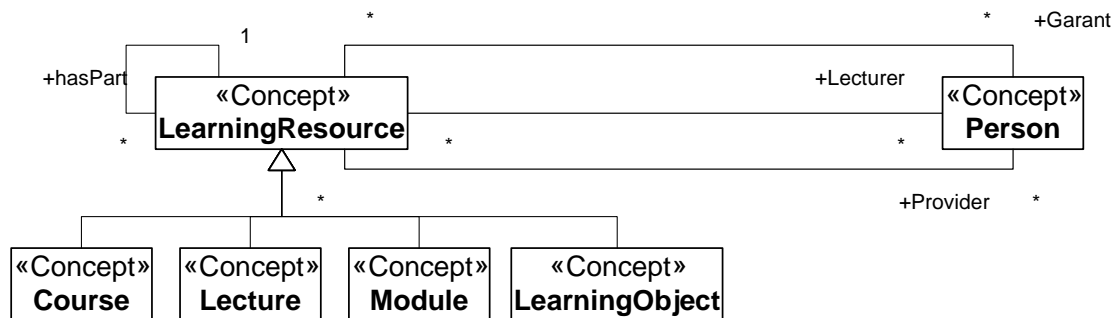
Figure 6.2: An excerpt from a conceptual environment/information model

An example of an environment conceptual model in a training suite is depicted in fig. 6.2. As the training suite can be provided with several possible virtual environments, a development company team needs to communicate how the environments are structured. An example of such an environment suitable for a Java tutorial can be a virtual course. Concepts such as `Course`, `Lectures`, `Modules`, `Learning Object`, `Lecturer`, and `Provider` would then appear in a similar UML class diagram for the *environment conceptual model.*

A process of information/environment modeling can consist of the following activities:

- Identifying existing and planned environments within a Web products portfolio

- Analyzing of organization patterns for such environments

- Identifying typical instances of concepts for such environments

- Classifying/categorizing the instances into concepts

- Creating relationships between the concepts

- Refining the environment conceptual models

User Domain Modeling.   User centered adaptation strongly depends on the process of user modeling. The user model attempts to capture and structure the various characteristics of a user. User modeling is understood from two different points of view in the literature. The first concerns a conceptual modeling view which is suitable for analysis and design of application. The second point of view is focussed on modeling by application. Such a user model is relevant at the instance level, being either a very simple stereotypic model or a more complex one, represented as interlinked objects describing the observed state of a user. This kind of model is dynamic (changes over time) and evolves according to the observations collected by sensors or the application itself and additional inferences applied on the collected facts. This model is a source of knowledge for decision making about which variant of a feature to choose when a particular user or user type is using the application.

In this section, we concentrate on the first point of view, i.e. guidelines for designing user modeling components for user-centered adaptation. As reviewed in chapter 2, there are several ways to represent such a user model. In some applications, a

probabilistic model is used to classify users under certain features. In other applications, a more object-oriented approach is used and the user is represented as a class with associations to further classes described by properties belonging to them.

A user model is created in a similar way to the application domain and environment model. The main characteristic of a user model then concerns the concepts and features which will be used as parameters for adaptation.

Definition 5 (User Domain Model) *A User Domain Model is a set of models with concepts interconnected by association, aggregation and generalization/specialization relationships. The concepts and the relationships are taken from a domain which characterizes the users, their behavior and features, knowledge and so on. The concepts are selected according to whether they are used in current Web applications or will be used in future Web applications to parameterize adaptation processes.*
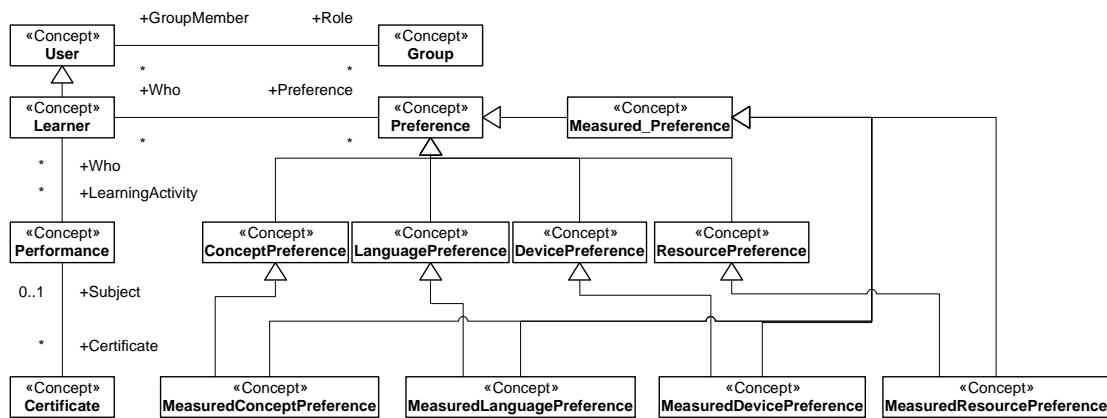
Figure 6.3: An excerpt from a user (learner) domain model specific to the eLearning domain.

An example of such a user model specific to eLearning domain is depicted in fig. 6.3. A `Learner` is a subclass of the generic `User`. The user usually belongs to one of several role groups. Learner's learning performance is represented by the `Performance` class. The learning performance is sometimes certified by `Certificates`. Each learner has own `Preference`-s. These can be specialized into specific subclasses. `ConceptPreference` refers to a learner's preference about a certain study concept. It can be further specialized to account for different learning style preferences in respect of different concepts. Similarly, `LanguagePreference` defines a user's preference for a particular language, `DevicePreference` records a user's preference for a particular device to be used for running a Web application or a device to be used in implementing additional external features connected to the application. `ResourcePreference` records a preference for a specific information resource which might also be used for recommendations. Further specialization is catered for via the `MeasuredPreference` class which accommodates metrics used to measure preferences of all kinds.

6.2   Feature Modeling

As mentioned in chapter 2, adaptation components in adaptive Web applications usually recommend one of the options for links, content fragments in a content composition or information items which are configurable in the Web application based on a user profile or the possibilities offered by a specific environment.

This means that there are parts of the content, environment and software components which are stable or common for any user or customer and parts which are variable depending on certain factors (mostly the values of user features).

To plan such an application, a designer should be able to think and reason about those common and variable parts. As pointed out in section 5.2, there is a similarity between domain engineering approaches concentrating on reuse and adaptive application engineering. In domain engineering, feature modeling takes a prominent role. The main reason for employing feature modeling in current domain engineering approaches is to handle variability in and dependencies between concept features of a system family resulting from different requirements of stakeholders using the applications from the family.

In adaptive applications, the customization has an even broader scope; i.e. in addition to the customization based on the explicit requirements of stakeholders there is still some variability which is left until runtime to be exploited according to an evolving user profile or other factors.

From the system point of view, variability has been studied in the context of the software configuration management community. In such cases, variability is handled at the systems components level by means of versions which which conttribute to different system releases. Version control in the document oriented hypermedia domain has been studied in several works [Nel, SRS00]. All the works mentioned provide a model of versions and a model of configuration, which defines how the versions contribute to the final configuration.

The variability considered in feature models has a broader sense when considered at the application level. At this level it is taken into consideration in several modeling aspects of Web applications and not just in the context of source code and changes to be made to the source code [DN04, DB02a].

Definition 6 (Variability) *Variability in product lines is defined as a measure of how members of a product family may differ from each other [WL99].*

Variability can occur in all the significant aspects (products) of the Web application engineering process; i.e., in the application domain, navigation and presentation.

In the application domain, different content can be used to communicate the same information to people with different backgrounds. Moreover, the same content can be represented by different media and this content can evolve in time. The content can also be presented in different environments using different media, e.g., as a book, lecture, or an article. Also overall access to the content can be managed through different patterns such as a digital library, an e-course (virtual university), on-line help etc.

Each user group may require different information fragment to browse, a different composition of the presented information (local navigation) and a different order and interconnections between information chunks (global navigation). Also, different navigation styles can be determined according to the target environment where the information is served to a user.

Similarly, it may be appropriate to supply different user types with specific display designs, layouts and organisation of the information to be read. The target environment can also restrict presentation possibilities. Thus it is important to take accout of this kind of variability as well.

There have been several proposals for techniques to model variability and commonality in software systems. Feature Oriented Domain Analysis (FODA) [KCHN90] employs a type of feature modeling where an AND/OR graph is used to denote variable features of domain concepts and variation points are used to depict dependencies between concepts. A Story Board is used to model variability in PuLSE [BFK$^+$99]. The story boards reflect the basic ideas behind the PuLSE methodology where incremental development of architecture is guided by scenarios. The story boards model the scenarios with alternative paths. Variation points and variants are considered as first class entities in associations between assets and components in [BGdPL$^+$03]. These concepts are used in requirements, architecture and implementation views as extensions for artifacts which are variable. There have been several extensions of UML for feature modeling (see e.g. [GFdA98]). In our work, we employ feature modeling and define a metamodel in UML which closely accords with the FODA version (Appendix A).

Definition 7 (Feature Model) *A feature model is a set of models which represent configuration aspects of concepts from domains analyzed in Web application engineering. Each feature model has one concept and its respective features. The concept and features are connected to each other by a composition relationship. Configuration relations between features and concept are represented as variation points. The concepts and features in feature models are mapped onto the concepts and relationships from the conceptual model.*

Definition 8 (Concept in Feature Model) *A* **concept** *in feature models represents:*

- **in an application domain model** — *an information item which is part of the main purpose (main information goal) of the content which an author had when he/she authored that content,*

- **in an environment/information model** — *a main structural unit of content in a particular Web-based application (different representations are modeled by different concepts),*

- **in a user domain model** — *a main concept governing an adaptation process (e.g. user preference for recommendation of items, or learner performance to recommend the next step in a learning path).*

A feature model has to be maintained for all concepts of a conceptual model which are going to be depicted as main information entities in a Web application environment.

Definition 9 (Feature) *A Feature is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems [Her98].*

Definition 10 (Feature in a Feature Model for Web Applications) *A* **Feature** *in feature models represents:*

- **in an application domain model** — *information fragments which are needed to communicate effectively a concept of a feature model,*

- **in an environment/information model** — *supporting structural units of content in a particular Web-based application,*

- **in a user domain model** — *qualitative and quantitative features which are needed for decisions about a certain adaptation strategy within an adaptation process (e.g. a competence acquired within learner performance to decide whether a user is able to grasp a particular content item or exercise or metrics of the performance for finer recommendations relating to the next learning steps).*

The fact that there are some features which are common to all configurations and that some vary is reflected by:

- *mandatory features* — form common or core features for all situations which are to be considered in our applications (application family), and

- *optional features* — form variable features needed only within a specific context.

Sometimes some features need to be presented together with other information features to provide sufficient explanatory material to enable the user to understand the presented information. Some other information features cannot be presented together because they could confuse a learner. In some cases, the combination of features is not so relevant. To distinguish between these cases, *variability relationships* have been introduced between features and they are usually denoted as *variation points* [Wit94] or *variations* [GFdA98].

Definition 11 *A variation point is a point/stage in design artifacts where a specific decision has been narrowed to several options but the option to be chosen for a particular system has been left open [Atk01].*

The variation point can define:

- *mutually exclusive variants (XOR),*

- *mutually required features (AND), and*

- *mutually inclusive features (OR).*

The semantics of the variation point types defined in the literature is usually the same, but the labels used to denote them sometimes differ. For example, [BGdPL$^+$03] defines several types of variation point as well but denotes them as *excludes* (XOR), *requires* (AND), and *ensures*. To achieve a consistent and unified framework for modeling, we maintain our feature models in UML. UML does not directly support feature

models, so it required us to provide a lightweight extension of the class diagrams of UML by using additional stereotypes (see Appendix A for the feature model profile). The «Concept» stereotype remains from the conceptual models. Features are annotated according to type by «MandatoryFeature» and «OptionalFeature» stereotype. A variation point is annotated by «VariationPoint» and its kind (XOR, AND, OR). The concepts, features and variation points are connected by directed edges which stand for composition. The direction indicates the parts of the composition. In some cases, the whole configuration of features (variation point with all the connected features) may be considered optional. In that case, the whole configuration can be excluded from the Web application. If no stereotype is mentioned, then it is considered to be a mandatory configuration and the rules implied by using such a variation point have to be observed.
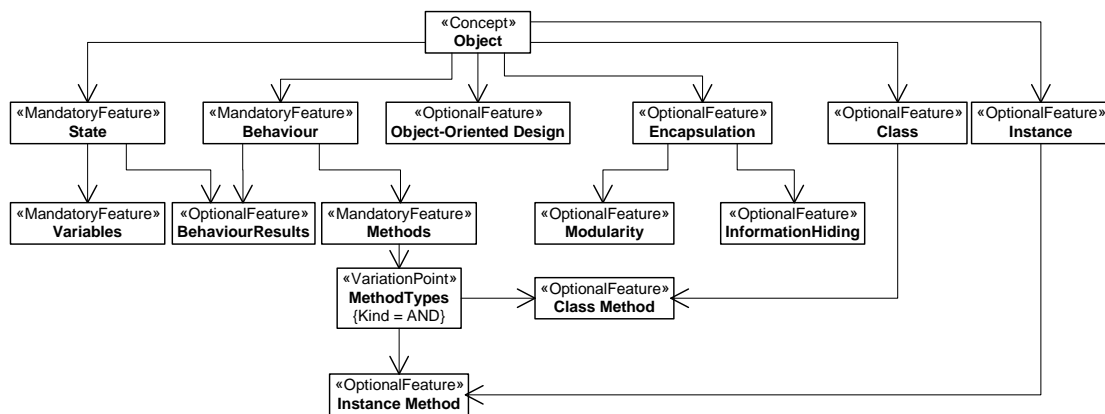
Figure 6.4: An excerpt of *Object* feature model

An example of a feature model is depicted in fig. 6.4. It is an excerpt of a feature model for the `Object` concept for Java lecture. The `Object` concept is usually described with the help of the concept of its `State` and `Behaviour`. Both appear as concepts in the conceptual model in fig. 6.1.

The `State` and `Behaviour` are considered to be mandatory features. The `Object-Oriented Design`, `Encapsulation`, `Class` and `Instance`) concepts are considered to be optional features; i.e. they do not have to appear in all applications.

Figure 6.4 depicts a variation point shown for the `Methods` mandatory feature. The model defines that the `Methods` also have to be described on the context of the `Instance Method` and `Class Method`.

All other concepts from the conceptual model (Fig. 6.1) usually have such feature models if they are communicated to learners as they become available in the application. Note also that the models depicted in our examples are not intended to determine the only possible solution, but just to exemplify how to create custom feature models which generate best practice guidance for information being served in Web based applications.

Similarly, a feature model is needed for the information/environment concepts. Figure 6.5 depicts an excerpt of such a feature model of a virtual environment for the `Course` concept. Usually, the information feature model for one virtual environment consists of just one feature model for the most general concept. The `Course` has
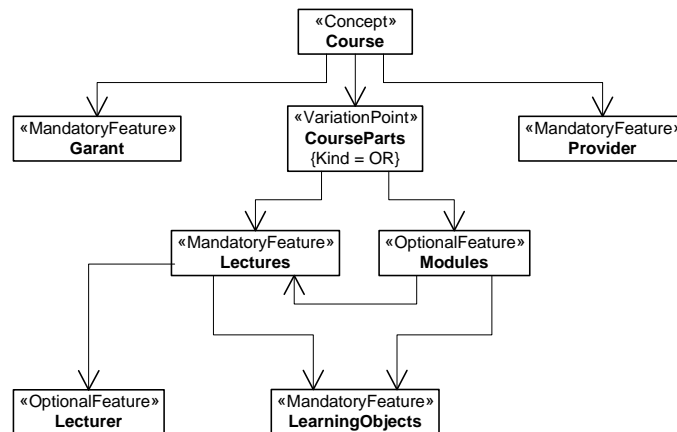
Figure 6.5: An excerpt from the *Course* feature model

to have a `Provider` and also a `Garant` (modeled by so called mandatory features). Then, if another customer stated such requirements, the `Course` can consist of either `Lectures`, where some of them can be encapsulated into thematic `Modules` or just from `Lectures` alone (this is reflected by the `OR` variation point of the `Course`). The `Lecture` can be provdided with a `Lecturer` who plays a role of a tutor when somebody needs further information or support related to the lecture. Both `Lectures` and `Modules` refer to learning objects.

A process of feature modeling can consist of the following activities:

- Identifying the concepts from the conceptual models which are of the main information entities to be communicated by Web application and mapping them on the concepts in feature models

- Identifying supporting concepts and relationships from conceptual models for each concept created for the feature models and mapping them to features in feature models

- Analyzing whether features are mandatory or optional in existing and planned applications

- Specifying mandatory and optional features according to the previous step

- Analyzing dependencies and composition relations between features of current and planned applications

- Specifying variation points and their kinds based on the results of previous steps

- Specifying composition relations between the identified concepts, features and variation points

- Refining the feature models

Similar to the domain models described in previous section, the user model is refined to the feature models level as well to provide explicit information on how it can be configured. Here, the variability dimension to be considered is based on which

adaptation strategies are involved in Web applications, which are mandatory and optional features, and what dependencies exist between them.
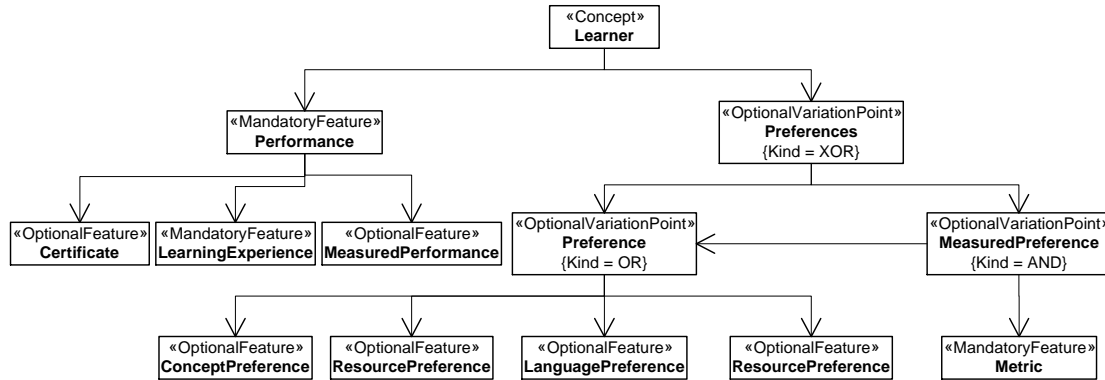
Figure 6.6: An excerpt from a user (learner) feature model specific to the eLearning domain.

An example of a feature model excerpt for a learner is depicted in fig. 6.6. The example indicates that adaptation of an eLearning application must at least be based on a learner's learning performance. The learning performance consists of a report on a learning experience, which is a mandatory feature. The experience can be certified and measured (two optional features). In addition, the adaptation can be complemented by a strategy based on learner preferences (`Preferences` optional `XOR` variation point). The preferences can be simply stated in a user profile (`Preference` optional `OR` variation point) or can be measured, for example by a degree of relevance for a learner. If the `Preference` variation point is chosen then one or more preference subtypes are chosen as well. On the other hand, when the `MeasuredPreference` `AND` variation point is chosen, both `Metric` feature and `Preference` variation point have to be selected.

The process of the user domain analysis/modeling is similar to the application domain or environment modeling. It is, however, worth mentioning that the user model can be useful just when it reflects real latent factors which might lead to the recommendations. Thus creating a user model for a particular class of applications is very challenging and usually involves at least some of research. The source of factors suitable for adaptation or, in other words, the source of factors which differentiate people from each other are sociological studies, demographic studies or psychological studies.

6.3 Conceptual and Feature Models in Web Application Solution Domains

The variability can also be considered at the presentation and navigation level. The navigation design defines groupings of information chunks into higher level contexts, which are interconnected by links. Links can be derived from semantical relationships between the concepts and/or features, or from the composition and variability relationships defined in the feature models. Presentation design has the purpose of

specifying presentation objects. These abstract presentation objects have appearance features and are also associated with spatial relationships.

Variability is expressed in a manner similar to that described for the examples from environment, user, and application domain models. Presentation and navigation objects can have their own features, which are variable or common with respect to the variability relationships between them. The features of the spatial relationship can also be considered as variation points.

Presentation and navigation objects are mapped to a subset of the concepts and features from information and application domain models. Mapping is performed at a design time or at bind time according to the specified rules. The mapping at design time is performed by choosing a particular set of the presentation features and its mapping to navigation or presentation objects. The mapping at bind time is performed by choosing a set of variable presentation features with association rules which are applied when a particular feature is current.

Furthermore, variability can be considered in the context of target run-time environments and technology employed or models used in design. For example, [ABGK02] considers variation, apart from the structural model view, also in activity and asset views for business processes. In [BGdPL$^+$03] choices are considered at the implementation level as well. Similarly, variability can be considered for Web technologies and environments for alternative decisions in JAVA, Java Beans, .NET or JSP.

Chapter 7

Domain Design

This chapter deals with the second aspect of customization: dynamic runtime adaptation design. In general, the driving force behind this may be multidimensional. So far we have concentrated on user centered adaptation. User centered adaptation can be also referred to as personalization; the application behaves according to items of knowledge about a person and adapts itself to them.

This chapter will provide a description of domain design techniques for application domains, environment domains, and user domains under consideration for adaptive web applications. It will discuss *dynamic content composition* using dynamic connectors between the application domain and environment domain models in terms of stories and environment access roles modeled by *collaboration diagrams*. Furthermore, it will discuss *adaptive navigation* modeling. *State transition diagrams* are employed for that purpose and are connected to the collaboration roles and user models for the purpose of accessing information content and user features for decisions about variants to be presented/followed at run time.

The feature models of the application domain and the environment domain have to be refined into the domain design. There are many ways of doing this. One way to go is to consider a domain design according to the methods reviewed in chapter 3. In our work, we follow the object oriented principles and the guidelines for using UML to expand domain analysis to domain design and further to application design.

Domain design also is also dependent on the situation and plans for the various domains we have described. For example, if for each feature a company plans to have just one content fragment it is probably vital to implement access through environment classes and to have just one class handling abstraction of a given content. On the other hand if there is a big pool of content objects implementing particular features from an application domain, it is probably necessary to make classes for all the features described.

Definition 12 (Web Application Family Domain Design) *Web Application Family Domain Design is concerned with design for particular domains considered in the context of domain analysis for adaptive web applications (environment, application, navigation) and their integration. Domain design represents a refinement of the domain analysis feature models and is adapted to the architecture and implementation technologies used within the web application family.*

There are several implementation technologies available for building web applications, such as like Microsoft Active Server Pages, Microsoft .NET, Sun Java Server Pages, PHP, J2EE and Java Beans. The architecture can be considered as a domain-view-controller, three-tiered architecture with business application servers and so on. All of them bring specific characteristics which may influence the domain design for the application family.

7.1   Application and Environment Domain Design
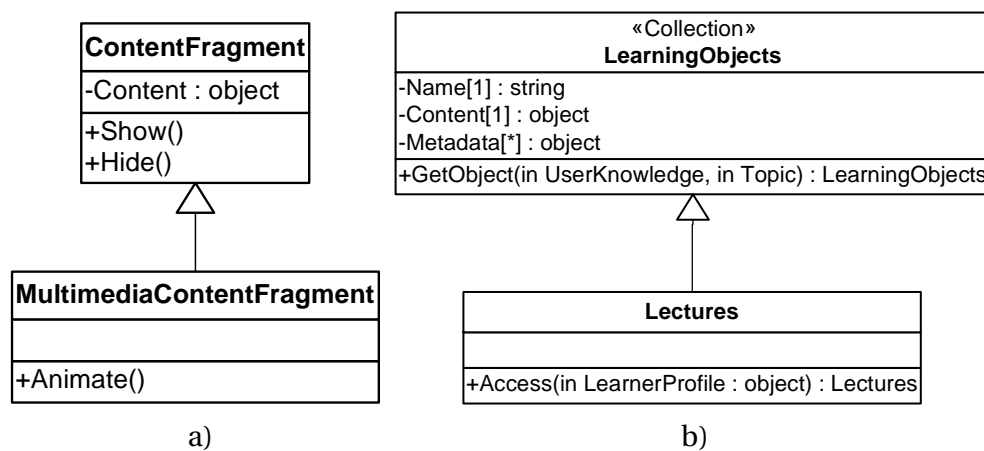
Figure 7.1: An excerpt from a domain design models (a/ application domain b/ environment domain) for an adaptive content management system

Figure 7.1 depicts an excerpt from an application domain and environment domain design model for very simple access to the content and environment of an application. The environment domain design more or less follows the concepts from environment domain analysis. It distinguishes the generic learning object interface and specific lecture interface. However, the application domain design model describes only the management interface to the content and access to particular content fragments is realized through an indexing mechanism using feature models. The feature models serve as metadata and can be represented in an appropriate format, e.g., XML or RDF metadata. In case there are multiple content fragments of the same features, features will be transformed into classes in design models. In addition, the same interface will be realized to access them, as depicted in figure 7.1a).

