

***Software Engineering for  
Adaptive Hypermedia Systems***

***Reference Model, Modeling Techniques  
and Development Process***

Nora Parcus de Koch

Dissertation

zur Erlangung des akademischen Grades  
des Doktors der Naturwissenschaften  
an der Fakultät für Mathematik und Informatik  
der Ludwig-Maximilians-Universität München

Tag der Einreichung: 20. Oktober 2000  
Tag der mündlichen Prüfung: 12. Dezember 2000

Berichterstatter

Prof. Dr. Martin Wirsing  
Priv.-Doz. Dr. Rolf Hennicker

Copyright © 2001 by Nora Parcus de Koch

*to my daughters  
Tania and Ivonne*



## **Abstract**

---

This work proposes an engineering approach for adaptive hypermedia applications. Adaptive hypermedia applications are user-centred systems that are based on the hypermedia paradigm, i.e. they are a network of nodes connected by links and they administrate a user model to adapt themselves dynamically to the user.

This software engineering approach consists of an object-oriented, incremental and iterative development process. It supports the entire lifecycle of adaptive hypermedia applications from feasibility study to maintenance and includes project management, software development and quality management activities. The main focus of the work is the description of a systematic methodology for the analysis and design of adaptive hypermedia applications. An extension to the Unified Modeling Language (a so-called UML profile) is specified to provide an adequate notation for the visual representation. It allows for an easy construction of navigation, presentation and adaptation models, which are part of the proposed methodology. The software engineering approach is based on an object-oriented reference model for adaptive hypermedia applications that is visually modeled in UML and formally specified in the Object Constraint Language (OCL).

This software engineering approach is also appropriate for the development of personalised Web applications and the development of non-adaptive hypermedia applications that can make use of the presented techniques and methodology merely by skipping the specific features of user modeling and adaptation.

## **Acknowledgements**

---

I am indebted to a large number of people for their inspiration and support during the research and writing of my doctoral thesis.

Particular thanks go to:

Martin Wirsing, my adviser, whose constant support, and confidence that I could do it, encouraged me to continue and complete this project.

Rolf Hennicker, whose interest in my work, discussions on software engineering and valuable critique of the methodology I proposed, had a great influence on my research progress.

The Hochschulsonderprogramm III of the Ludwig-Maximilians-Universität München, for financial support and Edda Ziegler for her organisational commitment to this program. Thanks also to the interdisciplinary and multicultural members of the HSP-Kolloquium for sharing their experience in the preparation of doctoral thesis and post-doctoral projects.

María Victoria Cengarle, for numerous discussions on software engineering, formal methods and, especially, on OCL.

Julita Vassileva, for suggestions and comments related to user modeling and adaptive systems.

Luis Mandel, Daniel Schwabe, Alicia Diaz, Hubert Baumeister, Alfred Helmerich, and Friederike Nickl, for discussions on hypermedia design and Web engineering.

Gustavo Rossi, whose doctoral thesis was a source of inspiration at the beginning of my research activities.

Florian Albrecht and Thomas Tiller, for the excellent work they did during their diploma thesis on implementing the adaptive hypermedia system SmexWeb.

Rudolf Haggemüller, for flexible working hours at F.A.S.T. GmbH.

Alexander Knapp and Luis Mandel, for helping me make my come back in the academic world after years of working in industry.

Dorith Schießl, Alba Püschel and Lindsay Dyson-Smith, for their friendship and advice in the use of the English language.

My special thanks to:

Miguel, my husband, without whose love and patience I would never have finished this work.

My daughters, Tania and Ivonne, for their love and understanding for so much time I spent in front of the computer over the last few years.

# Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>1</b>
1.1	MOTIVATION.....	5
1.2	GOALS AND RESULTS.....	6
1.3	ORGANISATION OF THE WORK.....	8
<b>2</b>	<b>ADAPTIVE HYPERMEDIA SYSTEMS.....</b>	<b>11</b>
2.1	ADAPTATION AND FEASIBILITY .....	13
2.1.1	<i>Objectives</i> .....	13
2.1.2	<i>Benefits</i> .....	14
2.1.3	<i>Risks</i> .....	14
2.1.4	<i>Difficulties</i> .....	15
2.2	ADAPTATION LIFECYCLE .....	15
2.3	LEVELS OF ADAPTATION.....	18
2.4	ADAPTATION METHODS AND TECHNIQUES.....	19
2.4.1	<i>Adaptive Content</i> .....	20
2.4.2	<i>Adaptive Navigation</i> .....	22
2.4.3	<i>Adaptive Presentation</i> .....	24
2.5	THE ROLE OF THE USER MODEL .....	25
2.6	ACQUISITION TECHNIQUES .....	26
2.7	USER INTERACTION POSSIBILITIES .....	27
2.8	APPLICATIONS AREAS.....	27
2.8.1	<i>Instructional Hypermedia Systems</i> .....	28
2.8.2	<i>Hypermedia Search Engines</i> .....	29
2.8.3	<i>Online Information Systems</i> .....	30
2.8.4	<i>Online Help Systems</i> .....	31
2.8.5	<i>Personal Assistants</i> .....	32
2.9	HOW TO MEASURE ADAPTIVE HYPERMEDIA SYSTEMS .....	32
<b>3</b>	<b>USER MODELS AND USER MODELING.....</b>	<b>35</b>
3.1	CHARACTERISTICS OF USER MODELS .....	38
3.2	USER MODEL FOUNDATIONS .....	39
3.3	TYPES OF USER MODELS .....	41
3.4	OBJECTIVES OF USER MODELING.....	44
3.4.1	<i>Help to Learn</i> .....	44
3.4.2	<i>Support Collaboration and Assistance</i> .....	45
3.4.3	<i>Find and Tailor Information</i> .....	46
3.4.4	<i>Improve Man-Machine Communication</i> .....	46
3.5	INITIALISATION OF USER MODELS.....	46
3.5.1	<i>Explicit Questioning</i> .....	46
3.5.2	<i>Default Assumptions</i> .....	47
3.6	USER MODEL INTERNAL STRUCTURE .....	48
3.6.1	<i>Overlay Model</i> .....	49
3.6.2	<i>User Profile</i> .....	50
3.6.3	<i>Stereotyped Model</i> .....	50
3.6.4	<i>Bayesian Networks</i> .....	51