7.2   User Domain Design

To allow an application perform a user modeling task and for run time adaptation, the user feature models have to be mapped onto one of the implementation technologies.

One possibility is to map a user feature model onto a class diagram. There are two steps involved in creating an application user model:

- Create user model implementation classes and user model management environment specifications for all the features and concepts which are to be or may need to be delivered to your customers

- Select those classes and user management environments which are particularly relevant for your current application

In case of more complex user models, the classes can be packaged according to subdomains. For example the user preference management can be encapsulated into a preference modeling package while the learner performance management can be included in a different package.
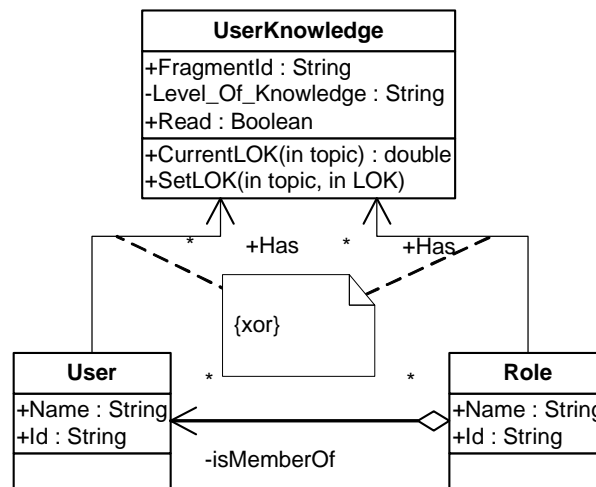
Figure 7.2: A user model for a simple e-lecture.

Clearly, there are many possible ways to map user model features and concepts onto implementation classes and relationships between them. A simple example which reflects a learner's learning performance in terms of the level of knowledge gained is depicted in fig. 7.2. The `UserKnowledge` class directly maps onto the performance feature. The `User` class maps onto the user concept and the `Role` class maps onto the role feature which is enclosed in a bigger user feature model (not depicted in fig. 6.6). The `Role` class is intended to represent a user group. When a user is a member of the role group his or her own level of knowledge cannot be associated with his person because he/she is then assigned the level associated with the role (`{xor}`). The user model also contains access operations for reading the current state of user characteristics and for updating user characteristics.

Another example follows, in contrast, the conceptual model depicted in fig. 6.3. It expands the concepts to include those additional attributes and access functions needed to maintain quantitative user features. An excerpt from the performance model concept is depicted in fig. 7.2. More on this topic will be described later in case studies. The framework implementing the user model for all the concepts is described in [DS05]. The user model is composed of the classes `User` and `Performance`, plus an association expressing that a learner can have several performance records based on the acquired `LearningExperience` and `Competence`. The `Performance` class stores the user's level of knowledge about the concepts described by the tutorial.
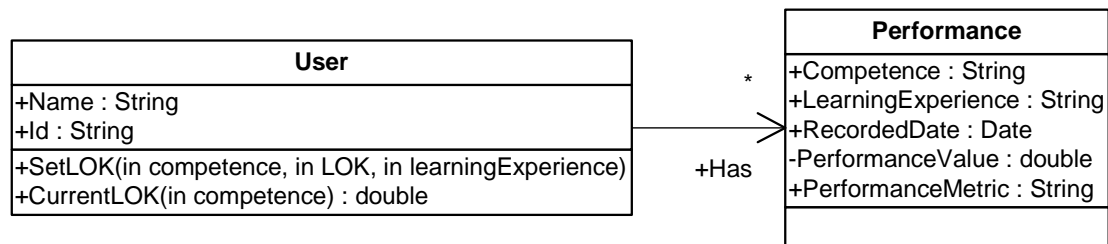
Figure 7.3: An excerpt from a learner (user) model for learning performance.

If an application follows the probabilistic approach instead and classifies a user according to particular domain features, the user model has to be extended by means of orthogonal application domain models which will provide domain features for user classification. The simpler case is when the same application domain is used for user features (this is the case for example for knowledge items in eLearning applications). However, if the domain of user classification differs — with the emphasis laid on buying stereotypes or personality stereotypes (e.g. collaborative person) — then such a model has to be created. Another application domain conceptual model is needed for such a model. The model is then mapped onto the feature model containing configuration knowledge. Furthermore, each feature has to be given a parameter which will be used for quantifying the likelihood of a user is characterized by such a feature.

7.3   Navigation Domain Design

The web is accessible mainly through document centric interfaces where information is provided in the form of hypertext, which is not programmable in any programming language but rather encoded in HTML. For this reason many of the design languages concentrating on hypertext or navigation design can be regarded as domain specific languages for the Web. The respective navigation or hypertext design can be seen as a solution domain for Web applications.

Navigation domain design should be performed with closer attention to the hypertext paradigm; i.e. to allow the generation of suitable business logic objects leading to an appropriate hypertext user interface. As reviewed in chapter 3, there are several methods for achieving hypertext or navigation design ranging from instances of concepts designed for web pages to access structures such as menus, guided tours and indexes. Here we show just one of many possibilities which is based on a metamodel of WebML [CFB+02]. WebML is a design language for data-intensive web sites.

Figure 7.4 depicts an excerpt from the WebML metamodel for hypertext partly described in UML. The `WebMLPage`-s are grouped under the `WebMLSiteView`-s and `WebMLArea`-s. The site views are used to specify different hypertexts for different user groups. The areas are used to conceptual grouping of pages which are related to particular topics. The `WebMLPage` is a container for a specific content. The content is generated according to `WebMLUnit`-s. Each unit is associated with a database entity which models the data storage. There are several kinds of units, each of which has different presentation options: `DataUnit` models a content fragment which presents
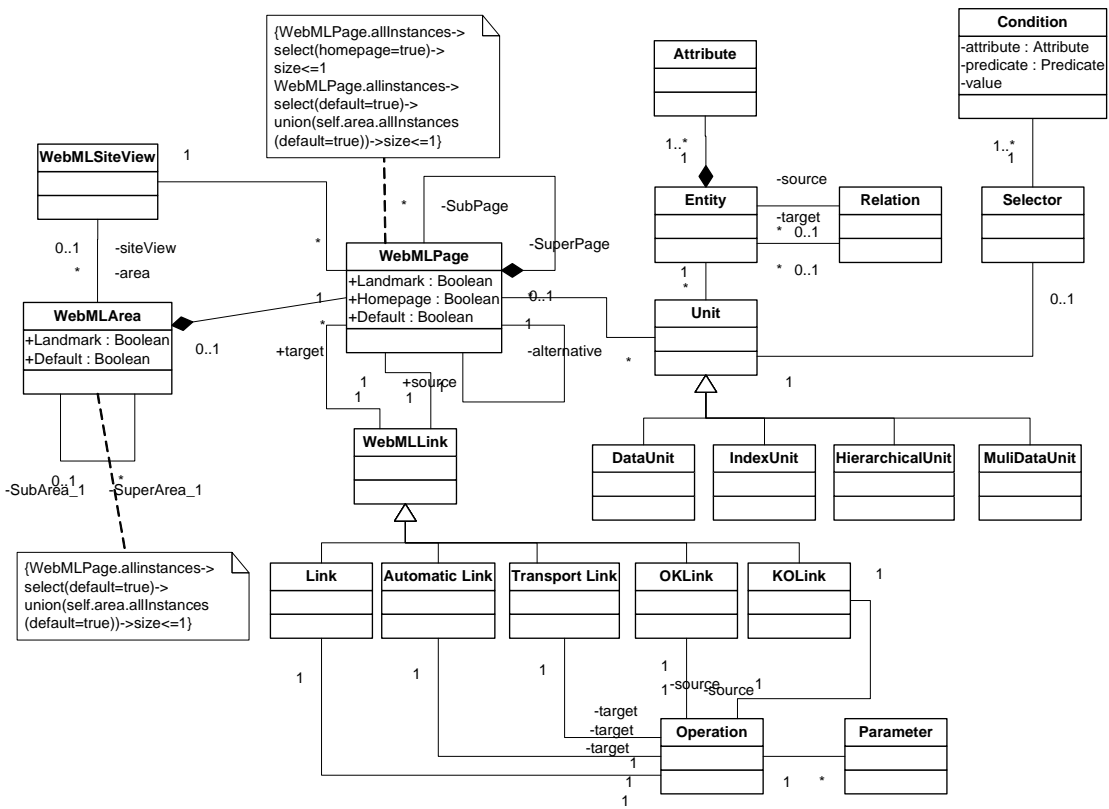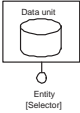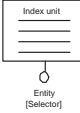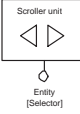
Figure 7.4: An excerpt from a metamodel of the WebML hypertext schema

exactly one instance of a database entity, `IndexUnit` models a list of entity instances, `HiearchicalUnit` models a nested index, and `MultiDataUnit` models a fragment of multiple instances of a database entity.

Table 7.1 summarizes visual syntax for some of the WebML concepts. WebML Pages and Units can be connected by links. Pages are connected by non-contextual links. Units are connected by contextual links which transport parameters (usually those of a clicked instance of a database entity generated by a unit). The context parameters are then used to select an appropriate instance from a database entity which is assigned to a unit where a user is about to arrive.

Table 7.1: Some basic WebML content units. The whole set of units is described in [CFB^{+}02].

Unit name	Visual Notation	Description
Data unit	Data unit / Entity [Selector]	Displays a set of attributes for a single entity instance.
Multidata unit	Multidata unit / Entity [Selector]	Displays a set of instances for a given entity.
Index unit	Index unit / Entity [Selector]	Displays a list of properties, also called *descriptive keys*, of a given set of entity instances.
Hierarchical Index unit	HierarchicalIndex / Entity1 [Selector1] NEST Entity2 [Selector2]	A variant of the index unit, which displays a list of properties of instances selected from multiple entities, nested in a multi-level tree.
Scroller unit	Scroller unit ◁ ▷ / Entity [Selector]	Represents a scrolling mechanism, based on a block factor, for the elements in a set of instances.

Besides classic links, there are special types of links in WebML: `AutomaticLink`, and `TransportLink`. The automatic link is used when two connected units have to display data right after accessing the page without further interaction by a user. The transport link is a link which just passes on contextual parameters but is not displayed within the anchor for interaction. This is especially useful when some actions are invoked which have to collect data and parameters from several units.

Similarly `KOLink` and `OKLink` are used with operations as automatic links after success or failure of the operation. `Operation` models particular actions a user can invoke by a button or a link at a user interface. There are several elements provided for several kinds of operations such as delete, connect data, insert and so on.

7.4 Variability at Runtime: Personalization

Personalization in the sense of resolving variability at runtime is concerned with two main areas of adaptive functionalities:

- *content adaptation* — variants of content fragments, their presentations and combinations are examined and selected according to knowledge about user features,

- *navigation adaptation* — variants associated with navigation directions in trails, the link annotations and their combinations are examined and selected according to conditions based on knowledge about user features.

The content adaptation and composition is modeled in the form of content stories as collaboration diagrams. The navigation trails are constructed as state diagrams connecting the stories into navigation guides.

7.4.1 Active Information Objects and their Collaborations

In chapter 5 we have described two domains which are used in web applications: the application domain and the environment domain. We have provided guidelines to make conceptual and feature models for both domains separately. The reason behind this is provided by the following particular advantage: the content can be plugged into any environment which we maintain as appropriate and vise versa.

Similarly, at the end of the previous section we pointed out that, especially with probabilistic approaches, we need two domains as well. However, for a concrete application or set of applications the connections have to be designed.

We introduce the idea that the application domain features "perform" a story in a particular environment. They collaborate between each other to communicate certain information with the help of the environment. Developing this concept of collaboration [DN04], we employ the collaboration diagrams to be found in UML.

Definition 13 (Story Collaboration Model) *A Story Collaboration Model is a set of collaboration diagrams which define dynamic content chunks accessible in a particular environment. The story collaboration diagrams contain collaborations between roles created as instance roles of features and concepts from an application domain feature model linked to instance roles of features and concepts from an environment feature model.*

In the case of user models, the user domain feature model plays the role of an environment feature model if such collaboration is needed.

At runtime, the feature instances collaborate to create a content. The idea of collaboration is based on the notion of active learning objects which provide a defined interface to access their content and presentation. With this idea in mind, dependencies from the domain feature models can be transformed into collaboration links.

Roles are used to model different purposes of a particular feature or concept in an environment/ information component. We use the following notation:

$$O/R : P :: C$$

where O is a Classifier or Feature instance,

R is a Classifier or Feature role,

P (optional) is a Package name to which Feature or Classifier belongs, and

C is a Classifier or a Feature.

Roles terminology can form complex structures. The UML class diagram can be employed to model such a structure [ADN+03]. This model can be used in a similar way to the domain and environment/ information conceptual models; i.e. as a means for communicating the roles terminology to be used in the web-based training suite.

The collaboration models are created as refinements of the feature models. The refinement consists of:

- Instantiating concepts and features from a feature model;

- Identifying roles of the instances in collaborations;

- Transforming associations between features and concept in feature models into collaboration links.

In the following we will explain the process of story modeling with the help of the examples used in domain and feature models in the previous section.

Domain Features Collaborations

The first refinement of the feature models involves refinement into the domain features collaborations. The refinement consists of defining concept and feature *instances and roles* and *links* between them as instances of associations between the concept and the features.

Instances and Roles. Figure 7.5 depicts an excerpt from a collaboration model of features from the `Object` feature model (the feature model depicted in fig. 6.4). Roles of domain features used in the collaborations are *definition, example, exercise, description* and so on. The `Definition` and `Example` roles played by the `Object` concept instances are used to model a situation where a term `JavaObject` is defined and then shown using the example. The term `JavaObject` is defined using the `JavaObjectState` and `JavaObjectBehaviour` definitions. The `Variable` definition is used to define the state variables.

The `JavaObjectBehaviour` definition collaborates with the `JavaMethod` definition at the moment when the behavior of an object is exposed by its methods. The `JavaInstanceMethod` and `JavaClassMethod` descriptions are provided as two alternatives for the Java methods (they play a descriptive role in collaboration with the `JavaMethod` definition). In addition, a comparison of the class method and instance method is provided to help clarify the difference between these kinds of methods. This is reflected by the additional comparison role of the `JavaClassMethod` in collaboration with the `JavaInstanceMethod`.
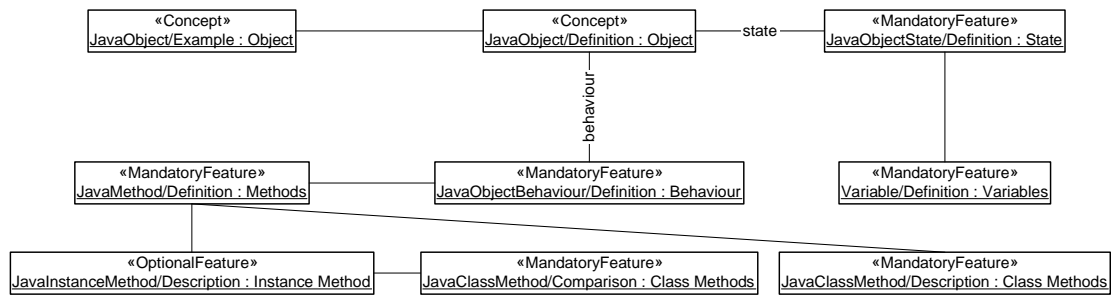
Figure 7.5: An excerpt from a collaboration model showing features from the *Object* feature model

Links. The collaboration links are usually defined as straightforward mappings from associations in the feature model. Special attention has to be paid where there are several instances and roles of the domain features in the model. A designer has to decide which roles and instances will be linked together. In our example, two instances of the `Object` concept appear in the collaboration model. We created a link just between the `Definition` roles of the `JavaObject` and the `JavaObjectBehaviour`. The `Example` role of the `JavaObject` contributes only to the collaboration with the `Definition` role of the `JavaObject` role in our example application, so the link to `JavaObjectBehaviour` is not created. Similarly, the `AND` variation point in the feature model from Fig. 6.4 was resolved as several links between the participating feature instance roles.

Names of the links are suppressed in the model except those of the `state` link between `JavaObject` and `JavaObjectState` and the `behaviour` link between `JavaObject` and `JavaObjectBehaviour`, which we will use descriptive purposes later.

Collaboration of Domain Features with Information/Environment features.

The second refinement of the abstract feature models which follows the domain feature collaborations described above is the refinement into so called *information components*. By information components we mean accessible environment components which are accessed through their interfaces and are able to deliver concrete information about particular domain concepts.

This is achieved by linking the instances of the domain features to the instances of the environment /information features. The refinement augments the above described collaboration models with:

- *Instance roles of environment/information features* mainly to model the pedagogical style and purpose of the information component;

- *Links* between instance roles of environment/information features and domain features instance roles which replace some of the links between domain feature roles (collaboration is delegated to the information components);

- *Messages* being sent between environment/information feature roles and domain feature roles when a user initiates an interactive sequence.

Figure 7.6: An excerpt from a collaboration between features from the *Course* and *Object* feature models

Instance Roles of Environment/Information Features. The roles of information features in collaborations with domain features usually consist of *container, provider* or more domain specific roles such as *narrative text, simulation, problem statement, instruction* and so on. Some of the roles are only suitable for higher level information objects like lecture or module. An example of this is the *instruction* role which refers to the learning theory used to construct an information component. Similar concepts are used in the learning object metadata standard [IEE02].

Figure 7.6 depicts an excerpt from such a collaboration model which links environment features to domain features. The `NarativeText` role is introduced for the `JavaObject`, the `ObjectState`, and the `ObjectBehaviour` instances of `LearningObjects`. The learning objects can be accessed through the `OOConcepts` instruction which is an instance role of `Lectures`. In a final training suite this lecture of the Java tutorial collaborates with other lectures.

Additional Links — Collaboration Delegation. The three learning objects introduced in Fig. 7.6 are linked to the corresponding domain features and/or concepts. The `JavaObject` is linked to the `Definition` role of the `JavaObject`. The `ObjectState` is linked to the `Definition` role of the `JavaObjectState`. The `Definition` role of the `JavaObjectBehaviour` is linked to the `ObjectBehaviour`. Note how these domain features are connected by the `state` and the `behaviour` links in Fig. 7.5. These links are replaced by introduced environment feature instance roles and the collaborations are delegated to information feature roles accessible by a user (in this case the `OOConcept/Instruction`). Remaining links between domain features are

derived from the objects depicted in fig. 7.5.

The collaboration links between environment/information features are created as instances of the associations in environment/information feature models. As in the case of the domain features collaborations, collaboration links have to be resolved by a designer when several roles and instances of one feature or concept appear in the collaboration model and when a variation point has to be transformed.

Messages. In addition, links are annotated by messages which are sent between the roles. User interaction generates the `Access(...)` message to be sent to the `OOConcepts` lecture. The `OOConcepts` lecture generates several `GetObject(...)` messages which are sent to the learning objects. Those learning objects request a content from the domain feature instance roles like `JavaObject` by sending for example the `Show()` messages. A content item is propagated to the user as the result of interaction between the instance roles. The messages used in examples are instances of the functions from domain design models depicted in fig 7.1.

The personalization of a content is reflected in the collaboration diagrams by constraining messages. The `LearnerProfile` parameter of the `Access(...)` message is propagated by several `GetObject(...)` messages to domain features instance roles. Just fragments of the profile relevant to prerequisite competencies and/or topics of the learning objects are transferred by the `GetObject(...)` messages. These fragments are used to determine which presentation options are valid for a learner with a particular level of knowledge, e.g. the two conditionally constrained messages at the link between the `Definition` and `Example` roles of the `JavaObject`. The `UserKnowledge` parameter is used to determine whether the example is shown as a static `Show()` message or animated `Animate()` message. The `ReqValue` is a predefined constant which determines which level of knowledge is required to switch between the presentation options. The conditionally constrained messages at the link between the `JavaMethod` and the `JavaInstanceMethod` can be interpreted similarly.

7.4.2 Personalized Navigation Design for Web Applications

One possible way of looking at navigation is in the context of browsing. Browsing in a web application can be seen in terms of following a certain path in a hypertext graph. From this point of view, a navigation model should concentrate on a user interaction during hypertext presentation or on a change of a hypertext state when the user performs the act of navigating. Nodes in such a path can be either single or composed of several subnodes. A node can also represent subpaths. Navigation models of the browsing scheme display the possible paths through information chunks and their contextual grouping.

Adaptation considered as handling variability at runtime is concerned with different possibilities for presentation of particular information pages or fragments, different possibilities for annotating the fragments and links, different link destinations and so on. This variability is a subset of the variability modeled by variable features of concepts from the conceptual feature model described in the previous chapter. A subset of different aspects of information modeled in the conceptual model can be

selected and placed into the navigation model. These features are then connected by links. Information nodes and links can be constrained with parameters based on user features from the user domain design model.

State Diagrams and Navigation in Web Application

The content fragments, which are composed according to the collaboration models described in the previous section, are connected into navigation trails. The UML state diagrams provide a useful abstraction for designing browsing/navigation trails concerning information or environment in web applications. The main elements of the navigation trail are the navigation state and transition between the states.

Definition 14 (Navigation State) *A navigation state for a user is an information chunk observed by a user at a hypertext node at a given time.*

Definition 15 (Transition in a Navigation Trail) *A Transition in a Navigation Trail is a transition between one navigation state and another. The transition is usually caused by a user interaction event or by another event (e.g. time event). When the transition is fired it leads to the production of a new hypertext node for a user — the new navigation state.*

Transitions represent active interconnections between information chunks, and usually correspond to associations in the application domain model.

Events cause transitions in a state machine; they include user-generated or system-generated events, and the latter include time events.

Atomic states can be grouped into *superstates*. States usually refer to concepts of an application domain. The superstate may comprise of substates arranged in an alternating or parallel fashion.

The *variability in the navigation trails* is modeled by the alternate (OR) states and by decision symbols which can split transition to several alternative transitions. In this way, the navigation trails can contain alternative navigation paths and information chunks constrained by conditions referring to certain user features, content features, device features, or environment features. From the point of view of user handling, this means that each trail can be adapted by taking into account the user's background, level of knowledge, preferences and so on [DN03].

Parallel substates represent information chunks to be presented simultaneously. *Fork* and *join* pseudostates are used respectively for splitting and joining computations and enabling parallelism. The *SyncState* pseudostate is used for synchronizing substates of parallel regions. It means that a navigation state as the product of information chunks can be composite; i.e., the navigation state can present an information chunk using several content fragments of different media types.

Guards can be used to constrain transitions, entry, and exit actions of states by adaptation rules. Usually, they consist of a predicate concerning user profile attributes or context information.

The transitions and events on states are useful abstractions for assigning sensors to observe user evolution. Each transition can have a side effect *action*. Actions can also be performed at entry, exit and as an internal transition side effect of a state.

Figure 7.7: A basic interaction scheme.

The side effect can be for example the modification of a user profile, or the choice of presentation styles for a given chunk of information. Actions can also process parameters used in guards stationed at the exits of branches. Side effect actions, as well as adaptation rules, can be assigned to *entry, exit,* and *do* actions of states.

Tagged values are domain-specific properties used to extend the semantics of elements in UML diagrams. These values can refer, for example, to concepts of the structural model of the application domain, or to specific terminologies which might be useful in identifying additional navigation requirements.

A Process of Adaptive Navigation Design

We propose six basic steps in the navigation modeling process: *identifying interaction schemes, identifying states, identifying transitions, identifying events, creating a user model,* and *mapping the user model elements to the state diagram.* These steps can be performed in parallel and in iterations.

Identifying an interaction scheme.   The basic interaction scheme identifies a basic interaction pattern between user and web application component roles. The main purpose of this step is to understand how content of the web application will be accessed and instantiated within a specific navigation state when the user interacts with the application. This step may therefore be identical with the story collaboration modeling. When a product line management team decides that the story collaboration design is not necessary in their context, the interaction scheme might be reduced to a sequence diagram connecting the environment roles. This is especially the case in the simple domain design example which we have shown in fig 7.1. Figure 7.7 depicts an example of such a sequence diagram, although with slightly changed role names. These roles can be seen simply as instances of added interfaces which are implemented by classes from the domain design.

Figure 7.7 depicts three basic system roles (the `Entry`, the `User Tracker` and the `Course Composer`) and the `User` role. When a user enters the system an assessment test is generated. The user then performs an assessment task. A course is

generated according to the assessed user feature values. This means that the course content is packaged into a navigation sequence (path). When a user interacts with the system by clicking on a particular navigation object, the system evaluates and changes the user model state and regenerates the navigation sequence and other navigation supporting features.

Identifying states.   The states are mapped to information or navigation domain models. There are two possibilities for mappings from an information model:

- a superstate mapped to a class which instantiates content with substates mapped to class attributes, and

- a superstate mapped to a class instance (role) with substates mapped to class instance attributes.

The second case applies in collaboration story models. Parallel substates are mapped to attributes of a class or its instances, which are presented simultaneously. Attributes which do not need to be presented simultaneously are grouped into ordinary substates. Those classes which are aggregated by another class are mapped to parallel or ordinary substates of that class's state. In addition, these substates are determined by the cardinality of the aggregation relationship. Specialized classes are mapped to ordinary substates. Special information chunks derived from several attributes and/or classes or special states needed for the purposes of navigation can also be considered. States can be augmented with a history. The history indicates that a user can restart his browsing at the point where he exited the system in a previous session.

Identifying Transitions.   Association relationships from domain design models or links from the collaboration models are transformed to transitions. Where needed, additional transitions can be incorporated into the model. The fork and join pseudostates and SyncState are intended for modeling the synchronization of parallel states. The first two are intended for splitting or joining transitions. The latter is for synchronizing substates of parallel regions. It means that whenever there is an interaction within fragments which are presented simultaneously with others and the interaction causes a transition which also influences the parallel content fragments, synchronization should be implemented between the fragments. The elements already mentioned are used in the specifications for such situations.

Identifying Events.   Events can be directly mapped to presentation elements with associated actions, i.e. an event is specified for each required interaction element at each navigation state. The events are mediators between the navigation model and the presentation model of actions. Events can be joined to a generalization/ specialization tree. An event can be mapped to more than one transition. There are two types of events which can be considered: those caused by user interaction or caused by a system (e.g. time event or an event raised by an action performed by a system).

Parameterization: Mapping User Model Elements to a State Diagram. Accessible features of user model classes are mapped to guard conditions. The guards represent constraints on variable features of information and variable destinations of links. The guards are tested for specific values, which have to be satisfied when a transition is raised or when a particular state is entered or left. Operations are mapped to actions of transitions and states. They are used for upgrading the user model state or for specific operations with the user model and/or information chunks. Operations for retrieving the current user model state can also be used in guard conditions. Guard conditions of transition and state specify local rules of adaptation. Global rules of adaptation can be specified as guards for internal transitions, points of entry, exits or do actions and conditions of superstates.

Navigation Model of an Adaptive Java e-Lecture

Fig. 7.8 depicts an adaptive navigation model for a simple JAVA e-lecture (introduction to JAVA). This is part of an e-lecture which was provided as a script for a JAVA course at the University of Hanover whose content was created by the course lecturers.
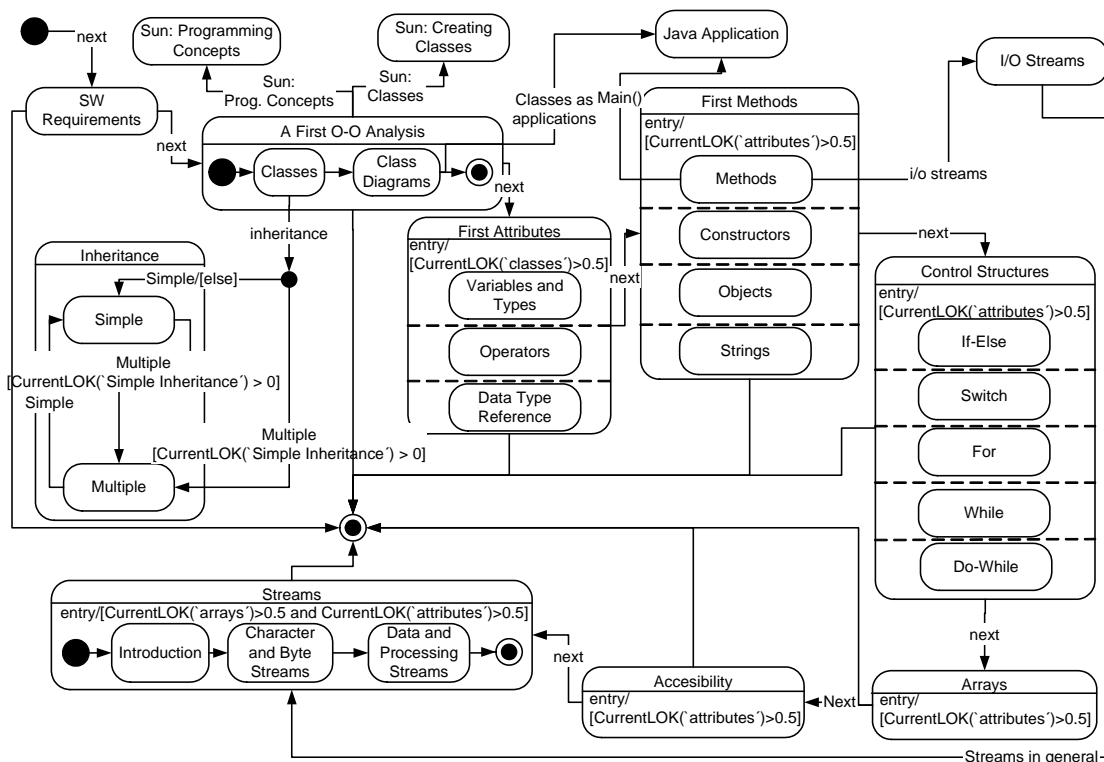
Figure 7.8: Part of an adaptive navigation model for a JAVA e-lecture.

The lecture describes the main concepts of object-oriented programming in JAVA using the example of a software version of a BINGO game. The role of `Requirements` in software projects is described at the beginning of the lecture based on the requirements for the BINGO game. The lecture continues with `A First O-O Analysis` where requirements for the BINGO game are analyzed and transformed into classes.

This part of the lecture conveys the concept of the `Classes` and `Class Diagrams`. Students can request an explanation for `Inheritance`, of which simple and also multiple inheritance are possible. This part also refers to essential programming concepts in an external Sun JAVA introduction. The lecture then goes into more detail on basic concepts in O-O programming, namely the `First Attributes`, the `First Methods`, the `Control Structures`, the `Arrays`, the `Accessibility` (public, private, protected) and the `Streams`.

The following parts of the lecture: `Inheritance`, `A First O-O Analysis`, and `Streams` are modeled as substate machines; i.e., they constitute sub-sequences in this JAVA e-lecture. Only one of the concepts modeled as substates is displayed at any given time. The content fragments about the `First Attributes`, the `First Methods` and the `Control Structures` are presented in one window and thus the fragments are modeled as concurrent states. The fragments can be represented for example using audio or video media. In such a case the concurrent states can model separate presentation channels. When synchronization between these channels is needed, synchronization symbols are used to model such situation.

There are several *variabilities* depicted in the model. Here, we consider variability in the case of the `Inheritance`. When a user already has some knowledge about inheritance[1] (`CurrenLOK( 'Simple Inheritance')>0`), he/she is guided to the learning state on `Multiple` inheritance and otherwise to `Simple` inheritance. The user can switch between these two fragments. The transition (link) from `Simple` inheritance to `Multiple` inheritance is constrained as well.

Variability is considered in all states except of the `SW requirements` and the `A first O-O Analysis`. The guards in the `entry` actions of states thus represent constraints on displaying/showing/hiding/annotating media (text) fragments.

The rules in the `entry` actions are depicted only in the high level states. There are several different ways of understanding the scope of such guards. The guard can be valid for a high level state alone or propagated into its substates because a user is prevented from invoking the subparts. Another interpretation may be that they are propagated by an automatic generator and thus the same rules appear for all the substates in an implementation. Alternatively, similar rules may be written for substates. This is especially useful when they differ from those introduced at higher level states. More complex rules on variability can be introduced in longer courses.

Operations for updating a user profile are not presented in the figure (are collapsed) for simplicity's sake. They are usually modeled as actions of transitions or as exit actions of states. A user profile can be updated as a result of a particular link being clicked or when an information fragment modeled by a state is exited. For example, in the state diagram of Figure 7.8, the user level of knowledge about "Classes and Inheritance" can be acquired in the `Inheritance` module. Exit Procedures for these states do indeed contain similar update operations, such as in the following example:

`CurrentUser.SetLOK(`"Classes and Inheritance"`,0.8,Content).`

Time events can generate an update of a user profile as well. For example a level

---

[1]The personalization specification within the JAVA eLecture state diagram can be based on the user model depicted in Figure 7.3.

of knowledge about a certain topic can be increased after a period of time spent in a certain state.

7.5   Domain Design and State Diagrams

Let us take a closer look at the interaction of state diagram elements and content instances. Each state should generate a content fragment which fits to the current navigation state and user features. As states are abstractions they have to be connected/mapped to content generation components.

In general, there are three ways of doing this:

- To assume that the content is generated and accessible through an identifier or a query. That identifier or query can be used for tagging a state.

- To assume that the content is generated by instances of classes from navigation domain models, environment domain models and application domain models. The appropriate classes have to be instantiated within the entry actions of a state and appropriate methods for displaying the content called.

- To assume that the content is generated by external services. If the services have an application programming interface, this reduces to the previous case and the interface instances have to be used appropriately within the entry actions of states. If the services are provided with design annotations, the symbols of particular functions or concepts from the design annotations can be used to annotate the states and thus link them with the content generation service.

The tagged values represent the most suitable extension mechanism to annotate the states according to the first and third option as they are very close to state variables. Side effect actions are on the other hand best suited for the second approach and also the third approach where a service interface exists.

Content Package.   A `Content` tag (as a state variable) can be used to maintain identifiers or queries for content intended to be displayed within a particular state. If the content is parametrized via the query, the query parameters can be used for content adaptation based on a user profile.

7.5.1   Side Effect Actions with Domain Design Objects

States in state diagrams can have entry actions. Scripts within the entry actions have to be employed to instantiate a given content.

Figure 7.9 depicts an excerpt from a state diagram representing a navigation model for account managers and product managers of a CRM application. The diagram represents one usage case: browsing daily assets. Each day, an account manager looks for accounts to contact in order to provide them with support. The application provides a view of a list of the accounts (the `AccountNameIndex` state). The manager can browse into account details if he needs them for communication with account
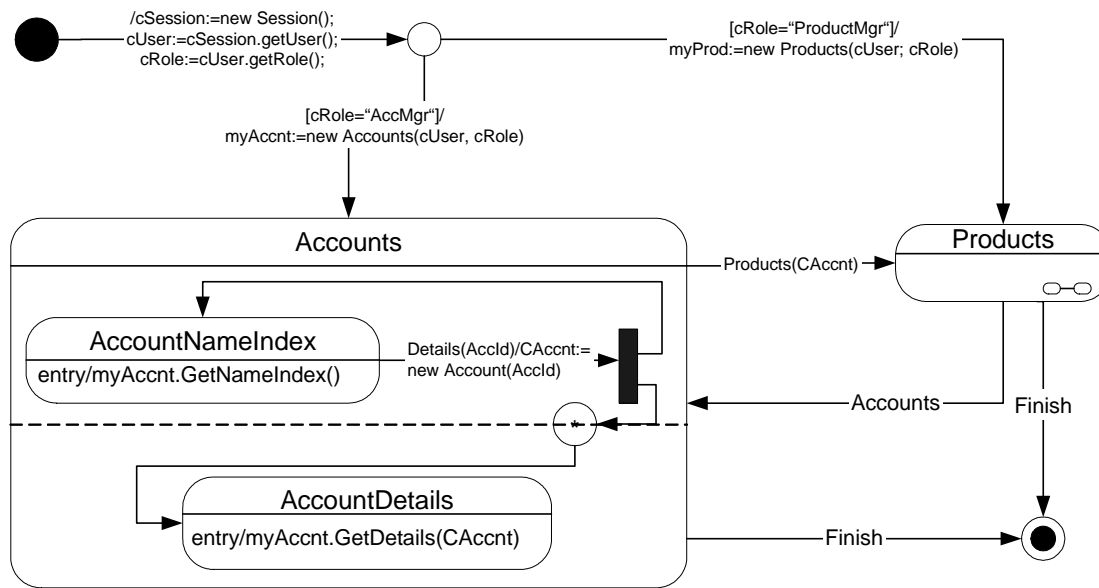
Figure 7.9: An excerpt from a navigation model for browsing assets in a CRM application.

representatives (the `AccountDetails` state). As the details should be displayed simultaneously with the list of accounts, two concurrent regions are depicted in the model. They are synchronized on clicking according to an account identifier taken from the record which was clicked (the transition from the `AccountNameIndex` state raised by the `Details(AccId)` event). As an aid to improving customer support, he can browse details about products sold, problems reported and so on (the `Products` state is displayed in abbreviated (collapsed) form due to space limitations). Product managers enter the system at a point covering those products which they have to support with a summary of the accounts which have bought the products.

The content is instantiated by calling methods of classes from the application domain design model. Similarly, transitions are annotated by guards with classes and methods from the user domain design model to resolve variability in the presentation order of states based on a user's profile.

An excerpt from the CRM application design model relating to the scenario mentioned above is depicted in fig 7.10. The main entities in the domain are `Account` for managing customers and `Product` for maintaining products respectively. Product and Account entities are associated with their business logic collection entities `Accounts` and `Products`. Those are used to retrieve a specific subset of instances of the products and accounts, e.g. the `GetNameIndex(...)` function used in the state diagram of the navigation model to generate a link index of account names for a particular account manager or the `GetDetails(...)` function used to generate details of an account which has been selected by a user from the account name index.

Similarly, fig 7.11 depicts an excerpt from a user domain model for a CRM application from our navigation model. The user model borrows the `SalesMan` entity from the application domain as the users of the application are sales managers and the application is expected to adapt to them according to their roles. A generic `User`
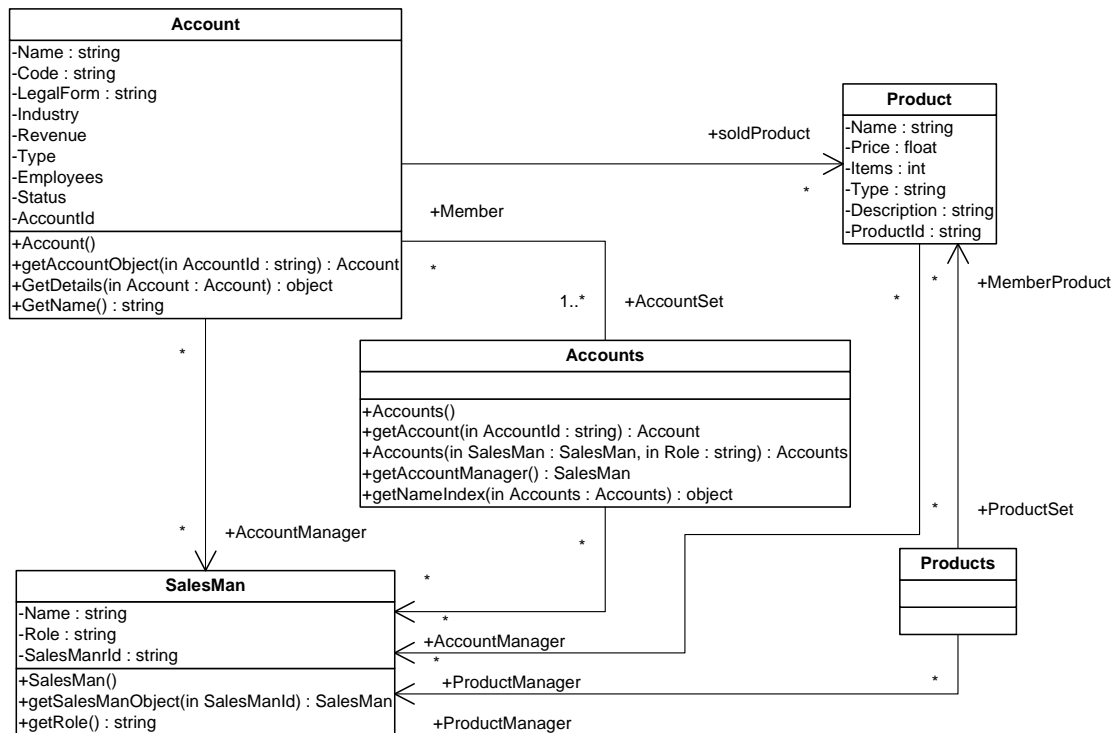
Figure 7.10: An excerpt from an application domain model of a CRM application.

entity with login information and `History` entities are used to track user behavior. The adaptation rules may be based on history or simply on a time of the day which is matched with the account and history records. Furthermore, the recommendations of accounts and products based on the date and status might play an important role in improving the effectiveness of the customer support. An additional entity used particularly for run time purposes is `Session`. The session class is used for example to identify a current user in the transition from the initial state of the state diagram shown in fig. 7.9. This information is used later in the decision branches of the state diagram to decide which state will be entered first according to whether the user is a product or an account manager.

A navigation model modeled by the state diagram with the scripts for instantiating content using domain design classes is still at a logical level. It is close to implementation, but implementation at the level of business logic. It remains necessary to map to presentation styles and pages. A designer has to decide which states to map to pages, at which level of composition to do this as well as which transitions should be mapped to physical links and which should be mapped to functions, buttons, menus and so on. Again, the simplest way is to use tagged values to annotate states with the page concept. A more automated option would be to create generative templates which, for example, create a page for each top level state. Other strategies for assigning states to pages are also possible depending on application, domain and customer requirements.
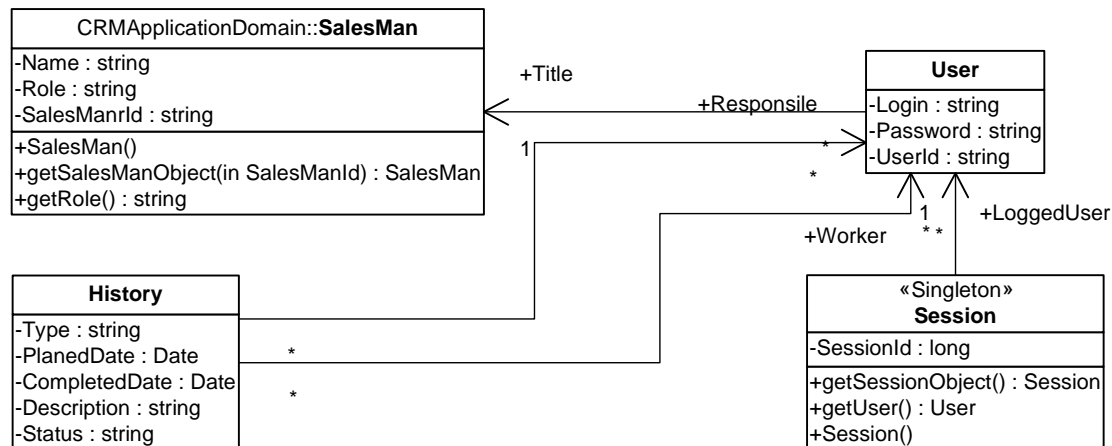
Figure 7.11: An excerpt from a user domain model of a CRM application.

7.5.2 Concepts from Domain Design Models in Tagged Values

The last but not least important option we have listed is to use domain design concepts for tagging states. This means that instead of having just one `Content` tag, we will have more tags depending on which design method a development team chooses.

We experimented with WebML, where content is generated according to hypertext specifications. We used the concepts from the WebML metamodel as a source for tag names and instances of attributes as a source for values to identify appropriate content for connection to a navigation state [CDMN04, CDMN05].

WebML is a visual language for specifying the content structure of a Web application and the organization and presentation of content in one or more hypertexts [CFB$^+$02]. The design process based on WebML starts with the specification of a data schema, expressing the organization of contents by means of the Entity-Relationship primitives. The *WebML Hypertext Model* then allows for a description of how content, previously specified in the data schema, is to be published as hypertext by the application using pages and units as described in section 7.3.
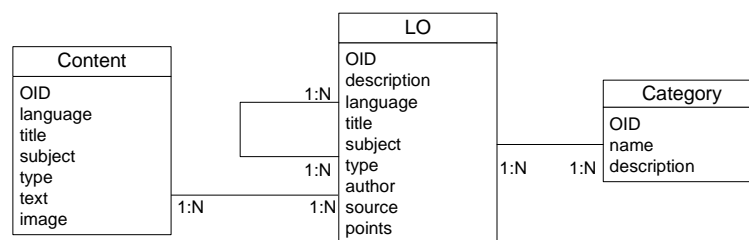
Figure 7.12: WebML Data schema for the e-learning application.

An example of a data schema of WebML which corresponds to the *application domain design* in our terminology is depicted in fig. 7.12. The data schema specifies the database of an e-learning application and is centered on the concept of a Learning Object (LO). The `LO` entity represents descriptions of learning objects using
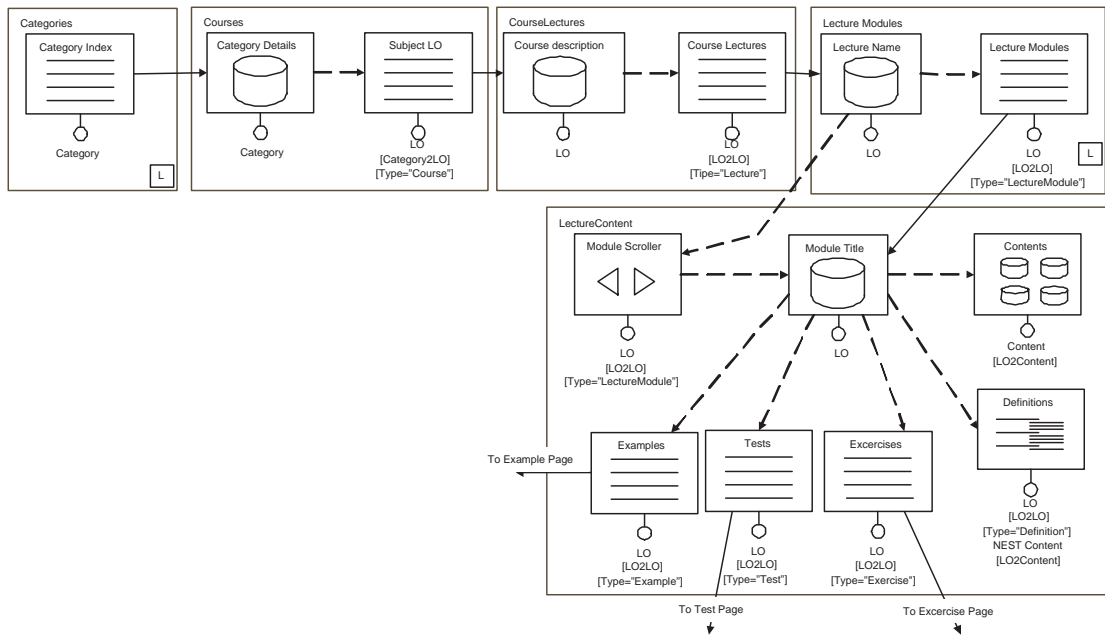
Figure 7.13: The WebML specification of the hypertext interface for the e-learning application.

attributes in accordance with the LOM standard[2]. Amongst these the attribute `type` covers the different types of LOs (e.g. lectures, lecture modules, definitions, exercises, tests) published by the application. Each LO has associations with other LOs: for example, a lecture module can be associated with some related definitions, exercises, examples or tests. The entity `Content` then represents the contents (texts, images, files) which comprise a LO. In order to facilitate LO access, the schema also includes the entity `Category`: This stores the ACM categories that classify the LOs published by the e-learning application.

Figure 7.13 shows a simplified excerpt from the WebML hypertext schema which corresponds to *navigation domain design* in our terminology and is defined for the e-learning application; it refers to pages for selecting a lecture module and accessing its contents as well as associated definitions, exercises examples, and tests. The lecture module selection is governed by means of a navigation chain, in which users progressively select a subject category (`Categories` page), then a course that refers to the selected category (`Courses` page), then a lecture (`CourseLectures` page), and finally the lecture module they are interested in (`LectureModules` page). Some pages like `Categories` and `LectureModules` are marked with an "L" label, which indicates that they are *landmark* pages. This property specifies that the pages will be reachable from any other page of the hypertext by means of landmark links.