3.7	USER MODELS UPDATING PROCESS.....	52
3.7.1	Acquisition techniques.....	53
3.7.2	Acquisition process.....	54
3.8	SHARING USER MODELS.....	58
3.9	DEVELOPMENT OF USER MODELS.....	59
<b>4</b>	<b>AN OBJECT-ORIENTED REFERENCE MODEL.....</b>	<b>61</b>
4.1	REFERENCE MODELS FOR HYPERMEDIA SYSTEMS.....	62
4.1.1	<i>The Dexter Hypertext Reference Model.....</i>	<i>64</i>
4.1.2	<i>AHAM: Adaptive Hypermedia Application Model.....</i>	<i>69</i>
4.1.3	<i>AHM: Amsterdam Hypermedia Model.....</i>	<i>71</i>
4.1.4	<i>DFHM: Dortmund Family of Hypermedia Models.....</i>	<i>72</i>
4.2	THE MUNICH REFERENCE MODEL.....	73
4.2.1	<i>Architecture of Adaptive Hypermedia Systems.....</i>	<i>73</i>
4.2.2	<i>Extensions to the Dexter Hypertext Reference Model.....</i>	<i>75</i>
4.3	FORMAL SPECIFICATION OF THE MUNICH MODEL.....	77
4.3.1	<i>The Domain Model.....</i>	<i>78</i>
4.3.2	<i>The User Model.....</i>	<i>92</i>
4.3.3	<i>The Adaptation Model.....</i>	<i>98</i>
4.3.4	<i>The Run-Time Layer.....</i>	<i>104</i>
4.3.5	<i>Authoring Functions.....</i>	<i>114</i>
4.4	BASIS FOR THE DEFINITION OF MODELING TECHNIQUES.....	119
<b>5</b>	<b>COMPARISON OF HYPERMEDIA ENGINEERING APPROACHES.....</b>	<b>121</b>
5.1	DEVELOPMENT OF HYPERMEDIA SYSTEMS.....	122
5.1.1	<i>HDM: Hypermedia Design Method.....</i>	<i>124</i>
5.1.2	<i>RMM: Relationship Management Methodology.....</i>	<i>125</i>
5.1.3	<i>EORM: Enhanced Object Relationship Methodology.....</i>	<i>126</i>
5.1.4	<i>OOHDM: Object-Oriented Hypermedia Design.....</i>	<i>127</i>
5.1.5	<i>SOHDM: Scenario-based Object-orientedHypermedia Design Methodology.....</i>	<i>129</i>
5.1.6	<i>WSDM: Web Site Design Method.....</i>	<i>129</i>
5.1.7	<i>MacWeb Approach.....</i>	<i>130</i>
5.1.8	<i>HFFPM: Hypermedia Flexible Process Modeling.....</i>	<i>131</i>
5.1.9	<i>OO/Pattern Approach.....</i>	<i>133</i>
5.1.10	<i>WAE – Conallen Process.....</i>	<i>133</i>
5.1.11	<i>Lowe-Hall’s Engineering Approach.....</i>	<i>134</i>
5.2	NOTATIONS USED IN HYPERMEDIA DESIGN.....	134
5.3	COMPARING HYPERMEDIA DEVELOPMENT METHODS.....	135
5.4	THE UNIFIED PROCESS AND HYPERMEDIA DEVELOPMENT.....	140
5.5	LESSONS LEARNED FROM THE COMPARATIVE STUDY.....	144
<b>6</b>	<b>MODELING TECHNIQUES FOR THE DESIGN OF ADAPTIVE HYPERMEDIA</b>	<b>145</b>
6.1	USE CASE MODEL.....	148
6.2	CONCEPTUAL MODEL.....	150
6.3	USER MODEL.....	153
6.4	NAVIGATION MODEL.....	155
6.4.1	<i>Navigation Space Model.....</i>	<i>156</i>



6.4.2	<i>Navigation Structure Model</i> .....	159
6.5	PRESENTATION MODEL.....	173
6.5.1	<i>Abstract User Interface Model</i> .....	174
6.5.2	<i>Presentation Structure Model</i> .....	179
6.5.3	<i>Presentation Flow Model</i> .....	184
6.5.4	<i>Object Lifecycle Model</i> .....	187
6.6	ADAPTATION MODEL.....	189
<b>7</b>	<b>THE SOFTWARE DEVELOPMENT PROCESS .....</b>	<b>195</b>
7.1	ADAPTIVE HYPERMEDIA SYSTEMS .....	197
7.1.1	<i>Covering the Life Cycle</i> .....	198
7.1.2	<i>Iterative Development</i> .....	199
7.2	PHASES OF THE PROCESS.....	201
7.2.1	<i>Inception</i> .....	203
7.2.2	<i>Elaboration</i> .....	206
7.2.3	<i>Construction</i> .....	207
7.2.4	<i>Transition</i> .....	209
7.2.5	<i>Maintenance</i> .....	210
7.3	DEVELOPMENT PROCESS.....	211
7.3.1	<i>Requirements Capture</i> .....	212
7.3.2	<i>Analysis and Design</i> .....	230
7.3.3	<i>Implementation</i> .....	252
7.4	PROJECT MANAGEMENT.....	262
7.4.1	<i>Risk Management</i> .....	262
7.4.2	<i>Iteration Planning</i> .....	268
7.4.3	<i>Iteration Evaluation</i> .....	272
7.5	QUALITY MANAGEMENT.....	275
7.5.1	<i>Validation</i> .....	276
7.5.2	<i>Verification</i> .....	279
7.5.3	<i>Testing</i> .....	282
<b>8</b>	<b>DEVELOPMENT OF SMEXWEB APPLICATIONS – A CASE STUDY .....</b>	<b>287</b>
8.1	THE SMEXWEB FRAMEWORK.....	288
8.1.1	<i>The Architecture</i> .....	289
8.1.2	<i>The Tutor</i> .....	292
8.1.3	<i>The Communication</i> .....	293
8.1.4	<i>The UserModel</i> .....	293
8.1.5	<i>The Hyperspace</i> .....	294
8.2	DEVELOPMENT OF THE EBNF-APPLICATION.....	295
8.2.1	<i>Inception Phase (Iteration 1)</i> .....	296
8.2.2	<i>Elaboration Phase (Iterations 2 and 3)</i> .....	298
8.2.3	<i>Construction Phase (Iteration 4 and 5)</i> .....	302
8.2.4	<i>Transition Phase (Iteration 6)</i> .....	305
8.2.5	<i>Maintenance Phase (Iteration &gt;= 7)</i> .....	308
8.3	ANALYSIS AND DESIGN MODELS FOR THE EBNF-APPLICATION.....	309
8.3.1	<i>Use Case Model</i> .....	309
8.3.2	<i>Conceptual Model</i> .....	311
8.3.3	<i>User Model</i> .....	312
8.3.4	<i>Navigation Model</i> .....	314