The contents of the selected lecture module are shown in page `LectureContent`. Via the `Module Scroller` unit users can browse lecture modules in a *Guided Tour* navigation that allows moving forward and backward in the (ordered) set of modules available for the currently selected lecture. For each module, the data unit `Module`

---

[2]`http://ltsc.ieee.org/`

next [CurrentUser.CurrentLOK
("Object Oriented Programming Concepts")>0]

Overview
{CourseStructure=Tutorial,
LearningPresentation=Summary}

Object Oriented Programming Concepts
exit/ CurrentUser.SetLOK("Classes and Inheritance", 0.2, Content)

What Is an Object

next

What Is a Message

next

Relations To Code
entry/
if(CurrentUser.CurrentLOK
(procedures)<0.5)
show("procedures")

What Is a Class
{CourseStructure=Content,
LearningPresentation=Definition}

next

next

Questions

Language Basics

Variables — next → Operators

next

Expressions ← next — Control Flow

next

next [CurrentUser.CurentLOK
("Language Basics")>0]

[CurrentUser.CurrentLOK
("Language Basics")>0]

Finish

Object Basics and Simple Data Objects

Object Life Cycle — next → Characters and Strings

next

Arrays ← next — Numbers

next [CurrentUser.CurrentLOK
("Object Basics and Simple Data Objects")>0]

[CurrentUser.CurrentLOK
("Classes and Inheritance")>0]

Interfaces and Packages

Creating Interfaces

next

next

Creating and Using Packages

Problems

Common Problems

next

Finish

next [CurrentUser.CurrentLOK
("Classes and Inheritance")>0]

Classes and Inheritance

Creating Classes — next → Managing Inheritance

next

Implementing Nested Classes ← next

[CurrentUser.CurrentLOK
("Object Basics and Simple Data Objects")>0]

Figure 7.14: A navigation model for a Java tutorial in the UML state diagram notation.

`Title` shows the title and a short description of the learning object, the `Contents` multidata unit shows texts and images that compose the module, while the hierarchical index of `Definitions` shows titles of the definitions associated with the module and, nested, the corresponding contents. Three index units then show the lists of examples, tests and exercises available for the current lecture module. The selection of one item from such lists leads users to a different page where the corresponding contents are displayed.

The domain and navigation domain design concepts are used within the state diagrams in a similar way as in the example depicted in fig. 7.14. The state diagram of fig. 7.14 is another example of a personalized learning environment for teaching object-oriented programming in JAVA, borrowed from a well-known Sun tutorial [3]. We have already described another example of a Java eLecture in section 7.4.2 which followed a different instruction path. The personalization example chosen here focuses on link adaptation.

The tutorial starts with an overview of available lectures, as represented by the `Overview` state, which summarizes the available lectures in the tutorial, as represented by the `Summary` value in the `LearningPresentation` tagged value. It also presents the high level tutorial steps (`Tutorial` value in the `CourseStructure` tagged value). Links from the overview point not only to the first section of the tutorial,

---

[3]See `http://java.sun.com/docs/books/tutorial/java/index.html`.

but also to the other main sections; all these links, except for the first one, are associated with guard conditions that check that the user has enough knowledge to jump directly to the respective lectures.

The next step after the `Overview` is a lecture on `Object Oriented Programming Concepts`. This state is accessible without the user needing to satisfy any criteria in respect of background knowledge; it is a composite state containing four steps, represented by four substates: `What is an Object`, `What is a Message`, `What is a Class`, and `Relations to Code`. The `Relations to Code` state also exposes an `entry procedure` addressing *content level adaptation*. The procedure applies to a learning step about building programs; it states that if the current user does not have sufficient knowledge on basic concepts about object-oriented programming procedures, then learning content on procedures will be added.

The next step from the `Object Oriented Programming Concepts` is the composite state `Language Basics`. The transition between the two states features a `next` event and a guard. The guard specifies a *link level adaptation* rule, saying that the link is recommended when the user's current knowledge level is greater then zero. The other learning steps modeled in the state diagram can be interpreted similarly.

The personalization specification within state diagrams is again based on the user model depicted in Figure 7.3.

A performance value is used to determine if a transition into a new state is appropriate and must be suggested to a given user. For example, the following condition:

```
[CurrentUser.CurrentLOK("Classes and Inheritance")>0]
```

is a guard in the state diagram that determines whether a link can be followed between the `Classes and Inheritance` state and the `Interfaces and Packages` state, based on the user's current knowledge level. The `Performance` class also maintains the levels of competence, recorded date and a metric used to measure the level of competence.

The `User` class provides operations to set and get the acquired level of knowledge or level of competence. These operations are used in guards and actions for adaptivity rules, and for updating the learner profile. For example, in the state diagram of Figure 7.14, the user's level of knowledge about "Classes and Inheritance" can be acquired either in the `Object Oriented Programming Concepts` lecture or in the `Classes and Inheritance` lecture. Exit Procedures of these states indeed contain similar update operations such as the following:

```
CurrentUser.SetLOK("Classes and Inheritance",0.2,Content).
```

The state diagram from figure 7.14 still uses application domain concepts. To map states to the pages and units generating content, the state diagram must be extended with tagged values to be used as pointers to content. This activity must be performed by UML-Guide designers, typically in the course of the transformations required for "implementing" navigation trails starting from their high-level descriptions.

It is worth noting that state diagrams augmentation does not require a complete mapping between the state diagram components and WebML conceptual primitives.
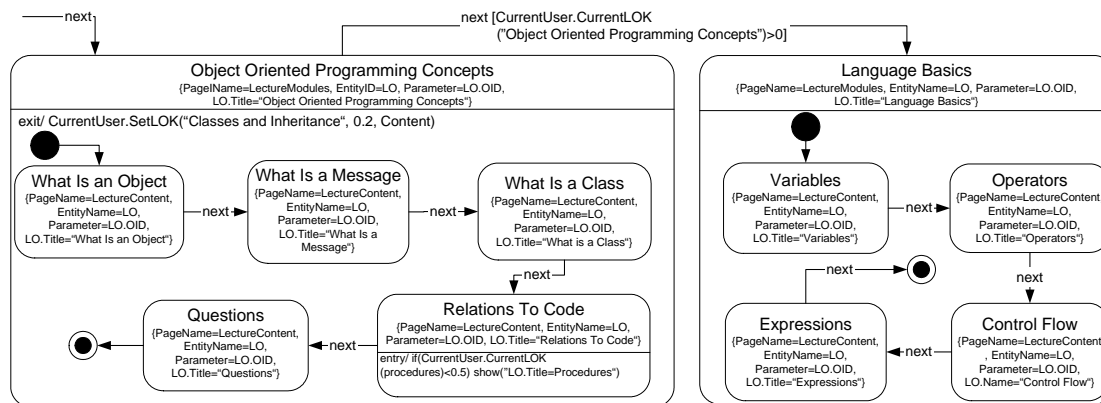
Figure 7.15: Excerpt from the UML-Guide state diagram extended with tagged values representing WebML concepts.

Navigation designers just need to specify the IDs of those WebML elements that are to be used for URL construction, i.e. pages, source entities from which page contents are extracted and entity attributes that are used in conditions for querying contents.

Figure 10.1 depicts an excerpt from a state diagram extended with tagged values for WebML concepts needed for computing WebML links. For instance, `Object Oriented Programming Concepts` is a lecture. The corresponding page name is `LectureModules` from the WebML hypertext model. The entity used to store lectures in the WebML data model is `LO`. The title used as an attribute to identify the lecture is the same as the state name. Entry and exit actions are transformed if they send parameters into WebML links, as in the case of `Relations To Code` (where the parameter of the `show` method is replaced by the specific WebML parameter `&LO.Title=Procedures`, used for selecting LOs about Java procedures). Although in our example tagged values for page and entity names are constant values, in more complex cases they can be specified as parametric selectors as well, so that they can automatically retrieve their values from the WebML specification based on specific conditions.

Chapter 8

Summary

We have proposed a domain engineering framework for adaptive web applications. The framework has following features:

- It defines the *domains of concern in domain analysis* for web applications: *application domain, environment, and user domain.* The application domain is a domain of information to be served in the web application. The environment domain is a domain of information organization and delivery. The user domain is a domain of user features which drive the adaptation in web application.

  - It uses *conceptual models* which are used to model *concepts (vocabulary) of interest in the domains* and linguistic relations between them.
  - It treats *adaptation as the configuration aspect* modeled in *feature models* where optionality, variation points, and configuration dependencies between features represent *adaptable (variable) aspect of adaptive web applications* while mandatory features represent the *common aspect of adaptive web applications* in a family. The adaptation is considered as static, i.e., adaptation (customization) by a designer, and dynamic, i.e., adaptation at runtime to certain features of a user.

- It defines an approach to specify *dynamic connectors* between the domains in terms of *story collaborations* in collaboration diagrams. The *connectors* are *links and messages* which bind together features, which represent *content fragments with a certain role in a content*, and *environment features*, which represent the *access structures to the content fragments*. Furthermore, the links and messages are constrained by *user domain features* to *constrain* fragments which are optional or variable. This features adaptive content composition specification for web application.

- It features *state diagrams* to specify *adaptive navigation trails* over information space. The states in state diagrams represent a user step in a navigation trail and a transition represents a move between the steps. The state diagrams bind content story collaborations to sequences of the stories which lead to a particular information goal. The *optional and variable transitions and states* can be *constrained by user domain features*. This features *adaptive navigation specification for web application.*

Advantage for Designers.   The main advantage of the framework is that it allows to reason about domains and adaptation separately but still provides a possibility to connect separate domains. Each domain can be considered as a close domain which is put into wider context when a particular application is instantiated. The context can be defined by different customers, which is another advantage of having the domains separated. An analyst or a developer can use the domain specifications in conceptual and feature models to communicate which options a customer already have and what should be still developed. In addition, it sets a common ground also for negotiation purposes about adaptation, i.e., which features of application should be provided just to some users and which to all.

Besides the reasoning, documentation, and communication purposes, the models can be used also for generating some aspects of the application. The models (especially the collaboration and state diagrams), when given in appropriate representation, provide information about software and information components of final application which can be used for generating environments, pages, and adaptation decisions. Some generators will be shown in our case studies in part III. The generators provide a rapid prototyping solutions so that developers can very promptly react to changed requirements of a customer by providing him with quick possibility to touch an application.

Advantage for Applications.   The fact that the models can be applied in generators and metadata implies that more generic applications interpreting the models can be build. The domain models plug in the domains into the applications, so the application infrastructure can be based on generic software components and HTML generators. Furthermore, the models can be used:

- *for search purposes* — to search for information about certain story,

- *provision purposes* — to provide information, environments, stories, and trails to external parties, and

- *integration purposes* — to semi automatically integrate information sources and applications which use them.

This usage of models in metadata, services, and adaptation will be shown in the case study in chapter 11.

Advantage for a Process.   First of all, the framework features an approach to systematically identify and manage adaptation aspects of web applications even by separating them in particular domains. The cross domain adaptation occurs then in the models which connect the domains. This allows for having team roles which are directly responsible for the adaptation aspects and can concentrate on them. Furthermore, if the adaptation aspects are not needed to be considered, the activities concerned with them can be omitted in the development process due to self-containment of the activities. In addition, the activities which are concerned with the adaptation aspects can be plugged into other methods for web application development as they have well defined interface in terms of concepts and models created within them. This aspect will be studied in case study in chapter 10.

Part III

Use Cases for the Conceptual Models

Chapter 9

The UML-Guide: Generating Adaptive Navigation from State Diagrams

The main purpose of the software design models and abstractions which they provide is to understand, reason about, and document the design views of software applications to be developed and deployed.

We have introduced a new engineering framework with new models to be used to design adaptive Web applications. The purpose of this case study is to show that one of the newly introduced models is generative, i.e., it can be used also for generating running Web application code for the aspect modeled by the model in addition to its documentation purpose.

Since UML models can be stored in XMI file, the OMG standard XML metadata interchange format files, it is possible to process and manipulate the files by standard XML processing tools (XML parsers, XSLT parsers and so on) [Ste03]. This simplifies a construction of generators of run-time software artifacts based on UML models including production of (adaptive) Web applications.

In this chapter we explain how the models (UML state diagram models for navigation and class diagrams for user modeling) can be utilized for generation of adaptive navigation sequences. The method utilizes the availability of the XML based standard for storing UML models — the XMI. This enables to take full advantage of XML technology which is very popular for making Web-based applications.

The adaptive navigation model presented in section 7.4.2 represents adaptive sequence of steps through the Web site of the Java course (or through the content items provided within the course). In this chapter we discuss a visualization of such a model on an adaptive Web site map (graph). We discuss a method which transforms the state diagram model into the Web site graph (navigation map). We describe an implementation of the generator provided as a transformation method utilizing the XMI and the XSLT. We discuss an implementation of a system which uses the generator to adaptively (re-)generate the navigation map as well. As the main purpose of the generator is to provide adaptive navigation guides by utilizing the UML models, we named the generating system the UML-Guide.
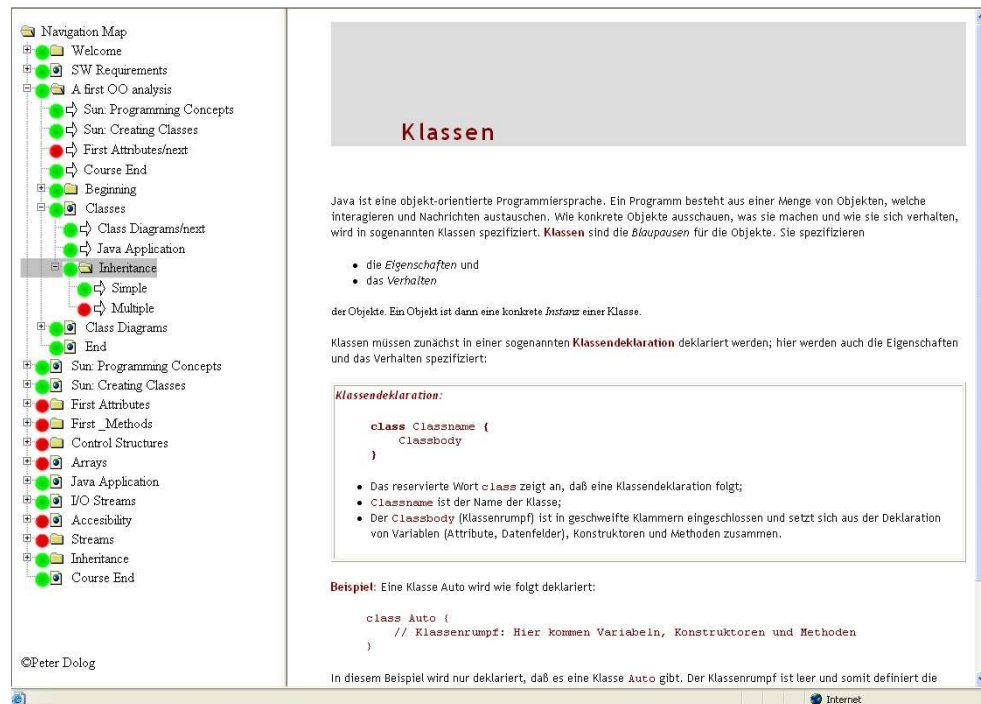
Figure 9.1: Visualization of navigation graph for java e-lecture.

9.1 Visualization of the Navigation Map

One approach to the visualization of the navigation map is depicted in fig. 9.1. We distinguish [DN03]:

- *Folder symbol* — which represents a composite information fragment composed from other composite information fragments, simple information fragments, groups of links or simple links;

- *Dashed box symbol* — which represents a composite information fragment, which has to be presented concurrently with other composite information fragments (the dashed boxes) depicted on the same level;

- *Document symbol* — which represents a simple information fragment; only links can be nested under the simple information fragment;

- *Arrow symbol* — which represents a simple link to another composite or simple information fragment; the arrow symbols can be nested under the folder when they represent different alternatives of link destination from particular document (e.g. grayed folder `Inheritance` with two link alternatives: `Simple` and `Multiple`);

A *composition* is represented by the plus/minus symbol for showing/hiding enclosed items and by the left hand indent of enclosed items.

A content can be associated to each symbol. A content and a name of corresponding target node is associated to arrows. The "/next" string is added to the names of

the arrows which represent guidance to the next fragment according to the course sequence.

We have implemented described structure for Web browsers. We have also implemented the functions for filling this structure and for interpreting the user actions on this structure. Current position of the user in the navigation graph is indicated by grayed background of the presented element. As it is obvious from Fig. 9.1, the navigation map was generated from the navigation model of JAVA e-lecture in UML state diagram depicted in Fig. 7.8.

The generator parameters are used to switch between several adaptive navigation techniques. For example, the links can be either annotated by appropriate color from traffic light metaphor or showed, hidden or sorted according to values assigned to different user features used in adaptation constraints. The green symbol annotates the documents which are appropriate for a user. The red symbol annotates the documents which are not appropriate for a user. Yellow or other symbols can be used to indicate other information about the document, e.g., already read. The actions assigned to the transitions or to entry or exit events of the states are transformed to the calls for operations over internal representation of the navigation map. The skeleton for implementation of such operations is generated as well. The code of this operations has to be filled manually. Operations are created as member functions of user model classes according to their specifications. We have implemented operations from the user model depicted in the figure 7.2.

9.2   Transforming State Diagrams into Navigation Map

A Method.   The input to the transformation method is a state diagram. The output is a navigation map. The transformation is carried out according to the following method:

1. Find the initial pseudostate and transform it to the *Folder* symbol. Embed outgoing transitions from the initial state as *Arrow* symbols under the created *Folder* symbol. Set the content of the *Arrow*-s to the content of outgoing transitions target. Make visible the *Arrows* whose guards of transitions are satisfied or annotate the *Arrows* by appropriate symbol from traffic lights metaphor (red, green, and yellow).

2. Transform each composite state to:

   (a) The *Folder* symbol if it is a state with alternative substates or substatemachine.

   (b) The *Dashed box* symbol if it is a concurrent region.

   Apply these rules for all substates recursively. Put substates under their composite ancestor. Make visible the components whose entry guards are satisfied or annotate the symbols by appropriate symbol from traffic lights metaphor (red, green, and yellow).

3. Transform each simple state to the *Document* symbol. Make visible the fragments, whose entry guards are satisfied, or annotate the nodes by appropriate symbol from traffic lights metaphor (red, green, and yellow).

4. Transform each transition to the *Arrow* symbol:

    (a) If the target of a transition is a state, assign the content target of this state to the symbol.

    (b) If the target is a peseudostate (junction, choice, join, fork), transform it to as much *Arrow* symbols as the number of outgoing transitions of the pseudostate. Group the arcs under the *Folder* symbol with the name of the incoming transition of the pseudostate. Assign the content of the outgoing transitions target states to these *Arrow* symbols.

    Associate the *Arrow*-s to their sources (make them subelements of their sources) and make visible the *Arrow*-s whose guard is satisfied or annotate the *Arrow*-s by appropriate symbol from traffic lights metaphor (red, green, and yellow).

5. Find the final state and transform it to the *Document* symbol. Associate content to it and make visible the document whose guard is satisfied or annotate the *Document* by appropriate symbol from traffic lights metaphor (red, green, and yellow).

```
...
<UML:SimpleState xmi.id = 'a53' name = 'SW Requirements'
    isSpecification = 'false'>
  <UML:ModelElement.taggedValue>
    <UML:TaggedValue xmi.id = 'a54' isSpecification = 'false'
       dataValue = '127-0-0-1-206be6:f479423af6:-7ffb'>
     <UML:TaggedValue.type>
        <UML:TagDefinition xmi.idref = 'a39'/>
      </UML:TaggedValue.type>
    </UML:TaggedValue>
    <UML:TaggedValue xmi.id = 'a55' isSpecification = 'false'
       dataValue = 'Course-2/course_info1/lesson_01/sco01.htm'>
     <UML:TaggedValue.type>
        <UML:TagDefinition xmi.idref = 'a51'/>
      </UML:TaggedValue.type>
     </UML:TaggedValue>
    </UML:ModelElement.taggedValue>
    <UML:StateVertex.outgoing>
       <UML:Transition xmi.idref = 'a56'/>
    </UML:StateVertex.outgoing>
    <UML:StateVertex.incoming>
       <UML:Transition xmi.idref = 'a52'/>
    </UML:StateVertex.incoming>
</UML:SimpleState>
...
```

Figure 9.2: A part of the XMI document for the state diagram of the Java e-lecture: `SW Requirements` simple state with reference of incoming and outgoing transition.

XMI. The UML model is represented by an XMI (XML Metadata Interchange) document [Gro00b]. An XMI document is an XML document where the tags are elements from the UML metamodel. An example of the XMI document fragment for the state diagram is depicted in Fig. 9.2. The XMI document can be generated directly using a CASE tool (e. g. ArgoUML [arg], or its commercial version Poseidon [pos]). Figure 9.2 depicts the `SW Requirements` simple state as the one of the state machine subvertexes together with the outgoing and the incoming transition references.

XSLT Templates as Generation Rules. XSLT templates are used to transform one XML document to another XML, HTML or text document. The XSLT templates contain transformation rules for transforming the XMI file to code fragments for filling the navigation map. The rules are interpreted by an XSLT parser which generates functions for inserting symbols into the navigation map structure.

```
...
<xsl:template match="UML:SimpleState" mode="transition">
  <xsl:param name="levelp"/>
  <xsl:param name="namep"/>
  <xsl:param name="stateidp"/>
  <xsl:param name="transnamep"/>
  <xsl:variable name="fldname" select="@name"/>
  <xsl:variable name="cond" select=
  "UML:State.entry/UML:CallAction/UML:Action.script/
    UML:ActionExpression/@body"/>
  <xsl:choose>
    <xsl:when test="$cond">
     <xsl:text>if (</xsl:text><xsl:value-of select="$cond"/>
     <xsl:text>) {</xsl:text>
     <xsl:value-of select="concat('aux', $levelp+1)"/> =
       insDoc(<xsl:value-of select="concat('aux', $levelp)"/>,
       gLnkLnk(2, "<xsl:value-of select="$fldname"/>
       <xsl:if test="$transnamep='next'">/
       <xsl:value-of select="$transnamep"/></xsl:if>",
       "<xsl:apply-templates select=
       "UML:ModelElement.taggedValue"/>",
       "<xsl:value-of select="$fldname"/>",
       "<xsl:if test="$visibility">green</xsl:if>",
      "<xsl:value-of select="$stateidp"/>"))
    <xsl:text>}
    </xsl:text>
    <xsl:text>else{
    </xsl:text>
     ...
    </xsl:when>
     ...
  </xsl:choose>
</xsl:template>
...
```

Figure 9.3: The example of XSLT template part for transforming simple state as a target of a transition.
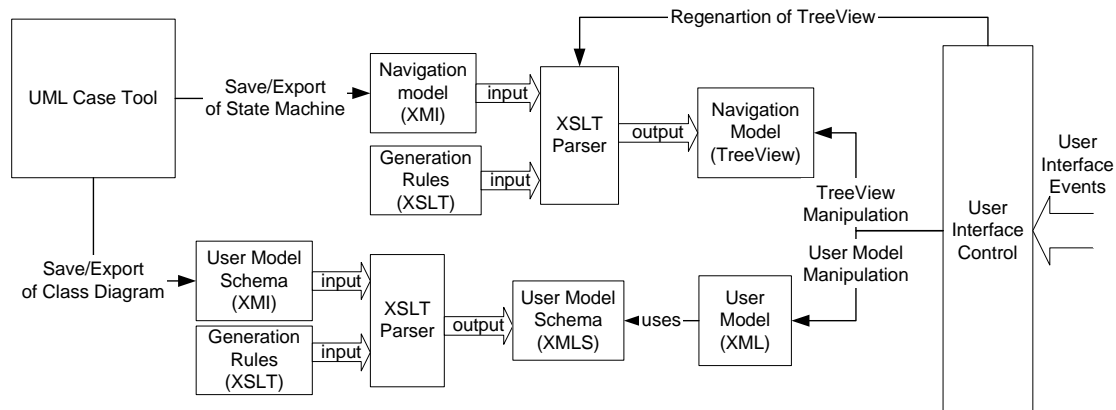
Figure 9.4: A general architecture of generator and final application.

Figure 9.3 depicts a fragment of a template where an arrow symbol is generated (`gLnkLnk(...)` function in the template). The template applies for a target simple state of a transition (`mode="transition"`). The `levelp` parameter determines where the arrow symbol has to be enclosed (under folder, document, or group of transitions). Target information fragment of the link (arrow) is passed to the template through `namep` parameter. The `stateid` parameter is used to identify the exact position of the target information fragment in the map. This is used to change the indication of current position of the user in the map. If "next" event is associated to a transition then this string is added to the target state name to indicate the next information fragment according to the planned path. The global `visibility` parameter is used to switch between the presentation options (annotation by traffic lights or showing/hiding).

9.3   System Implementation

Figure 9.4 depicts an architecture of the implemented system. The central parts of the system are user interface control over the navigation map and XSLT parser, which regenerates the navigation map according to the current state of a user profile. The navigation map is regenerated according to rules described in previous sections. The XML schema for user profile structure is generated from the user model class diagram.

We use standard library for manipulating user records in XML. The user records are maintain at the client side during browsing. They are transmitted to the server database when user closes the course in whichever state. When a user opens the course again his profile is initialized from server database. The navigation map is manipulated at the client side as well but it is possible to implement it at the server side as well.

When a user enters the system first time, initial navigation map is generated. The initial generation is derived from the entry state and according to the rules for nodes visibility, annotations, and collapsing. The user profile is initialized according to the entry user assessment (entry form).

Events generated by user actions at the user interface invoke associated actions which process and store new values to the user profile. They also initiate regeneration of the navigation map according to the new values from the user profile.

The XMI document is generated by standard export facility of Poseidon UML tool. We have implemented the XSLT templates for transforming the XMI representation to the format described in the section 9.1 (Visualization of the navigation map). We have also developed the user interface control mechanism, which interprets user actions over navigation model (tree).

9.4   Lessons Learned

State Diagrams and Other Approaches.   The key issue which is addressed by this case study is the focus on user interaction and user's and system's generated events in navigation. This can be naturally described by utilizing the UML state diagrams.

The important aspect of in navigation is also how content items are grouped in certain navigation nodes and relationships between such composite structures. This has been addressed by means of views in OOHDM [SR98] or navigation classes in UWE [Koc01]. In this chapter, the content was referenced by a content identifiers in the introduced "Content" tagged value of a state. To fully benefit also from a dynamic content generation, the state diagram approach should be aided with the structural methods (for example OOHDM [SR98], UWE [Koc01], WebML [CFB00], W2000 [BGP01] or other methods which are reviewed in chapter 3). The integration can be made as addressed in section 7.5 by introducing concepts from the domain design models, for example operations used as the side effect actions in a state machine or as additional tagged values to enhance the state machines produced.

The introduced state diagrams complement the structural methods for navigation modeling by focusing user interaction and traversing events while the structural approaches are more focused on composition and structural relationships between the navigation structures. The guidelines for integration of both views can provide useful understandings of Web application engineering processes. This issue is the main concern of our second case study on the integration of the adaptive navigation modeling with the WebML reported in the next chapter. Similarly, our state diagram approach can also aid the navigation analysis based on a user aim introduced in [CK02].

The importance of behavioral modeling techniques as a complement to the structural techniques was recognized by other authors as well. For example, the WebML has been extended by workflow oriented modeling approach in [BCC+03].

Several XML based transformation methods have been introduced for the Web-based information systems such as the generators for WCML [SG00] or WebML [CFB00]. Another approach to rapid prototyping has been introduced in [SL02]. It is based on use case models, activity diagrams and Web site structure models. An approach to generate HTML pages from page and data graphs was presented in [LFS+98]. Our approach is based on the XSLT transformation templates. Tools employed are similar to those transformation approaches based on the XML. Our transformation templates are similar to the Abstract Presentation Diagram Refinement [GCP01]. They employ

rule based approach[1] to refine the diagrams taking specific conditions of target environment into account. We follow two level rule based approach. Adaptation (restriction) rules based on user model are specified in the state diagrams. Transformation rules are encoded in the XSLT templates, which serve for transformation of the state diagrams to navigation maps as one possible visualization.

Visualization.   There are other possibilities to visualize and implement a navigation support. For example the navigation map can be restricted just to the nodes which reflect actual position of a user with links to the next and related information fragments and the folders to which the currently presented information fragment belongs to. If a user clicks a link which brings him to an information fragment which has not been contained in the presented map, the map is regenerated to visualize a new context of the user.

Generator.   The generator can be further parametrized to generate just outgoing and incoming links as a navigation guide. The generator generates scripts for creating the folder structure for HTML content referenced in the navigation map. It can be further extended to generate HTML files with anchors for links modeled by transitions and division tags for blocks modeled by substates. Adaptation operations can be generated as parameters of events for such anchors. The page skeletons with generated anchors can be then filled with a content. All this variability at the generator level can be captured by the XSLT template parameters.

More complex presentations require more than one generator template. Consider for example a situation when the most read topics for last three days should be displayed and sorted according to user preferences and number of visits for a particular information node. This can be achieved by three templates. One template will transform user models stored in XML format to another XML document, which will contain topics, URLs of the content items which present the topics, and count of users who visited the topics. Another XSLT template can enhance this file by generating additional annotations from XMI file (e.g. links between the topics, subparts of the text fragments, and so on). Sorting mechanism of the XSLT templates can be finally applied in third template, which will sort the XML document according to conditions which have been set by a designer during the design phase(e.g. count of users).

---

[1]They use a language similar to OCL.

Chapter 10

Domain Specific Languages with the UML-Guide

The UML is a modeling language very close to the object-oriented programming environments. However, the Web, especially from presentation point of view, does not provide an execution environment similar to the operating systems where applications can run as binary distributions. The Web applications are accessible through document centric hypertext pages though recently extensions of applications servers allowed to use a sort of object-oriented principles within Java Server Pages (JSP), Active Server Pages or other technologies. Researchers started intuitively to look at extensions or languages which are closer to the hypertext domain and document centric presentation facilities of the Web.

According to [vDKV00, DK02], a domain-specific language is a programming or executable specification language that offers, through appropriate notation and abstraction, expressive power focused on, and usually restricted to a particular problem domain. According to that definition, the languages, which are provided with extensions to the Web servers to provide execution environment for the models/programs described by them can be called domain-specific languages for the Web, the navigation, or the hypertext domain.

WebML [CFB$^+$02] is one of the languages which can be classified as a domain-specific language for data-intensive Web applications. WebML-based development is supported by a CASE tool [CFB$^+$03], which offers a visual environment for drawing the WebML conceptual schemas, and supports the automatic generation of server-side code. The generated applications run in a standard runtime framework on top of Java 2 application servers, and have a flexible, service-based architecture allowing components customization. The domain specificity is located especially in the hypertext schema specification which is executable in the extensions of the Tomcat Web server and its JSP engine and Microsoft .Net platform.

The UML is on the other hand application domain independent. The UML state diagrams for adaptive navigation trails with class diagrams for user models have to be specialized or mapped to the models which are executable on the Web. In previous chapter, we have described how to generate Web-based user interface for adaptive navigation from the navigation guides specified by the state diagrams. In that study, we have not explored the content generation part to full extent.

In this chapter we present a case study, which maps the UML state machines on the WebML application and hypertext model executable at the WebML platform. The main purpose of this study is to show whether:

- State diagrams can be expanded with the content generation concepts and thus to provide more generic solution for adaptive Web-based application design.

- State diagrams complement existing structural solutions for the navigation design and as such can be plugged in or out of the development process as needed.

10.1   Generating Adaptive Navigation over WebML Generated Application

We have studied the integration of WebML with the UML-Guide at composing an e-learning WebML application with the UML-Guide that is focused on a specific learning goal [CDMN04, CDMN05]. The users are offered with the user interface generated by the WebML engine, populated by content spawning a large body of knowledge. More focused learners are offered with a guide, available on an interface that can be opened "aside" the main one, and that points to pages and contents published by the WebML-generated interface, according to a specific learning objective and user experience.

The case study is based on the conceptual models from section 7.5.2. To remember, the excerpt of a state diagram from section 7.5.2, which is taken for integration purposes, is depicted in fig. 10.1. The states are extended with tagged values for WebML concepts as pointed in section 7.5.2. The process consists of two steps: identifying application domain concepts to be used to identify content and identifying hypertext elements to be used to present the content.

For instance, `Object Oriented Programming Concepts` is a lecture. The corresponding page name is `LectureModules` from WebML hypertext model. The entity used to store lectures in the WebML data model is `LO`. The title used as an attribute to identify the lecture is the same as the state name. Entry and exit actions are transformed if they send parameters into WebML links, as it is in the case of `Relations To Code` (where the parameter of the `show` method is replaced by the specific WebML parameter `&LO.Title=Procedures`).

Generating Links.   The existing UML-Guide generator described in previous case-study in chapter 9 accessed the content through URLs assigned to each symbol of the navigation map. To be able to reuse the generator, links have to be generated according to the tagged values used in states. The WebML runtime works similarly, it generates URLs out of the link specifications in the WebML hypertext schemas pointing to the pages which are targets of particular link. Therefore, each page in WebML runtime is accessible through a link generated from object identifier of the page and URL of the Web application. WebML links take the format:

```
ApplicationURL/page_identifier.do?ParameterList
```

Figure 10.1: Excerpt of the UML-Guide state diagram extended with tagged values representing WebML concepts.

where `page_identifier` denotes a WebML page and `ParameterList` is a list of tag-value pairs, in the form `entity_id.attribute=parameter`. Thus, UML-Guide state diagrams must be extended with tagged values to be used as pointers to WebML concepts.

The UML-Guide generator must be supplemented with additional procedure concerned with the generation of WebML links "pointing" to the WebML-controlled portion of the application, to be addressed while building the UML-Guide interface.

The links are generated by submitting queries for retrieving OID's of the WebML concepts. The OIDs of pages are retrieved from the WebML design documents while the content OIDs and parameters are retrieved from an application database. The XMI document containing the UML state diagram is rewritten to add a Content tagged value containing the generated link. After that step, the original UML-Guide generator for visualizing navigation map can be used as is.

10.2   System Implementation

The implementation is based on integration of the original UML-Guide engine implemented as combination of JSP, JavaScript and Java and the WebML Web Ratio platform. The integration is loose and preserves the distinctive features of the two systems. In particular, some nodes and links in the UML-Guide state diagram point to content that is managed in the WebML e-learning application; therefore, the integration of UML-Guide with WebML requires UML-Guide adopting WebML concepts, such as page identifiers and content identifiers. In this way, concepts used as state names or as tagged values within UML-Guide are mapped to learning resources stored in the database generated from the WebML data model.

In the resulting application, the user-specific adaptation occurs in UML-Guide. The proposed solution is an example of how client-side computations, specified at high-level in UML, can integrate WebML-designed solutions. As such, this experiment can be replicated for many other applications and the focus on UML-Guide can pursue different objectives.

Figure 10.2: Architecture of the composed system.

Figure 10.3: Visualization of navigation graph for java e-lecture.

Figure 10.2 describes the system architecture. The high-level WebML and UML-Guide specifications are mapped into XML-based internal representations, respectively built by the code generator component of WebRatio [CFB$^+$03] and by the XMI generator of Poseidon.

The WebML run-time component runs JSP templates (also embedding SQL), and uses XSL style sheets for building the application's presentation. The XMI representation of a UML-Guide drives a run-time adaptation engine, written in XSLT, which dynamically changes the content of the profile variables and produces the UML-Guide user interface. The WebML and UML-Guide interfaces are then composed and presented to the user.

10.3   Visualization of Integrated Application

The user profile records are maintained at the client side. When users begin a new session, their profile is initialized from a client-side XML-based database. The navigation map is manipulated at the client side as well. Java script is used to implement the user interface control and user profile manipulation. The events, which are generated by user actions on the user interface, invoke profile adaptation actions, which possibly process and add new values to the user profile. They also trigger regeneration of the navigation map, according to the newly computed values.

The navigation map responds to changes in user profile by changing recommendation annotations (e.g., changing colors of nodes or showing/hiding nodes). When specific requirements, for example those set by conditions in entry actions of states, are met, the WebML vertical adapts delivered content based on additional parameters that UML-Guide is able to send to the server-side application.

Figure 10.3 highlights a lecture on "What is an Object". The UML-Guide panel placed on the left shows the position of the user reading the material for the module by the shaded background. The content of the lecture is delivered by the WebML vertical based on the generated link that is assigned to the document symbol at the "What is an Object" entry. The symbol is generated from the simple state with the same name depicted in Figure 10.1. The state has a transition to the next state "What is a Message", which in the UML-Guide panel is depicted as an outgoing arrow, under the symbol of the current lesson. As the user has sufficient background knowledge needed to understand the next step in his learning path, the direct next steps are annotated by a green ball. Further rules apply for additional entries to hide documents and folders which are not relevant to the user's learning goals.

The simple state "What is an Object" is a substate of the "Object Oriented Programming Concepts" state, which is rendered as a folder symbol in the navigation map. The constraints and side effect actions are transformed into conditions and procedure calls in the UML-Guide which dynamically generates the traffic-light annotations. Other symbols and their grouping under folders are generated similarly from the state diagram according to the method described in 9.2.

10.4   Lessons Learned

State Diagram and Content Generation.   We have shown that the integration of the UML guide and the WebML can be achieved. Furthermore, we have shown that the integration can be done at the conceptual level by reusing concepts of the WebML inside the UML-Guide to provide concept interoperability, and the URL generation technique of the WebML runtime inside the UML-Guide XSL code to provide systems interoperability and the content generation. The state diagrams augmentation does not require a complete mapping between the UML-Guide components and the WebML conceptual primitives. The UML-Guide designers just need to specify IDs of those WebML elements that are used for the URL construction, i.e., pages, source entities from which page contents are extracted, and entity attributes that are used in parametric selectors for retrieving page contents. The resulting application generator can be considered an "adaptive hypermedia generator" in full strength, whose

potential expressive power goes well beyond this experiment.

Complementariness.   The combination of the two methods offers several benefits. Among them the most relevant one is the orthogonality of the adaptation design with respect to the content and the hypertext design. This is particularly useful in the e-Learning domain where, in order to increase the application effectiveness, the provision of LOs by courseware companies must be accompanied by the definition of personalized learning paths or individual curriculum sequences [WB01], based both on local learning strategies as well as on specific user competencies. These two requirements are not easy to identify by companies offering e-Learning services, while they are generally well-contextualized within organizations exploiting the services. The availability of customization extensions therefore enables the latter to customize the application locally, according to their learning goals and the knowledge level of their members. The UML-Guide state diagrams constitute an easy to use specification tool, especially due to the popularity of UML. Also, these diagrams allow definition of learning paths and conditions for state transitions without mastering the complexity of the server-side application design, keeping the specification at a higher level of abstraction. This case study also proves that the adaptive navigation design proposed in this thesis can be used to extend existing methods for Web application engineering even without employing further methods or the whole domain engineering framework.

Software Process.   The case study also provides contribution to the software process of Web applications. Let us describe the process on a scenario. Figure 10.4 depicts the design process for building adaptive applications intermixing the UML-Guide with the WebML specifications and platforms [CDMN05].

The courseware company develops its e-Learning vertical according to well defined top-down steps suggested by the WebML, consisting of designing the data content first, then the hypertext and finally the presentation. Such a process can be paired with the use of the WebRatio tool, which automatically generates the database for storing learning objects and the hypertext composing site views, Web pages, and the content chunks presented to the user.

Then, of course, the courseware company also authors the learning objects in particular domains (e.g. modules about "Programming in Java"). The authoring consists of adding, changing or excluding learning objects and also managing their metadata, like classifying the objects according to topic, title, description, difficulty, background required, related learning objects and so on.

The SMEs start the design of the adaptive guide by collecting knowledge and background data of their employees, and then proceed by selecting bottom-up relevant contents from the body of learning objects that are made available. The SME designers gather information about the employee's skills and needs (e.g., the integration of Java programming and Oracle 9i), and build the adaptive user guides to be used throughout the SME, then they select personalized portions of such guides and install each of them on the employee's client application. This application is able to keep trace of the employees' progresses into the correspondent user model while they perform learning activities.
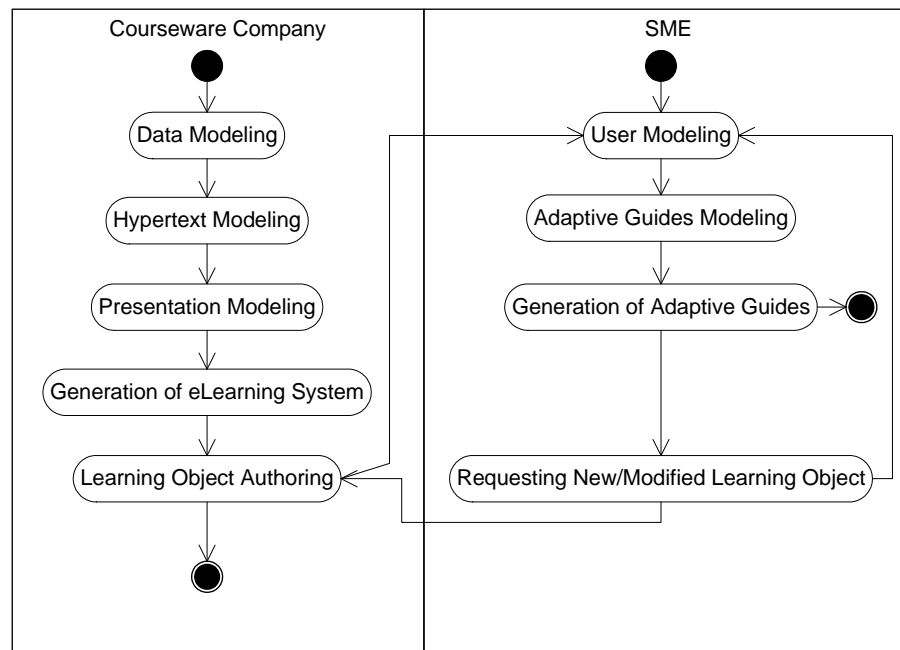
Figure 10.4: Adaptive application design process.

If a SME encounters a need for alteration or additions of new learning object, the request for that activity is submitted to the courseware company. The courseware company alters/adds the new learning object and updates its metadata. The database of the e-Learning vertical is updated at the courseware side and the curricula developers are notified about the new learning objects. In case the SMEs' curricula specifications have to be updated, the generation of user guides is repeated.

This case study also proves the hypothesis given by the Web application modeling framework highlighted in fig. 3.1 used to review currently established Web application model driven development methods. The hypothesis is that the combination of methods into a Web application software process depends a development team, project, and company context. In this case study, we have described a specific context given by a courseware company providing a generic application and SMEs customizing the application to their specific needs where combination of WebML and the UML-Guide suits well. In other contexts, other methods might be suited for integration better.

Chapter 11

Domain Engineering and Adaptive Semantic Web Information Systems

The idea behind the *Semantic Web* [TLHL01] is to endow information on the web with a "well-defined meaning, better enabling computers and people to work in cooperation". The Semantic Web is regarded as a new Web generation which graduates from interlinked HTML pages to networks of objects which can be described according to a subject, object, predicate schema.

The aim of this case study is to show how the web application conceptual models proposed in this thesis relate to the idea of the Semantic Web. The hypotheses behind the study are:

- The identification and separation of different aspects into a number of different domains can be helpful in order to obtain semantic web metadata about information and computation resources.

- The instantiation of the separate domains in resource metadata serves the same purpose as the story collaboration model — to provide a story which is depicted in the resource.

- The variability in feature models which is to be resolved at run-time corresponds to adaptation reasoning rules for semantic web metadata.

- The domain design models guide web service design for adaptive semantic web applications.

11.1   The Model for Semantic Web Metadata

Semantic web technologies like the Resource Description Format (RDF) [LS] or RDF Schema (RDFS) [BG02] provide us with appropriate modeling constructs to model and represent the domain of the resources, the resources themselves, as well as users and links. RDF is used to describe specific resources; an RDF Schema serves to define domain-specific vocabularies for the metadata records represented as RDF descriptions. The following paragraphs summarize the basic principles of semantic web representation formats which we will use to describe vocabularies needed for person-

Figure 11.1: Example of an RDF graph

alized access to web resources. For more information we refer the reader to [Cha01, W3C03].

On the Web, each resource is provided with its own identifier, specified as a *Unified Resource Identifier (URI)* which is globally unique. Descriptions about resources are represented as triples consisting of `subject`, `object`, and `predicate`.

For example, an assertion about the fact that the website of Peter Dolog was authored by Peter Dolog is depicted in figure 11.1. The subject of this triple is `http://www.l3s.de/~dolog`, the predicate is `author` (the dublin core namespace is used to encode the predicate) and the object is `Peter Dolog` as a literal. Object values can be `resources` or `literals`. Literals are strings of text, resources are referenced by URIs. Triples can be embedded in HTML files in an appropriate XML serialization.

Concepts and vocabularies can be provided explicitly on the semantic web and used for these RDF descriptions. The semantic web metadata model distinguishes three types of concepts: *fundamental concepts, schema definition concepts*, and *utility concepts*. Each concept has its own identifier in the form of an URI. The concept definitions are grouped into schemas or namespaces which are identified by URIs as well. It is possible to use abbreviated syntax for the concepts where a namespace is abbreviated into a string and separated from the concept identifier by a colon.

The *fundamental concepts* define the RDF triples, providing *rdf:Resource* as a subject, and *rdf:Property* as predicate. A triple statement can be represented by *rdf:Statement* for the purpose of reification. These concepts are mandatory for all agents which are developed for the semantic web.

The *schema definition concepts* are used to define custom vocabularies to be used with metadata descriptions. These concepts are usually domain specific and will be understood only by the domain specific agents, e.g. web applications for particular purposes. The new vocabulary is defined by means of classes (*rdfs:Class*). The classes can be extended to include properties by defining a domain of properties (*rdfs:domain*); i.e. their inclusion in a particular class. Properties can be further restricted by defining their range of values (*rdfs:range*). Classes and properties can be specialized by using subclassof and subpropertyof predicates (*rdfs:subClassOf* and *rdfs:subPropertyOf*). Any property defines a relation between resources. Subpropertyof defines a subset of the property range. Similarly, the subclassof relation between classes is defined to allow for subset inclusion. Classes define sets of resources of a certain kind. *rdf:type* is used to denote that a resource is an instance of a class or in other words that it belongs to a certain set of resources. Furthermore, the resources types give the resource a meaning in a certain context, defined and constrained by a schema.

The *utility concepts* are additional concepts used to define collections and for deploying RDF vocabulary on the web. Collections can be defined by one of the subclasses of the *rdfs:Container* as a bag (*rdf:Bag*), ordered sequence (*rdf:Seq*), or alter-

natives (*rdf:Alt*). *rdfs:seeAlso* and *rdfs:isDefinedBy* are used to point to alternative descriptions of a resource. *rdfs:label* and *rdfs:comment* are used to add human readable descriptions of a resource.

The Web Ontology Language (OWL) extends RDFS with restrictions on properties, equality between classes and properties, intersection of classes, property characteristics, 0 and 1 cardinality restrictions, and versioning in a light version. OWL Full and DL (relates to description logic) add class axioms, arbitrary cardinality, filler information and boolean combination of class expressions.

11.2 Reasoning on the Semantic Web

Several query and reasoning languages have been introduced to query for and reason with metadata on the semantic web. The semantics of the language are often based on Datalog, as used in the Edutella Query Language (QEL) [NWQ+02, NS03], and extended rule and logic programming languages.

QEL offers a full range of predicates as well as equality, general Datalog rules, and outer join (see [NS03]). An example for a simple QEL query regarding resources is the following:

```
s(X, <dc:title>, Y),
s(X, <dc:subject>, S),
qel:equals(S, <java:OO_Class>).
```

The query tries to find resources where `dc:subject` equals `java:OO_Class`. The prefixes `qel:`, `dc:`, and `java:` are abbreviations for URIs of the schemas used. The variable `X` will be bound to URIs of resources, variable `Y` will be bound to titles of the resources, and variable `S` will be bound to subjects of the resources.

A rule language especially designed for querying and transforming RDF models is TRIPLE [SD02]. Rules defined in TRIPLE can perform reasoning operations in respect of RDF-annotated information resources; translation tools from RDF to TRIPLE and vice versa are provided.

TRIPLE supports *namespaces* by declaring them in clause-like constructs of the form *namespaceabbrev := namespace*; resources can use these namespaces abbreviations.

```
sun_java := "http://java.sun.com/docs/books/tutorial".
```

Statements show a similarity to F-Logic object syntax: An RDF statement (which is a triple) is written as $subject[predicate \rightarrow object]$. Several statements with the same subject can be abbreviated in the following way:

```
sun_java:'index.html'[rdf:type->doc:Document;
  doc:hasDocumentType->doc:StudyMaterial].
```

RDF *models* are explicitly available in TRIPLE: Statements that are true in a specific model are written as "@model", e.g.

```
doc:OO_Class[rdf:type->doc:Concept]@results:simple.
```

The usual connectives and quantifiers for building logical formulae from statements are allowed, i.e. $\land$, $\lor$, $\lnot$, $\forall$, $\exists$, etc. For TRIPLE programs in plain ASCII syntax, the symbols `AND`, `OR`, `NOT`, `FORALL`, `EXISTS`, `<-`, `->`, etc. are used. All variables must be introduced via quantifiers, therefore marking them is not necessary.

11.3   Models and Semantic Web Application Components

The models created as products of the domain engineering process for adaptive web applications can be used on the Semantic Web. The notion of objects as instances of particular classes from an ontology is similar to object orientation principles in software engineering methodologies. Furthermore, the configuration knowledge maintained by the domain feature models can be mapped to adaptation rules which enable decisions to be made about appropriate variants of a feature according to knowledge available about a user.

Figure 11.2 depicts relations between domain engineering models, semantic web ontologies, metadata, adaptation rules and services which collaborate in an application to serve a user. The domain conceptual models correspond to ontologies and schemas; i.e., the application domain model corresponds to the application domain ontology, the environment domain model corresponds to the environment domain ontology and the user domain model corresponds to the user domain ontology. The application uses interfaces to access the ontologies, therefore appropriate interfaces and services to access the ontologies have to be provided (application domain, environment, and user domain ontology services). The services may be parts of frameworks to access the ontologies and their instances. The configuration information of the services in the framework is based on the feature models of the domains.

As with the domain analysis models, the ontologies are instantiated and refined into metadata. The instances in the metadata correspond to the story collaboration models as they describe what a resource is about and how the concepts appear in the resource content. Furthermore, concepts and content sequencing information are also provided within the metadata which correspond to the navigation trail models. The adaptation rules and constraints used in the collaboration models and navigation trail models correspond to the adaptation rules which are used to perform reasoning operations on metadata and content fragments. Both navigation and content composition have to be realized by appropriate services, i.e. a navigation generation service and a content composition service.

The navigation trail and collaboration models may contain side effect actions and messages to user observation components. The event specification which supplements the state diagram model is a source for the observation ontology and its observation service. The observation service generates user metadata instances (which are inserted into a user profile) based on the observation and user domain ontologies. Observation services are very often integrated with user modeling frameworks.
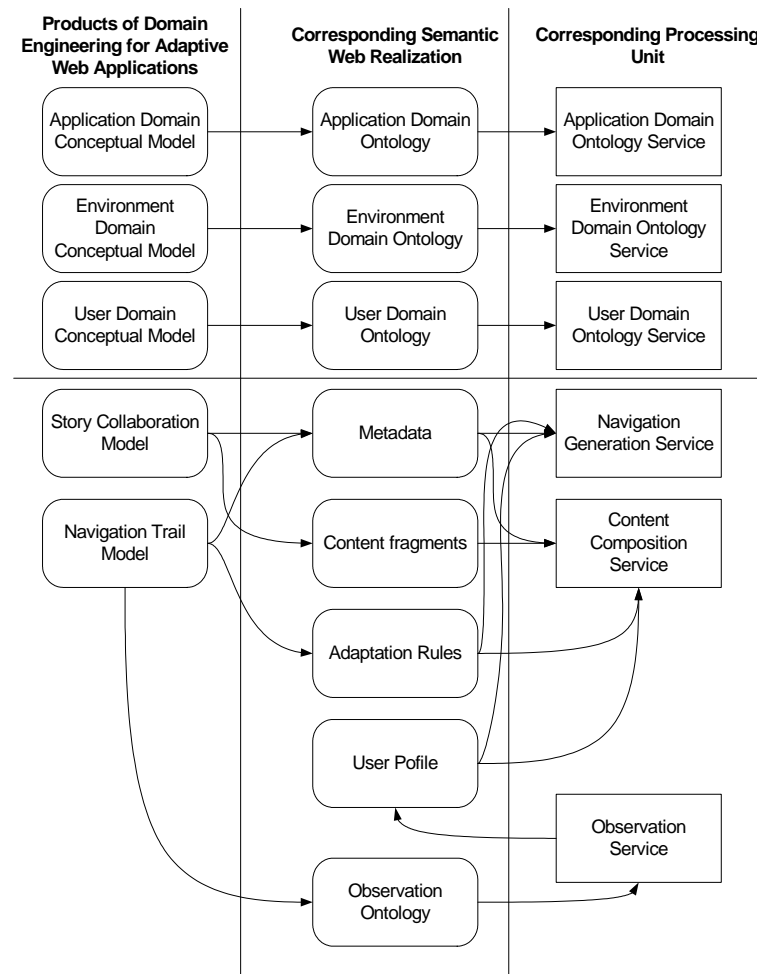
Figure 11.2: Correspondences between the models created in domain engineering process for adaptive web applications and semantic web ontologies, metadata, and application processing services

11.4 Ontologies

As suggested in section 6.1, specific domain information is usually described by concepts and their mutual relationships. The semantic web vocabularies (ontologies) in RDFS or OWL serve the purposes of domain specific models [HDN04]. The domain ontologies are usually described by classes (classifies objects from a domain) and relationships between them in a manner similar to our conceptual models.

The Application Domain. Figure 11.3 depicts an excerpt from the Java programming domain following on from our eLearning examples from part II. We show just a fragment of a domain knowledge base covering Java programming concepts with an `isa` (subConceptOf) relationship between these concepts. Figure 11.3 depicts the `Programming`
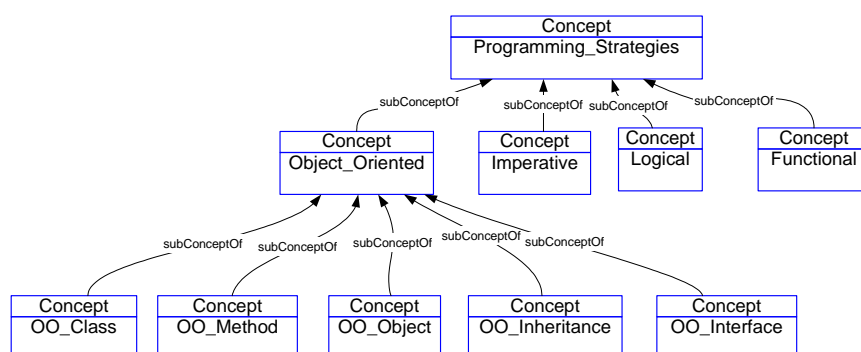`_Strategies` concept with its subconcepts: `Object_Oriented`, `Imperative`, `Lo-`

Figure 11.3: An excerpt from an application domain ontology for a Java e-lecture

gical, and Functional. OO_Class, OO_Method, OO_Object, OO_Inheritance, and OO_Interface are depicted as subconcepts of Object_Oriented. Other relations between concepts might be useful for personalization purposes as well, e.g., sequencing or dependency relations.
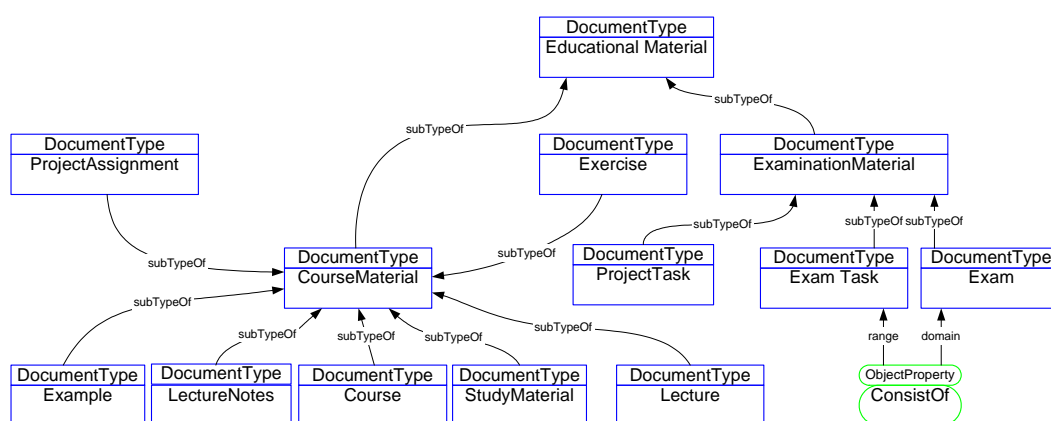
Figure 11.4: An excerpt from an environment ontology as a document types hierarchy for eLearning applications

The Environment Domain. Figure 11.4 depicts the ontology for document types in an educational domain. Educational Material as the most general document type has two subtypes: Course Material and Examination Material. Examination Material can be further specialized to Project Task, Exam Task and Exam. The Exam can consist of the Exam Task-s. Course Material can be further specialized into Lecture, Example, LectureNote, Course, Exercise and Project Assignment.

An ontology for documents and their relationships to other components is depicted in fig. 11.5. The ontology represents a context of learning material which is usually provided as a document. The class Document is used to annotate a resource which is a document. The documents describe some sort of concepts. We use the class Concept to annotate concepts. Concepts and documents are related through

Figure 11.5: An excerpt from an environment domain ontology for documents

the `hasKeyword` property.

Documents can be ordered by means of the `hasPrerequisite` property. Multiplicity is allowed to provide for different ordering per and within an application. The hasPrerequisite property 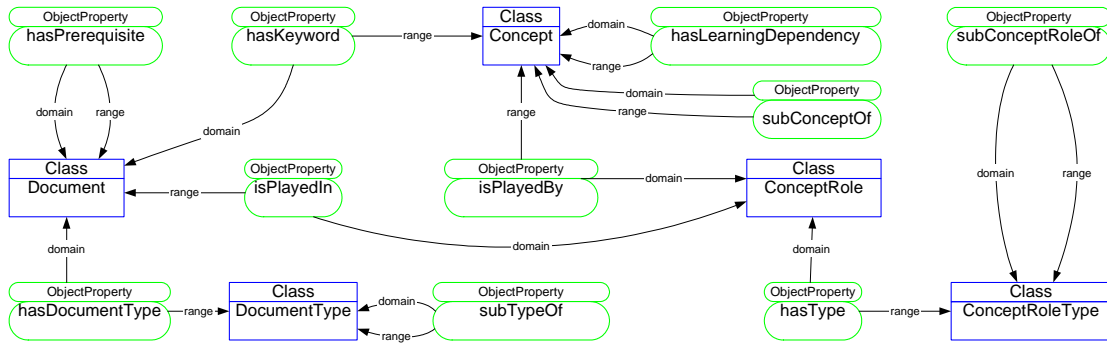is intended for navigation purposes. The ordering relation values are derived from the *state machines* which are used for *navigation modeling* in our approach.

Concepts and documents have certain roles to play in their mutual collaboration. This reflects the idea of story collaborations modeled in our conceptual modeling approach. The concrete instances are derived from the *collaboration models*. In the ontology, we represent these facts by instances of the `ConceptRole` class and its two properties: `isPlayedIn` and `isPlayedBy`. Document properties can be extended further by assigning a `DocumentType`. Similarly, the roles can be extended by specifying their types. Concepts, concept role types, and document types can form hierarchies. We define `subTypeOf`, `subConceptRoleOf`, and `subConceptOf` properties for these purposes.

The User Domain. Data about a user serves the purpose of deriving contextual structures. It is used to determine how to adapt the presentation of hypertext structures. Here we define an ontology for a user profile based on IEEE Personal and Private Information (PAPI) [IEE]. PAPI distinguishes between *personal, relations, security, preference, performance* and *portfolio* information. The *personal* category contains information about a user's names, contacts and addresses. The *Relations* category serves as a category for specifying relationships between users (e.g. classmate, teacherIs, teacherOf, instructorIs, instructorOf, belongsTo, belongsWith). *Security* aims to provide slots for credentials and access rights. *Preference* indicates the types of devices and objects which the user is able to recognize. *Performance* is for storing information about the measured performance of a user obtained via learning material (i.e. what a user knows). *Portfolio* is for accessing the previous experience of a user. Each category can be extended. For more discussion on learner modeling standards see for example [DS05].

Figure 11.6 depicts an example of an ontology for a learner profile. The ontology is based on the *performance* category of `PAPI`. We are storing sentences about a learner who has achieved a certain `Performance`. The `Performance` is based on
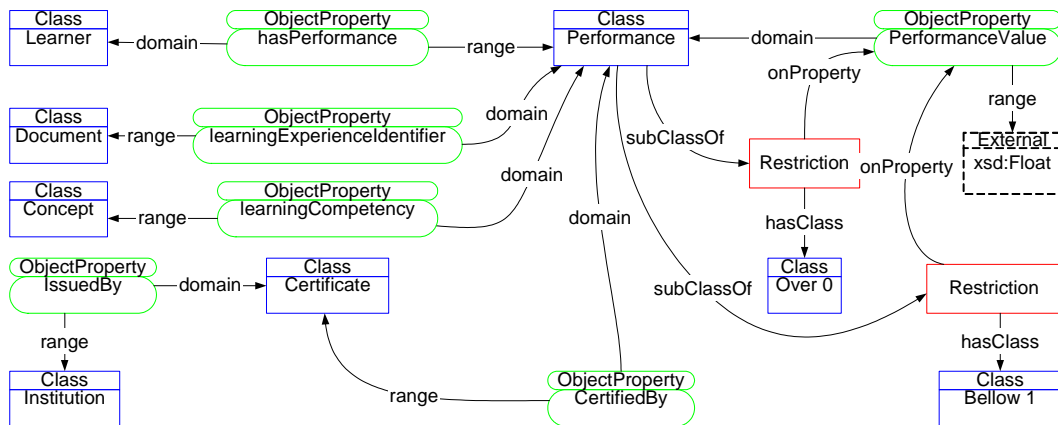
Figure 11.6: Ontology for learner performance

learning experience (`learningExperienceIdentifier`) which is obtained from a particular document. The experience implies that a `Concept` has been learned. This is maintained by `learningCompetency` property. The `Performance` is certified by a `Certificate`, which is issued by a certain `Institution`. The `Performance` has a certain `PerformanceValue`, which in this context is defined as a floating point number and restricted to the interval from 0 to 1.

Observations.   During runtime, users interact with a web system. The user's interactions can be used to draw conclusions about possible user interests, about his or her goal, task, knowledge, etc. The facts gathered about a user can be used for providing personalized views on hypertexts. An ontology of observations should therefore provide a structure of information about possible user observations, and - if applicable - their relations and/or dependencies.

A simple ontology for observations is depicted in fig. 11.7. The ontology allows us to instantiate facts as follows: that a `Learner` has interacted (`hasInteraction` property) with a particular `Document` (`isAbout` property) via an interaction of a specific type (`InteractionType`). Examples of `InteractionTypes` are `access`, `bookmark`, `an-`
`notate`. The information that an interaction has taken place during a time interval is maintained by the `beginTime` and `endTime` properties. The `ObservationLevel` describes a particular activity type which was the purpose of the interaction. Possible instances of ObservationLevel are that a user has `visited` a page, has `worked` on a project, has `solved` some exercise, etc.

11.5   Metadata

The ontologies described above are used in annotations of concrete information resources. The annotation metadata serves as a knowledge base about domain information, collaborations and navigation which involve particular resources.
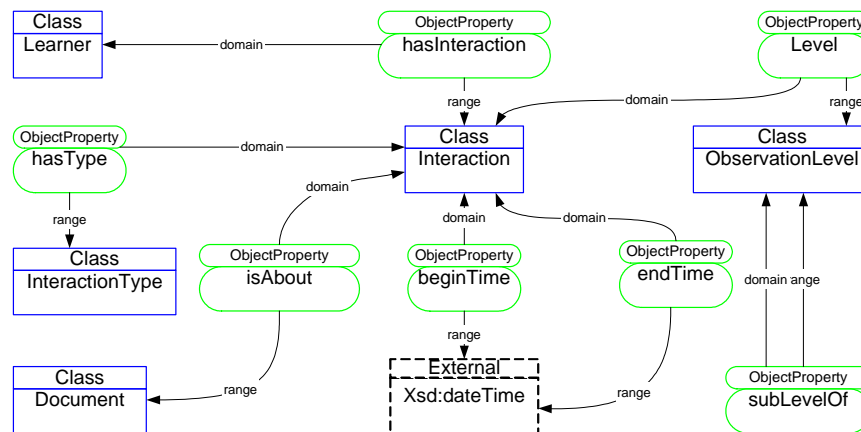
Figure 11.7: Ontology for observations

Story Collaborations. The feature collaborations in a particular resource represent concrete content realization or composition in that resource. An example of such a resource could be a page describing sun_java:'java/ concepts/class.html'.

The following example shows how such a page can be annotated by means of implementing ontologies.

```
sun_java:'java/concepts/class.html'[rdf:type->doc:Document;
  hasTopic->doc:OO_Class].
doc:OO_Class[rdf:type->doc:Concept;
  doc:subConceptOf->doc:Classes_and_objects].
doc:ClassesIntroduction[rdf:type->doc:ConceptRole;
  doc:isPlayedBy->doc:OO_Class;
  doc:isPlayedIn->sun_java:'java/concepts/class.html';
  doc:hasType->doc:Introduction].
doc:Introduction[rdf:Type->doc:ConceptRoleType;
  doc:subConceptRoleOf->doc:Cover].
```

The page is a document (RDF type `Document`). The type prescribes through which environment the documents can be accessed. The document describes information about classes (`OO_Class` concept). The `OO_Class` concept is annotated with the type `Concept` and is a subconcept of the `Classes_and_objects` concept.

The collaboration knowledge is represented by the `ClassesIntroduction` resource which is of type `ConceptRole`. The `OO_Class` concept plays an introductory role (the `Introduction` role type) in the document which is annotated by using the properties isPlayedBy and isPlayedIn respectively, invoking references to the `OO_Class` concept and the document . The `Introduction` is of type `ConceptRole-Type` and means that the concept is covered in the content to a certain extent. Therefore, the `Introduction` is a subtype of the `Cover` concept role type — a generic role type used for stating that a concept is covered in a document content.

Navigation. The navigation relations are derived from the navigation trails specified by state diagrams. Each transition is transformed into a `hasPrerequisite`

(or inverse property `isPrerequisiteFor`) of a concept or resource depending on what is used as a reference for identification of navigation trails. In our example, the `OO_Class` concept is a prerequisite for the `OO_Inheritance`. Thus the example mentioned above is extended using the instance of this property.

```
sun_java:'java/concepts/class.html'[rdf:type->doc:Document;
  hasTopic->doc:OO_Class].
doc:OO_Class[rdf:type->doc:Concept;
  doc:subConceptOf->doc:Classes_and_objects;
    doc:isPrerequisiteFor->doc:OO_Inheritance].
doc:ClassesIntroduction[rdf:type->doc:ConceptRole;
  doc:isPlayedBy->doc:OO_Class;
  doc:isPlayedIn->sun_java:'java/concepts/class.html';
  doc:hasType->doc:Introduction].
doc:Introduction[rdf:Type->doc:ConceptRoleType;
  doc:subConceptRoleOf->doc:Cover].
```

Runtime User Model. To be able to personalize access to resources, a run-time user model has to be maintained. The run- time user model instantiates the user domain model selected for a particular application. An example of a simple learner profile could look like this:

```
user:user2[rdf:type -> learner:Learner;
  learner:hasPerformance -> user:user2P].
user:user2P[rdf:type->learner:Performance;
  learner:learningExperienceIdentifier->
    sun_java:'java/concepts/object.html';
  learner:learningCompetency->doc:OO_Object;
  learner:CertifiedBy->KBScerturi:C1X5TZ3;
  learner:PerformanceValue->0.9].
KBScerturi:C1X5TZ3[rdf:type->learner:Certificate;
  learner:IssuedBy->KBSuri:KBS].
KBSuri:KBS[rdf:type->learner:Institution].
```

The learner `user2` has the performance record (`user2P`). The performance contains a learning experience about the KBS Java objects resource. The concept covered in the resource is stored in the performance as well. Then a certificate about the performance containing the performance value and the institution which issued the certificate is stored in the learner performance as well.

11.6 Services

The metadata described above is used by services which collaborate in the web application to serve a user. Figure 11.8 depicts a UML collaboration diagram showing a message flow between service providers which we have implemented for a personal learning assistant. Boxes represent service providers, lines represent links between the providers. The direction of a message or invoking operation is indicated by a small arrow on top of a line with the name and parameters of that operation. We use two kinds of arrows in fig. 11.8. The normal arrow ($\rightarrow$) is used to indicate a plain message.
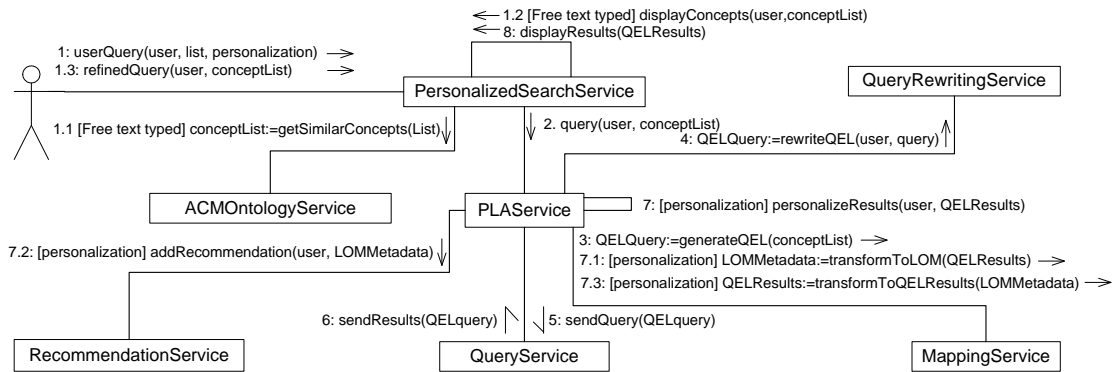
Figure 11.8: A collaboration diagram from a current implementation.

The "harpoon" ($\rightharpoonup$) indicates explicitly that a message is asynchronous. Square brackets are used to indicate a condition which determines whether a certain message is to be passed: If the condition is not satisfied then the message is not sent.

The `PersonalizedSearchService` provides a user interface for searching and displaying personalized results to a user. A user can send two messages through the user interface provided. First the message (`userQuery`) notifies the `Personalized-SearchService` about the user, text typed in fields or concepts selected from the ACM classification hierarchy, and whether to provide personalization information or not. If the user typed free text into fields which are provided at the user interface, the `PersonalizedSearchService` contacts an ontology service (in our case the `ACMOntologyService`) to retrieve concepts similar to the text typed (the message `getSimilarConcepts`). The `PersonalizedSearchService` then displays these concepts to a user to refine his/her query. After selecting precise concepts from suggested entries from the ontology, the user can send a refined request to the `PersonalizedSearchService`.

The `PersonalizedSearchService` notifies the `PLAService` about the user query (the `query` message). The `PLAService` first makes use of the `MappingService` provider to generate a QEL query by sending the `generateQEL` message. The service constructs an appropriate QEL query from the concepts list. In addition, the `PLAService` contacts the `QueryRewritingService` provider after receiving the `QELQuery` to rewrite the `QELQuery` according to a learner profile, adding additional constraints to the `QELQuery`.

The `PLAService` sends a message with the rewritten `QELQuery` to a `QueryService`, in our case the Edutella query service which propagates the query into the Edutella P2P resource provision network. The Edutella `QueryService` returns all query results.

If the learner prefers recommendation information to be included with the query results, the PLAService contacts the `RecommendationService` to derive such recommendation information according to the learner profile or to group profiles (collaborative recommendation). When such personalized results are available, the `PLAService` notifies the `PersonalizedSearchService` to display the results to a learner.

Personalization Services. The personalization services are services which decide about variable features of web applications at runtime based on user profiles. The adaptation decisions are specified as reasoning rules in TRIPLE which is an extension of the horn logic based on the XSB prolog with namespaces and views.

A *query rewriting service* adds additional constraints to a QEL query created according to which concepts a user selected. These constraints reflect concepts and language preferences maintained in user profiles. They restrict a search space in the metadata and ensure that only resources which are relevant for a user are selected. The example below is based on a language feature variation point which specifies alternative language features of a resource.

We illustrate the query rewriting principle using the following simple restriction profile, implemented in TRIPLE.

```
@edu:p1 {
   edu:add1[rdf:type -> edu:AddSimpleRestriction;
       rdf:predicate -> dc:lang;
       rdf:object -> lang:de].
   edu:add2[rdf:type -> edu:AddTopicRestriction;
           edu:addTopic -> acmccs:'D.1.5'].}
```

This heuristic is used to extend a QEL query with a constraint which restricts the results to learning resources in the German language (restriction `edu:add1`).

Another restriction derived from the user profile is a restriction on resources about *object-oriented programming* (`edu:add2`). The ACM Computer Classification System [oCm02] is used to encode the subject mentioned. In this classification system, *object-oriented programming* can be found in the category `D` which covers *software*. The subcategory `D.1` covers *programming techniques*, with the fifth subcategory being *object-oriented programming*. Heuristics for query rewriting are usually more complex, especially in the case of concept or subject restrictions. They depend on concepts being selected or typed as a user query.

The derived restrictions profile is used in a TRIPLE view which takes the profile and QEL query model as input. One of the rules for reasoning processes in respect of language restrictions profiles is illustrated below. The view `@edu:p1` encapsulates the restrictions model.

```
FORALL QUERY, VAR, PRED, OBJ, NEWLIT
 QUERY[edu:hasQueryLiteral -> edu:NEWLIT] AND
 edu:NEWLIT[rdf:type -> edu:RDFReifiedStatement;
           rdf:subject -> VAR;
           rdf:predicate -> PRED;
           rdf:object -> OBJ]
 <-
 EXISTS LITERAL, ANY (
   QUERY[rdf:type -> edu:QEL3Query;
         edu:hasQueryLiteral -> LITERAL]
   AND
   LITERAL[rdf:type -> edu:RDFReifiedStatement;
           rdf:subject ->
               VAR[rdf:type -> edu:Variable];
```

```
              rdf:predicate -> dc:ANY])
  AND
  EXISTS A
    A[rdf:type -> edu:AddSimpleRestriction;
      rdf:predicate -> PRED;
      rdf:object -> OBJ]@edu:p1
  AND
  unify(NEWLIT, lit(VAR,PRED,OBJ)).
```

The *recommendation service* provides the following functionality: It can annotate learning resources for a user according to their educational state. For instance, it can *recommend* a resource to a specific user, or give a less strong recommendation like *might be understandable.* Furthermore, it can *not recommend* a learning resource or point out that this learning resource leads to a page that the user has already visited. The annotation property again reflects a variation point where resource metadata can vary and the variation is resolved according to an adaptation rule.

The appropriate recommendation annotations are derived according to the degree to which a user mastered prerequisite concepts for a learning resource. The `lr:isPrerequisiteFor` relationships of concepts covered in a learning resource are analyzed and compared to a user performance profile and competencies acquired.

One example of a recommendation rule is a rule which determines learning resources which are `Recommended`. A learning resource is recommended if *all* prerequisite concepts of all those concepts which it covers have been mastered by a user:

```
FORALL LR,U learning_state(LR, U, Recommended) <-
  learning_resource(LR) AND user(U)
  AND NOT learning_state(LR, U, Already_visited)
  AND FORALL Ck ( prerequisite_concepts(LR, Ck) ->
        p_obs(Ck, U, Learned) ).
```

Predicates used in the rule derive concepts like learning resource, concepts, users, observations and learning states from metadata based on types taken from the ontologies described in section 11.4.

We have implemented other rules to compute less strong recommendations. This includes, for instance, a recommendation that a resource `Might_be_understandable` if at least one prerequisite concept has been learned.

This kind of recommendation can be used for example as a link annotation technique in the area of adaptive hypermedia, or to annotate query results with the recommendation information. On the user interface side, it is often implemented using the traffic lights already mentioned.

Link Generation, or in other words a *Navigation Generation Service* connects a learning resource to other learning resources, or it connects a learning resource to a context, e.g. within a course with links to previous and next steps. As an example of a Link Generation Service, we have implemented a service that relates a learning resource to other resources which provide related *examples* of the learning resource's content.

One example for deriving such an example-relation for a resource `R` involves ensuring that each concept on `R` is covered by the example `E`:

```
FORALL R, E example(R,E) <-
  LearningResource(R) AND example(E) AND
  EXISTS C1 (R[dc:subject->C1]) AND
  FORALL C2 (R[dc:subject->C2]->E[dc:subject->C2]).
```

The optionality of the resource is made dependent on similarity between concepts. If they are similar in concept, then the feature is considered; if they are not, the feature is suppressed. The second line in the rule above ensures that `R` is a `Learning-Resource` and `E` is an `Example` (using the ontology for learning resources described in section 11.4). The third rule verifies that `R` really is about some concept - i.e. there exists a metadata annotation like `dc:subject`. The fourth line then expresses what our rule should check: Whether each concept in `R` will be explained in the example `E`.

The example feature can be expanded further and considered as a variation point. One of the options in the variation point might be a best_example. A user profile can be taken into account when generating the example relationship. A personalized *pedagogical* recommendation of an example might include an example showing new things to learn in a context of concepts which are already known or have been newly learned: This would embed the concepts to learn in the previous learning experience of a user. The rule for deriving this *best_example* follows.

```
FORALL R, E, U best_example(R,E,U) <-
  LearningResource(R) AND example(E) AND user(U)
  AND example(R,E) AND FORALL C (
  (E[dc:subject->C] AND NOT R[dc:subject->C])->
           p_obs(C, U, Learned) ).
```

Further rules for generating personalized hypertext associations can be implemented. Other relationships, classes and properties from the domain, user, and learning resource ontology can be used for these purposes [DHN03]. The `isa` relationship in the concept-ontology of the Java application domain can be utilized to recommend learning resources covering either more general concepts, e.g. introducing a concept of programming strategies, or more specific ones. The sequencing relationship can be used to recommend learning resources in the following way: A resource which describes a concept (the concept appears in the `dc:subject` property for the resource) from the beginning of the sequence will be recommended earlier than a resource which describes a concept from the end of such a sequence. A dependency relationship referring to whether a concept depends on another concept can be used as well to recommend learning resources which describe dependent concepts together with a learning resource describing a concept which was recommended by another rule.

Supporting Services.   The Edutella P2P infrastructure [NWQ[+]02] allows us to connect peers which provide RDF metadata about resources. Edutella also provides us with a powerful Datalog-based query language, RDF-QEL, provided within a *Query Service*. A query can be formulated in RDF format as well, and it can reference several schemas. An example for a simple query about resources is the following:

```
s(X, <dc:title>, Y),
s(X, <dc:subject>, S),
qel:equals(S, <java:OO_Class>).
```

The query tries to find resources where `dc:subject` equals to `java:OO_Class`. The prefixes `qel:`, `dc:`, and `java:` are abbreviations for URIs of the schemas used. Variable `X` will be bound to URIs of resources, variable `Y` will be bound to titles of the resources, and variable `S` will be bound to subjects of the resources.

We have implemented a *mapping service* for mapping QEL variable bindings to LOM RDF bindings and vice versa. This was needed because our recommendation service accepts input in LOM RDF bindings. On the other hand, additional recommendation information plus LOM metadata have to be transformed back to QEL variable bindings because the personalized search service uses QEL variable bindings as a result set. These transformations are again performed in TRIPLE.

Concept mappings between different subject ontologies, different ontologies for describing learners, and different learning resource ontologies are important as well. The TRIPLE view/model mechanism allow us to specify and implement models which embed rules for mappings between those ontologies [MNZS03]. Currently we are implementing such mapping heuristics between the ontologies used in different systems connected within the ELENA network.

We have implemented a simple version of an *ontology service* for the ACM classification system and its RDF LOM bindings. The current version of our ontology service supports requests for getting the whole ontology using the HTTP protocol as well as requests for deriving concepts from the ontology which are "similar" to the submitted text string.

11.7   Applications

Personal Learning Assistant (PLA).   The purpose of the PLA [DHNS04b] is to connect and integrate the services which are needed to perform the learning support task. *Personalized Search* for example connects mapping, query rewriting, query and recommendation services.

Figure 11.9 depicts a user interface for formulating a user query to search for a particular concept or competence a user would like to acquire, combined with a user interface providing results with recommendations represented by the traffic light metaphor. Using this metaphor, a green ball indicates recommended learning resources, a red ball indicates non-recommended learning resources and a yellow ball indicates partially recommended learning resources.

The user interface is generated by a service which uses the ontology service chosen (the ACM ontology service). A list of learners who have a learner profile maintained at the PLA service chosen is displayed as well.

Users can type free text into three available fields or they can select concepts from one of the ontologies provided (in our example figure the user typed "intelli agent").

The user interface returning the results is generated according to the concepts chosen and includes the query results returned by the query service and personalized by the recommendation services chosen through the PLA service. The personal recommendation is depicted in the first column (PReco). The second column (labeled as "Reco") provides learners with a group-based recommendation.
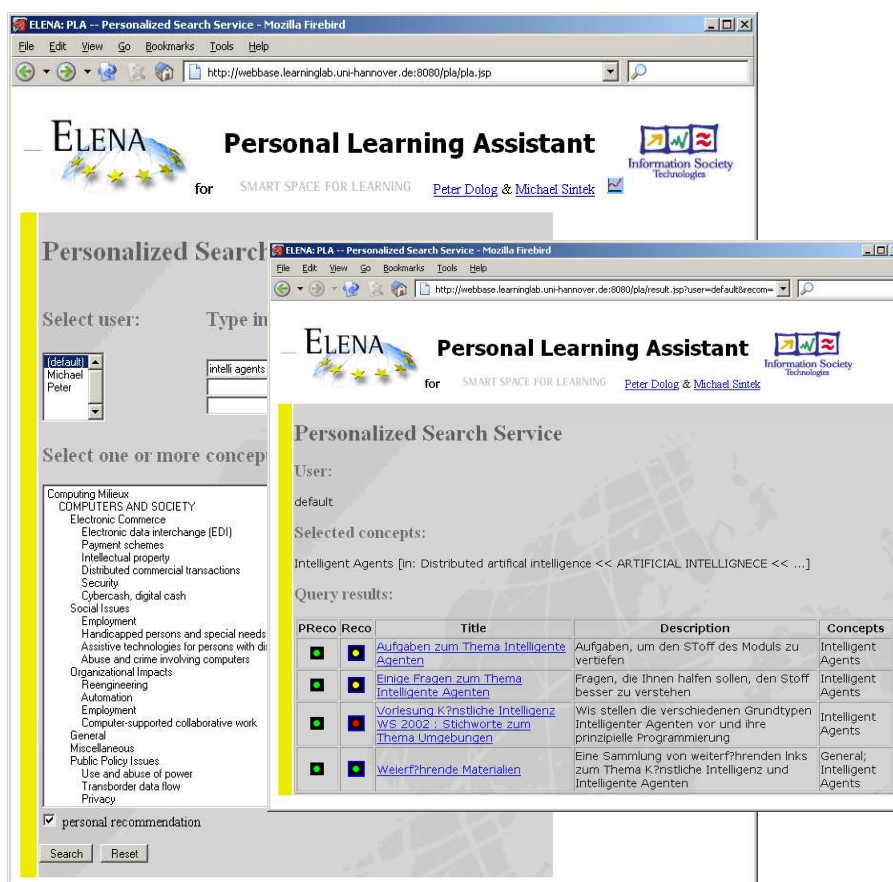
Figure 11.9: A prototype user interface for search operations.

The Personal Reader.   We have experimented with the annotation instances within the personal reader framework [DHNS04a]. The personal reader enables the learner to work with learning resources in an embedding context generated according to the metadata described above. In the local context, more details related to the topics of the learning resource, the general topics the learner is currently studying, examples, summaries, quizzes, etc. are generated and enriched with personal recommendations according to the learner's current learning state, as shown in Fig. 11.10.

Link generation in the framework can be configured to take several properties into account. One of them is the hasPrerequisite relation mentioned previously. However, other relations might be useful as well. The isa/subclassOf hierarchy of the Java ontology can be taken into account to generate more detailed or more general resources on a particular topic.

11.8   Lessons Learned

This case study description was created in the context of the Elena project (`http://www.elena-project.org`) and it represents part of the much larger task of creating semantic models and a service network for smart spaces for learning. Final schemas and services are documented in the project documentation [DNe05, DNe04].
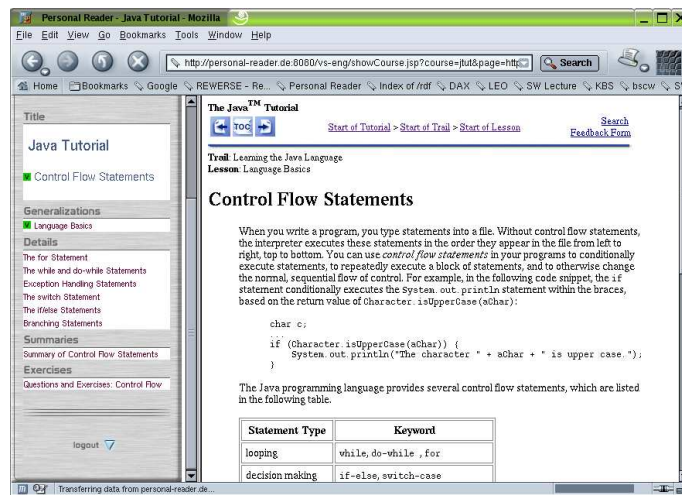
Figure 11.10: Screenshot of the Personal Reader, showing the adaptive context of a learning resource in a course.

Variability.   We discovered that there are other dimensions of variability in such information spaces in addition to the user-centered variability. This includes variability between conceptual models employed by the connected systems, conceptual models of exchange models between the systems, and requirements relating to user applications such as different personal learning assistants. The feature models have been very helpful aids in the task of analyzing existing information models of standards in eLearning domains, user requirements relating to the way search processes and queries for learning resources should be performed and conceptual schemas of connected systems. Furthermore, the natural variability (known as heterogeneity in the semantic web community) led us to study further non-user adaptation services such as mappings between different vocabularies.

Representation.   This case study shows that semantic web representation is very close to the domain engineering models. We have shown correspondences between ontologies, conceptual models, and metadata and navigation trails, and content compositions. In addition, we have shown that the semantic web representation is very well suited to the task of considering variable features of content and navigation on the web. Selected variable and optional features are examided according to adaptation rules embedded in application services which decide whether to present them to a particular user or not based on his or her background. Furthermore, the case study has shown that the models aid the design of search processes which enable a user to find resources more effectively, because they can exploit and expose information about story collaborations and links to further relevant resources which can be compared with the user's history, preferences and goals.

Software Process.   The case study has also shown that the domain engineering process for adaptive Web applications proposed in this thesis is feasible. The application family consists currently of two applications — the Personal Reader and Personal

Learning Assistant, developed according to the guidelines presented in part II. The conceptual models and its instances have been used in the development process to reason about information provided in the applications, the environment which supports its provision, adaptation strategies, user models which give rise to them and services which support and realize them. Furthermore, the instances of the models are also used by the final application for presentation and navigation purposes as well as decision making in respect of adaptation requirements.

Chapter 12

Summary

This part has provided three case studies showing different aspects of the domain engineering framework for adaptive Web applications.

The first case study has shown how the state diagrams together with the adaptation rules employing the concepts from the class diagrams of user domains can be utilized for generating an adaptive navigation map for a Web-based application. The generator can be considered as a domain implementation for an adaptive navigation support with features of an adaptive content selection. It has shown that besides the modeled adaptivity in the navigation model, the generator itself can be adaptive and parametrized for different presentation options such as a whole map, a local context, or a guide panel. The case study has shown, that the models created as the products of the domain engineering framework for adaptive Web applications proposed in this thesis are also useful for generating purposes besides their documentation, communication, and reasoning purpose.

Furthermore, the second case study has shown that the adaptive navigation trails specified by the state diagrams can be integrated with existing Web application engineering methods like the WebML. The interoperability between the specifications created according to such methods and the state diagrams approach is realized by tagged values; i.e., concepts from the models created according to the other methods are encoded as tagged values in the UML state diagrams. Existing generator can be easily modified to include additional steps to transform the tagged values to run time references such as links and queries. Besides these domain implementation features, the process of the adaptive navigation trail design has been successfully integrated into the WebML method which has shown complementariness of the adaptive navigation specifications to the existing methods.

The third case study has shown a comprehensive guide through the instance of the domain engineering process for adaptive eLearning semantic Web applications. It shows correspondences between the models and the semantic Web implementation technologies, the rules to reason on top of the semantic Web metadata as counterpart to variation points and optionality resolution, and Web services interacting in the semantic Web applications. Two applications are build within the family, the personalized search service of a personal learning assistant and the personal reader. Both applications share the application domain, the environment, and the user domain features. They also share the adaptation rules to resolve variation at runtime

and services to support the final user-adapted application. They differ in the realized function, PLA is used to search for learning resources while the personal reader is used to guide users through the resources.

Part IV

Outlook

Chapter 13

Conclusions

We have proposed a domain engineering framework for adaptive Web applications in this thesis. We have successfully applied it in three different cases focusing on three different aspects of the framework.

13.1   Contributions

The main contribution of the thesis is an adoption of feature models and their variability and commonality modeling for the purpose of adaptation. Adaptation means a selection of appropriate variants and their combinations for a user, based on his user profile. The profile can be considered as requirements profile or as a dynamic evolving user model observed during runtime of a Web application. Therefore, the variability and commonality notion in feature modeling is a straightforward technique for modeling the variants and their combinations for adaptive applications. This has not been up to our knowledge proposed by anybody so far.

Another contribution of this thesis is a separation of three domains (application, environment, and user) within domain analysis. Though the domains are used in connection to each other in the final applications, the separation has advantages especially in the development process. One reason for separating the application domain and environment domain is the fact that the standardized browsers do not support delivery and structuring environments like dialogs, function menus, control panels, or eLearning related course and module access environments. While such environments are usually provided as frameworks for programming other software applications, they have to be designed and developed for Web applications separately. Another reason for having the two domains separated is that the content can be delivered in different environments, so the separation allows for later integration according to the requirements of specific customer. The proposed separations help analysts and designers to reason about both domains separately by using appropriate techniques proposed by this thesis. The reviewed methods do not consider this separation which results in a raising complexity of models when developing adaptive applications and applications where information fragments are considered to be delivered through several environments.

The proposed framework also considers the run time variability resolution which

can be specified in state diagrams and collaboration diagrams. This is a contribution to a conceptualization of adaptation at run time. The adaptation is considered in the content composition and bindings to the environment by means of the messages which are constrained by the features from user profile. The constrains either allow some features to be accessed in the runtime or they disable the access. Adaptive navigation specification is supported by state diagrams which model the navigation trails for users. The variable transitions and alternative states are realization of the variable features and the variation points. They are constrained by the guards which specify under which conditions transitions are allowed and which from alternative substates to chose. The guards again use the features from a user profile. Up to our knowledge, this was not realized before.

We have also shown that the models can be utilized for adaptation purposes in implementation in our case studies. The state diagrams are utilized for the generation of the adaptive navigation guides on the Web and in the integration with the WebML platform by employing the widely accepted W3C XML standard and the processing technologies for the XML documents. Furthermore, the domain, the collaboration, and the state diagram models correspond directly to the semantic Web ontologies, metadata, and services realized in the W3C standards like RDF or OWL. The adaptation rules are derived from variation points of the feature models. We have also developed an extension of the UML class diagrams for feature models used in the domain engineering framework proposed within this thesis.

13.2   Wider Implications

The proposed domain engineering framework for adaptive Web applications fits to the generic domain engineering frameworks. The basic activities like the domain analysis, the domain design, and the domain implementation stay untacked. The framework proposed in this thesis suggest different internal activities within the generic phases. This better reflects peculiarities and requirements of the World Wide Web environment.

The framework activities are self-contained enough to be taken as components and integrated with other methods. We have shown an example of such an integration on the adaptive navigation trail modeling with the WebML. Similarly, the feature modeling can be approached by other methods as well, as it has a clear purpose: configuration knowledge about domain features.

We have also shown that the conceptual approach and separation of concerns adopted in the framework can help also to organize semantic Web metadata about Web resources in a way which improves search results and automated reasoning on the adaptation in Web applications, as the information provided by the models is richer; i.e., the models provide more information needed for decision purposes.

Chapter 14

Further Directions

The domain engineering framework presented in this thesis should further supported by more tools which realize refinements between its models. Semi-automatic solutions for some of the refinements would be very helpful in the design process. It would also be interesting to provide a semiautomatic re-engineering methods to construct the models from existing Web applications.

The domain specific languages for domains which we have considered should be designed too to allow domain experts to be more involved in the construction of such models. For example eLearning experts on specific area would not be able to use the raw UML.

Several dimensions of variability in Web applications should be studied as well. For example in the more open Web environment, there is at least one additional variability given by the different conceptual models and metadata provided on the Web. If more dimensions of the variability would be considered, a problem of conflicting variation points might arise.

We have studied the models created within the framework proposed in this thesis in the context of information and its description on the semantic Web. Further studies on computing descriptions as the collaboration models and state diagrams would bring light whether they might be helpful for semantic Web services and their location, retrieval, composition, and orchestration. We have made some prospects regarding to the state machines and conceptual models in that area already in [Dol04]. However, further investigations are needed.

Finally, more generators in the domain implementation are needed to supplement the ones developed within this thesis. The additional generators would provide more options to select from and to choose the most appropriate one for different applications in an application family.

Bibliography

[AB91]   Serge Abiteboul and Anthony Bonner. Objects and views. In J. Clifford
           and R. King, editors, *Proc. of the ACM SIGMOD International Confer-
           ence on Management of Data*, pages 238–247, San Francisco, California,
           March 1991. ACM Press.

[ABGK02]   Colin Atkinson, Christian Bunse, Hans-Gerhard Groß, and Thomas
           Kühne. Towards a general component model for web-based applica-
           tions. *Ann. Software Eng.*, 13(1-4), 2002.

[ADN⁺03]   Heidrun Allert, Peter Dolog, Wolfgang Nejdl, Wolf Siberski, and
           Friedrich Steimann. Role-oriented models for hypermedia construc-
           tion — conceptual modeling for the semantic web. Technical report,
           Learninglab Lower Saxony, University of Hannover, February 2003.

[AG00]   Liliana Ardissono and Anna Goy. Tailoring the interaction with users in
           web stores. *User Model. User-Adapt. Interact.*, 10(4):251–303, 2000.

[AGP⁺03]   Liliana Ardissono, Anna Goy, Giovanna Petrone, Marino Segnan, and
           Pietro Torasso. Intrigue: Personalized recommendation of tourist at-
           tractions for desktop and hand held devices. *Applied Artificial Intelli-
           gence*, 17(8-9):687–714, 2003.

[arg]   ArgoUML CASE tool web page. Available at: http://www.argouml.org/.

[AT97]   Fabio A. Asnicar and Carlo Tasso. ifweb: a prototype of user model-
           based intelligent agent for document filtering and navigation in the
           world wide web. In *Workshop on Adaptive Systems and User Modeling on
           the World Wide Web at Sixth International Conference on User Modeling,
           Chia Laguna, Sardinia*, 1997.

[Atk01]   Colin Atkinson. *Component-Based Product Line Engineering with UML*.
           The Component Software Series. Addison Wesley, November 2001.

[BCC⁺03]   Marco Brambilla, Stefano Ceri, Sara Comai, Piero Fraternali, and Ioana
           Manolescu. Specification and design of workflow-driven hypertexts.
           Journal of Web Engineering, 1(2):163–182, April 2003.

[BES98]   Peter Brusilovsky, John Eklund, and Elmar Schwarz. Web-based edu-
           cation for all: A tool for developing adaptive courseware. *Computer
           Networks and ISDN Systems (Proceedings of Seventh International World
           Wide Web Conference)*, 30(1–7):291–300, April 1998.

[BFK⁺99]   Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, and Jean-Marc DeBaud. Pulse: A methodology to develop software product lines. In *SSR*, pages 122–131, 1999.

[BG02]   D. Brickley and R. V. Guha. Resource Description Framework (RDF) Schema Specification 1.0, 2002. `http://www.w3.org/TR/rdf-schema`.

[BGdPL⁺03] Felix Bachmann, Michael Goedicke, Julio Cesar Sampaio do Prado Leite, Robert L. Nord, Klaus Pohl, Balasubramaniam Ramesh, and Alexander Vilbig. A meta-model for representing variability in product family development. In Frank van der Linden, editor, *Software Product-Family Engineering, 5th International Workshop, PFE 2003*, volume 3014 of *Lecture Notes in Computer Science*, pages 66–80, Siena, Italy, November 2003. Springer.

[BGP01]   Luciano Baresi, Franca Garzotto, and Paolo Paolini. Extending UML for modeling web applications. In *Proc. of 34th Anual Hawaii International Conference on System Sciences (HICSS'34)*, Maui, Hawai, January 2001. IEEE Press.

[BHW99]   Paul De Bra, Geert-Jan Houben, and Hongjing Wu. AHAM: A dexter-based reference model for adaptive hypermedia. In K. Tochtermann, J. Westbomke, U.K. Wiil, and J. Leggett, editors, *Proc. of ACM Conference on Hypertext and Hypermedia*, pages 147–156, Darmstadt, Germany, February 1999.

[BN96]   Martin Bichler and Stefan Nusser. Modular design of complex web-application with W3DT. In *Proc. of IEEE 5th Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises (WET ICE 96)*, Standford, California, USA, June 1996.

[Bru01]   Peter Brusilovsky. Adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 11(1-2):87–100, 2001.

[CDMN04]   Stefano Ceri, Peter Dolog, Maristella Matera, and Wolfgang Nejdl. Model-driven design of web applications with client-side adaptation. In Nora Koch, Piero Fraternali, and Martin Wirsing, editors, *ICWE 2004 - International Conference on Web Engineering*, volume 3140 of *LNCS*, pages 201–214, Munich, Germany, July 2004. Springer.

[CDMN05]   Stefano Ceri, Peter Dolog, Maristella Matera, and Wolfgang Nejdl. Adding client-side adaptation to the conceptual design of e-learning web applications. *Journal of Web Engineering*, 4(1):21–37, 2005.

[CE00]   Krysztof Czarnecki and Ulrich Eisenecker. *Generative Programing: Principles, Techniques, and Tools*. Addison Wesley, 2000.

[CFB00]   Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Language (WebML): a modeling language for designing web sites. *Computer Networks and ISDN Systems*, 33(1–6):137–157, June 2000.

[CFB+02]   S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kauffmann, 2002.

[CFB+03]   S. Ceri, P. Fraternali, A. Bongio, S. Butti, R. Acerbis, M. Tagliasacchi, G. Toffetti, C. Conserva, R. Elli, F. Ciapessoni, and C. Greppi. Architectural issues and solutions in the development of data-intensive web applications. In *Proceedings of CIDR 2003, January 2003, Asilomar, CA, USA*, 2003.

[CFM02]   Stefano Ceri, Piero Fraternali, and Maristella Matera. Conceptual modeling of data-intensive web applications. *IEEE Internet Computing*, 6(4), August 2002.

[Cha01]   Pierre-Antoine Champin. Rdf tutorial. http://www710.univ-lyon1.fr/ champin/rdf-tutorial/rdf- tutorial.html, April 2001.

[CHB92]   Derek Coleman, Fiona Hayes, and Stephen Bear. Introducing objectcharts or how to use statecharts in object-oriented design. *IEEE Transactions on Software Engineering*, 18(1):9–18, January 1992.

[CK02]   Cristina Cachero and Nora Koch. Conceptual navigation analysis: a device and platform independent navigation specification. In D. Schwabe, O. Pastor, G. Rossi, and L. Olsina, editors, *Proc. of Second International Workshop on Web-oriented Software Technology (IWWOST02)*, June 2002.

[Con99]   Jim Conallen. UML extension for web applications, March 1999. White paper.

[DB02a]   Peter Dolog and Mária Bieliková. Hypermedia modelling using UML. In Petr Hanáček, editor, *Proc. of ISM'2002 - Information Systems Modelling*, pages 79–86, Rožnov pod Radhoštěm, Czech Republic, April 2002.

[DB02b]   Peter Dolog and Mária Bieliková. Towards variability modelling for reuse in hypermedia engineering. In Yannis Manolopoulos and Pavol Návrat, editors, *Proc. ADBIS 2002 — Advances in Databases and Information Systems : 6th East European Conference*, volume 2435 of *LNCS*, pages 388–400, Bratislava, Slovakia, September 2002. Springer.

[DHN03]   Peter Dolog, Nicola Henze, and Wolfgang Nejdl. Logic-based open hypermedia for the semantic web. In *Proc. of International Workshop on Hypermedia and the Semantic Web, Hypertext 2003 Conference, Nottinngham, UK*, August 2003.

[DHNS04a]   Peter Dolog, Nicola Henze, Wolfgang Nejdl, and Michael Sintek. The personal reader: Personalizing and enriching learning resource using semantic web technologies. In Wolfgang Nejdl and Paul De Bra, editors, *Proc. of AH2004 — International Conference on Adaptive Hypermedia*, volume 3137 of *LNCS*, Einghoven, The Netherlands, August 2004. Springer.

[DHNS04b] Peter Dolog, Nicola Henze, Wolfgang Nejdl, and Michael Sintek. Personalization in distributed e-learning environments. In *Proc. of WWW2004 — The Thirteen International World Wide Web Conference*, New Yourk, May 2004. ACM Press.

[DK02] A. van Deursen and P. Klint. Domain-specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10(1):1–17, 2002.

[DN03] Peter Dolog and Wolfgang Nejdl. Using UML and XMI for generating adaptive navigation sequences in web-based systems. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *Proc. of UML 2003 — The Unified Modeling Language. Model Languages and Applications. 6th International Conference*, volume 2863 of *LNCS*, pages 205–219, San Francisco, CA, USA, October 2003. Springer.

[DN04] Peter Dolog and Wolfgang Nejdl. Using UML-based feature models and UML collaboration diagrams to information modelling for web-based applications. In *Proc. of UML 2004 — The Unified Modeling Language. Model Languages and Applications. 7th International Conference*, LNCS. Springer, October 2004.

[DNe04] Peter Dolog, Wolfgang Nejdl, and Daniel Olmedilla (eds.). Artefacts and service network v3 (d2.3). Deliverable of fp5 eu/ist Elena project ist-2001-37264, L3S Research Center, University of Hannover, Hannover, Germany, June 2004. `http://wwww.elena-project.org`.

[DNe05] Peter Dolog, Wolfgang Nejdl, and Daniel Olmedilla (eds.). Schema distribution and evaluation report (d2.7). Deliverable of fp5 eu/ist Elena project ist-2001-37264, L3S Research Center, University of Hannover, Hannover, Germany, May 2005. `http://wwww.elena-project.org`.

[Dol04] Peter Dolog. Model-driven navigation design for semantic web applications with the uml-guide. In Maristella Matera and Sara Comai, editors, *Engineering Advanced Web Applications*, pages 75–86. Rinton Press, 2004.

[DS05] Peter Dolog and Michael Schäfer. A framework for browsing, manipulating and maintaining interoperable learner profiles. In Liliana Ardissono, Paul Brna, and Antonija Mitrović, editors, *Proc. User Modeling 2005: 10th International Conference, UM 2005*, volume 3538 of *LNAI*, Edinburgh, Scotland, UK, July 2005. Springer.

[FP00] Piero Fraternali and Paolo Paoliny. Model-driven development of web applications: The autoweb system. *ACM Transactions on Information Systems*, 28(4):323–382, October 2000.

[GCP01] Jaime Gómez, Cristina Cachero, and Oscar Pastor. Conceptual modeling of device-independent web applications. *IEEE Multimedia*, 8(2):26–39, April–June 2001.

[GFdA98]   Martin L. Griss, John Favaro, and Massimo d' Alessandro. Integrating feature modeling with the RSEB. In P. Devanbu and J. Poulin, editors, *Proc. of 5th International Conference on Software Reuse*, pages 76–85, Victoria, Canada, June 1998. IEEE Computer Society Press.

[GG99]   Hans-Werner Gellersen and Martin Gaedke. Object-oriented web application development. *IEEE Internet Computing*, 3(1):60–68, 1999.

[Goo94]   Kees G. W. Goossens. Structure and behaviour in hardware verification. In J.J. Joyce and C.-J.H. Seger, editors, *International Workshop on Higher Order Logic Theorem Proving and its Applications*, volume 780, pages 73–87, Vancouver, Canada, 1994. Springer-Verlag.

[GP93]   Franca Garzotto and Paolo Paolini. HDM — a model-based approach to hypertext application design. *ACM Transactions on Information Systems*, 11(1):1–26, January 1993.

[Gro00a]   Object Management Group. OMG unified modelling language specification, version 1.3, March 2000. Available at http://www.omg.org/. Accessed on June 1, 2001.

[Gro00b]   Object Management Group. OMG XML metadata interchange (XMI) specification, version 1.1, November 2000. Available at http://www.omg.org/. Accessed on June 1, 2002.

[GSG00]   Martin Gaedke, Chrstian Segor, and Hans-Werner Gellersen. WCML: Paving the way for reuse in object-oriented web engineering. In *Proc. of 2000 ACM Symposium on Applied Computing (SAC 2000)*, Villa Olmo, Como, Italy, March 2000.

[HBFV03]   Geert-Jan Houben, Peter Barna, Flavius Frasincar, and Richard Vdovjak. Hera: Development of semantic web information systems. In Juan Manuel Cueva Lovelle, Bernardo Martin Gonzalez Rodriguez, Luis Joyanes Aguilar, Jose Emilio Labra Gayo, and Maria del Puerto Paule Ruiz, editors, *Proceedings of International Conference on Web Engineering, ICWE 2003*, number 2722 in LNCS, pages 529–538, Oviedo, Spain, July 2003. Springer Verlag.

[HBR94]   Lynda Hardman, Dick C.A. Bulterman, and Guido Van Rossum. The Amsterdam Hypermedia Model: Adding time and context to the dexter model. *Comunications of the ACM*, 37(2):50–64, February 1994.

[HDN04]   Nicola Henze, Peter Dolog, and Wolfgang Nejdl. Towards personalized e-learning in a semantic web. *Educational Technology and Society Journal. Special Issue on Ontologies and the Semantic Web for E-learning*, 7(4), October 2004.

[Her98]   American Heritage. The american heritage dictionary. Houghton Mifflin, Boston, MA, 1998.

[HK00]   Rolf Hennicker and Nora Koch. A UML-based methodology for hyper-media design. In S. Stuart A. Evans and B. Selic, editors, *Proc. of UML 2000 Conference*, York, England, October 2000. Springer LNCS 1939.

[HMT87]   Frank Halasz, T. Moran, and R. Trigg. Notecards in a nutshell. In *Proc. of CHI and GI Conference*, pages 45–52, Toronto, Canada, April 1987.

[HN99]   Nicola Henze and Wolfgang Nejdl. Bayesian modeling for adaptive hypermedia systems. In *Proc. of LWA'99: Lernen, Wissensentdeckung und Adaptivität — ABIS'99: 7ter Workshop Adaptivität und Benutzer-modellierung in interaktiven Softwaresystemen*, Magdeburg, Germany, September 1999.

[HS94]   Frank G. Halasz and Meyer Schwartz. The Dexter Hypertext Reference Model. *Comunications of the ACM*, 37(2):30–39, February 1994.

[IEE]   IEEE. IEEE P1484.2/D7, 2000-11-28. draft standard for learning technology. public and private information (papi) for learners (papi learner). Available at: `http://ltsc.ieee.org/archive/harvested-2003-10/working\_groups/wg2.zip`. Accessed on December 20, 2003.

[IEE02]   IEEE Learning Technology Standards Committee. IEEE standard for learning object metadata (IEEE 1484.12.1–2002). http://ltsc.ieee.org/, July 2002.

[IKK97]   Tomás Isakowitz, A. Kamis, and M. Koufaris. Extending the capabilities of RMM: Russians dolls and hypertext. In *Proceedings of the 30th Annual Hawaii International Conference on System Sciences*, January 1997.

[JBR99]   Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison Wesley, Massachusetts, 1999.

[JGJ97]   Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success*. ACM Press, 1997.

[KCHN90]   Kyo C. Kang, Sholom G. Cohen, James A. Hess, and William E. Novak. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-21, ESD-90-TR-222, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1990.

[Koc98]   Nora Koch. Towards a methodology for adaptive hypermedia systems development. In Timm and Marc Rössel, editors, *Proc. of ABIS-98: Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen*, October 1998.

[Koc01]   Nora Koch. Software engineering for adaptive hypermedia systems? In Paul De Bra, editor, *Proc. of Third Workshop on Adaptive Hypertext and Hypermedia, 8th International Conference on User Modeling*, July 2001.

[Kra97]   Miloš Kravčík. Internet based collaboration: the presence and perspectives. In *Proc. of WebNet 97*, Toronto, Canada, November 1997.

[LBW99]   David B. Lowe, Andrew J. Bucknell, and Richard G. Webby. Improving hypermedia development: a reference model-based process assessment method. In *Proceedings of the ACM International Conference on Hypertext and Hypermedia(Hypertext '99)*, pages 139–146, Darmstadt, Germany, February 1999.

[LFS⁺98]   Alon Levy, Daniela Florescu, Dan Suciu, Jaewoo Kang, and Mary Fernandez. Catching the boat with Strudel: experiences with a web-site management system. In *In SIGMOD'98*, 1998.

[LLY99]   Heeseok Lee, Choongseok Lee, and Cheonsoo Yoo. A scenario-based object-oriented hypermedia design methodology. *Information and Management*, 36(3):121–138, September 1999.

[LS]   O. Lassila and R.R. Swick. W3c resource description framework (rdf) model and syntax specification. Available at: http://www.w3.org/TR/REC-rdfsyntax/. Accessed on October 25, 2002.

[MA02]   Dirk Muthig and Colin Atkinson. Model-driven product line architectures. In Gary J. Chastek, editor, *Software Product Lines, Second International Conference, SPLC 2*, volume 2379 of *Lecture Notes in Computer Science*, pages 110–129, San Diego, CA, USA, August 2002. Springer.

[MNZS03]   Zoltán Miklós, Gustaf Neumann, Uwe Zdun, and Michael Sintek. Querying semantic web resources using TRIPLE views. In *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, USA, October 2003.

[Myl98]   John Mylopoulos. Information modeling in the time of the revolution. *Inf. Syst.*, 23(3-4):127–155, 1998.

[Nel]   Theodor Holm Nelson. Xanalogical Structure, Needed Now More than Ever: Parallel Documents, Deep Links to Content, Deep Versioning and Deep Re-Use . Available at: http://www.sfc.keio.ac.jp/ ted/XUsurvey/xuDation.html. Accessed on March 1, 2002.

[NF00]   Jin-Cheon Na and Richard Furuta. Context-aware hypermedia in dynamically-changing environment supported by a high-level petri net. In *Proc. of 11th ACM Conference on Hypertext and Hypermedia*, pages 222–223, San Antonio, Texas, June 2000.

[NN95]   Jocelyne Nanard and Marc Nanard. Hypertext design environments and the hypertext design process. *Communications of the ACM*, 38(8):49–56, August 1995.

[NS03]   Michael Nilsson and Wolf Siberski. RDF Query Exchange Language (QEL) - Concepts, Semantics and RDF Syntax. Available at: `http://edutella.jxta.org/spec/qel.html`. Accessed: 20th September 2003, 2003.

[NWQ⁺02] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Micahel Sintek, Ambjoern Naeve, Michael Nilsson, Matthias Palmér, and Tore Risch. EDUTELLA: a P2P Networking Infrastructure based on RDF. In *In Proc. of 11th World Wide Web Conference*, Hawaii, USA, May 2002.

[oCm02] Assosiation of Computing machinery. The acm computer classification system. http://www.acm.org/class/1998/, 2002.

[PBvdL05] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering Foundations, Principles and Techniques*. Springer Verlag, 2005.

[PMdO99] Fabiano Borges Paulo, Paulo Cesar Masiero, and Maria Cristina Ferreira de Oliviera. Hypercharts: Extended statecharts to support hypermedia specification. *IEEE Transactions on Software Engineering*, 25(1):33–49, January–February 1999.

[pos] Poseidon for UML CASE tool web page. Available at: http://www.gentleware.com/.

[RS00] W. Retschitzegger and W. Schwinger. Towards modeling of dataweb applications - a requirements' perspective. In *Proc. of the Americas Conference on Information Systems (AMCIS)*, Long Beach, California, August 2000.

[SD02] Michael Sintek and Stefan Decker. TRIPLE—A query, inference, and transformation language for the semantic web. In Ian Horrocks and James A. Hendler, editors, *Proc. of ISWC 2002 — 1st International Semantic Web Conference*, volume 2342 of *LNCS*, Sardinia, Italy, June 2002. Springer.

[SEI00] Carnegie-Mellon University Software Engineering Institute. Domain engineering and domain analysis, September 2000. URL: http://www.sei.cmu.edu/str/descriptions/deda-body.html.

[SF89] P. David Stotts and Richard Furuta. Petri-net-based hypertext: Document structure with browsing semantics. *ACM Transactions on Information Systems*, 7(1):3–29, January 1989.

[SF98] P. David Stotts and Richard Furuta. Hyperdocuments as automata: Verification of trace-based browsing properties by model checking. *ACM Transactions on Information Systems*, 16(1):1–30, January 1998.

[SG00] Christian Segor and Martin Gaedke. Crossing the gap - from design to implementation in web-application development. In *Proc. of Information Resources Management Association International Conference 2000*, Anchorage, USA, May 2000.

[SL02] Tim Schattkowsky and Marc Lohmann. Rapid development of modular dynamic web sites using uml. In J.-M. Jézéquel, H. Hussmann, and S. Cook, editors, *In Proc. of 5th International Conference on UML 2002*

- The Unified Modeling Language, pages 336–350. Springer, LNCS 2640, October 2002.

[SR98]   Daniel Schwabe and Gustavo Rossi. An object-oriented approach to web-based application design. *Theory and Practise of Object Systems (TAPOS), Special Issue on the Internet*, 4(4):207–225, October 1998.

[SRS00]   Luiz Fernando G. Soares, Rogério F. Rodrigues, and Débora C. Muchaluat Saade. Modeling, authoring and formatting hypermedia documents in the HyperProp system. *ACM Multimedia System Journal*, 8(2):118–134, March 2000.

[STA96]   Unisys STARS. Software technology for adaptable reliable systems. organization domain modeling (ODM) guidebook, version 2.0. Technical Report STARS-VC-A025/001/00, Unisys STARS, 1996.

[Ste03]   Perdita Stevens. Small-scale xmi programming: A revolution in uml tool use? *Automated Software Engineering*, 10:7–21, 2003.

[TC92]   W. Tracz and L. Coglianese. Domian-specific software architecture engineering process guidelines. Technical Report ADAGE-IBM-92-02, Loral Federal Systems, 1992.

[THH95]   Manfred Thüring, Jörg Hannermann, and Jörg M. Haake. Hypermedia and cognition: Design for comprehension. *Communications of the ACM*, 38(8):57–66, August 1995.

[TLHL01]   T.Berners-Lee, J. Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[vDKV00]   Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: An annotated bibliography. *SIGPLAN Notices*, 35(6):26–36, 2000.

[W3C03]   W3C. Owl web ontology language semantics and abstract syntax. Technical Report. Available at: http://www.w3.org/TR/owl-semantics/. Accessed: 20th September 2003, August 2003.

[WB01]   Gerhard Weber and Peter Brusilovsky. Elm-art: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education*, 12(4):351–384, 2001.

[Wie98]   Roel Wieringa. A survey of structured and object-oriented software specification methods and techniques. *ACM Comput. Surv.*, 30(4):459–527, 1998.

[Wit94]   James V. Withey. Implementing model based software engineering in your organization: An approach to domain engineering, 1994. CMU/SEI-94-TR-01, see also `http://www.sei.cmu.edu/mbse/index.html`.

[Wit96]   James V. Withey.   Investment analysis of software assets for product
           lines, 1996. CMU/SEI-96-TR-010.

[WL99]   David M. Weiss and Chi Tau Robert Lai.   *Software Product-Line Engi-
           neering: A FamilyBased Software Development Process.* Addison Wesley,
           August 1999.

[WR98]   Weigang Wang and Roy Rada. Structured hypertext with domain seman-
           tics. *ACM Transactions on Information Systems,* 16(4):372–412, October
           1998.

Appendix A

Metamodel for Feature Model

We use standard extension mechanism of the UML — the stereotypes to define the feature metamodel. For explanation purposes we use the class diagram to relate introduced stereotypes to existing metamodel elements similarly to [Gro00a]. Figure A.1 depicts a metamodel for feature models. We omit several attributes of metaclasses, which are defined at the metamodel level. They can be found in [Gro00a]. The newly introduced elements are marked by `stereotype` stereotype. The metaclasses already defined in the UML metamodel are marked by `metaclass` stereotype.

First of all, the «`Concept`» element is a stereotype of the `Class`. We need to differentiate the «`Concept`» from the `Class` because we assign features to the «`Concept`» differently from the `Class`. Although we associate features to a concept differently from UML, we do not restrict classical possibilities to assign attributes to a concept.

We need to associate features to the `Concept`-s or to other features. We need to define dependencies between features as well. This is not possible with the `Feature` element as it is defined in the UML. The aggregation relationship of the `Feature` to the `Classifier` at the metamodel level prescribes the way how the `Feature` can be associated (aggregated) into the `Classifier`. If features are inserted into a class at the model level, they are mandatory for the class. A class can have features in a model but relationships between features are not allowed.

We define the «`ConceptualFeature`» as an abstract stereotype of the `ModelElement`. The «`ConceptualFeature`» can be either the «`MandatoryFature`» or the «`OptionalFeature`». This definition of the «`ConceptualFeature`» allow us to distinguish it from the UML `Feature` element and to model associations (by means of the `Association` and its `AssociationEnd`-s) between features as well.

To model dependencies between features we introduce the «`VariationPoint`». The «`VariationPoint`» represents constraints on feature configurations, i.e. it somehow relates features. Because of this, we define «`VariationPoint`» as a stereotype of the `Association`. It means that it has at least two features or concepts associated. It differs from normal association by its root element, which represents the element for which a configuration of features is relevant. It has the `VariationKind` attribute of the type `VariationKind`, which is enumerated from exclusive or (`XOR`), inclusive or (`OR`), and conjunction (`AND`). This attribute is represented as tagged value at the model level. The «`VariationPoint`» is also a stereotype of the `Classifier` to be able to model associations between different variation points. This is useful for ex-
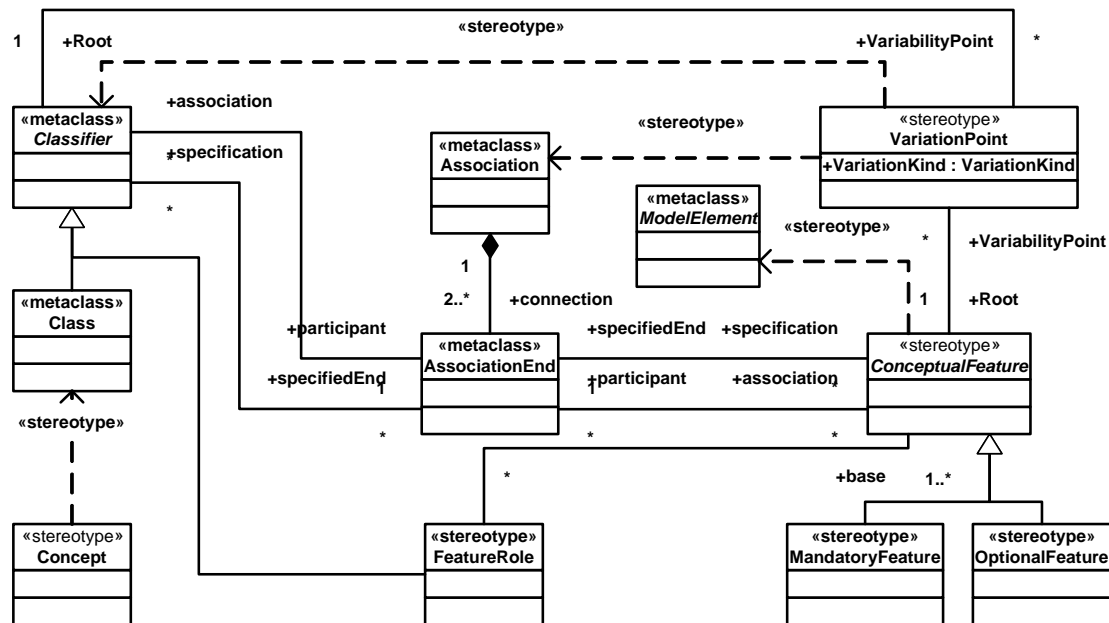
Figure A.1: A metamodel for a feature model in UML

ample when we want to assert that some features are mutually required but one or more features have alternative possibilities.

As the features are further refined to collaboration diagrams, they have to be related to existing elements for the collaboration diagrams in the UML. We model roles similarly to collaboration diagrams at the specification level. It means that we use the `ClassifierRole`, the `AssociationRole`, and the `AssociationEndRole` similarly to the UML. The metamodel of mentioned elements can be found in [Gro00a].

We introduce the «`FeatureRole`» as a stereotype of the `Classifier` for feature roles modeling. The relationship to features is modeled similarly to the `Classifier-Role` metaclass. We cannot use the `ClassifierRole` metaclass for modelling feature roles, because feature is not defined as a kind of the `Classifier` and the `ClassifierRole` is defined just for the `Classifier`-s.

Appendix B

Lebenslauf

Peter Dolog, geboren am 28. August 1976 in Brezno, Slowakische Republik.

1994 – 1998	Bachelor Studium Informatik, Slowakische Technische Universität, Bratislava, Slowakische Republik
1998	Bachelor Diplom in Informatik (Bc.) Title der Bachelorarbeit: Informationssysteme für kleine Firmen
1997 – 2002	Software Developer, Quality Auditor, CRM Consultant, TTC spol. s r. o., Slowakische Republik
1998 – 2000	Master Studium Informatik, Slowakische Technische Universität, Bratislava, Slowakische Republik
2000	Ingenieur Diplom in Informatik (Dipl.-Ing.) Titel der Diplomarbeit: Performance-Aspekte einer Application-Server-Implementierung basierend auf der Oracle Application Server Designer Technology
2000 – 2002	Promotion Studium, Informatik, Slowakische Technische Universität, Bratislava, Slowakische Republik
2002 –	Wissenschaftliche Mitarbeiter am Forschungszentrum L3S, Universität Hannover
Aug. 2003	Forschung Aufenthalt am DFKI, Michael Sintek
May 2004	Forschung Aufenthalt am Politecnico di Milano, Stefano Ceri
Jul. 2004 –	Best Paper Award, 4th International Conference on Web Engineering, ICWE'2004, Münich, Germany
Nov. 2004	Forschung Aufenthalt am Politecnico di Milano, Stefano Ceri
Sep. 2005	Forschung Aufenthalt am Vrije Universiteit Amsterdam, Heiner Stuckenschmidt, Holger Wache, headed by Frank van Harmelen