8.3.5	<i>Presentation Model</i> .....	317
8.3.6	<i>Adaptation Model</i> .....	321
8.4	THE TAXONOMY APPLICATION .....	324
8.5	LEARNING PROCESS SUPPORTED BY SMEXWEB .....	325
<b>9</b>	<b>CONCLUSIONS</b> .....	<b>327</b>
9.1	CONCLUDING REMARKS ABOUT THE REFERENCE MODEL .....	328
9.2	CONCLUDING REMARKS ABOUT THE MODELING TECHNIQUES .....	329
9.3	CONCLUDING REMARKS ABOUT THE DEVELOPMENT PROCESS.....	331
9.4	PROPOSING FUTURE RESEARCH .....	332
	<b>APPENDIX UML EXTENSION FOR HYPERMEDIA</b> .....	<b>335</b>
	STEREOTYPES FOR MODELING HYPERMEDIA APPLICATIONS.....	335
	STEREOTYPES FOR MODELING ADAPTIVE FEATURES.....	338
	<b>REFERENCES</b> .....	<b>341</b>

## List of Figures and Tables

Figure 2-1	<i>Lifecycle Model of Adaptation</i> .....	16
Table 2-2	<i>Methods and Techniques for Adaptive Content</i> .....	21
Table 2-3	<i>Methods and Techniques for Adaptive Navigation</i> .....	24
Table 2-4	<i>Methods and Techniques for Adaptive Presentation</i> .....	25
Figure 3-1	<i>A User Stereotype Hierarchy</i> .....	48
Figure 3-2	<i>Bayesian Network</i> .....	52
Figure 4-1	<i>Layers of the Dexter Hypertext Reference Model</i> .....	65
Figure 4-2	<i>UML Class Diagram for the Storage Layer of the Dexter Model</i> .....	68
Figure 4-3	<i>UML Class Diagram for the Run-Time Layer and Part of the Storage Layer</i> .....	69
Figure 4-4	<i>Architecture of Adaptive Hypermedia Systems</i> .....	74
Figure 4-5	<i>UML Class Diagram of the Domain Model</i> .....	78
Figure 4-6	<i>The Children Association Class</i> .....	89
Figure 4-7	<i>UML Class Diagram of the User Model and Associations to the Domain Model</i> .....	93
Figure 4-8	<i>Adaptation Model</i> .....	100
Figure 4-9	<i>UML Class Diagram for the Run-Time Layer and Part of the Storage Layer</i> .....	105
Table 5-1	<i>Methods for Hypermedia Development: Processes, Techniques, Notations and Tools</i> .....	136
Table 5-2	<i>Concepts Used in the Methods</i> .....	138
Figure 5-3	<i>UP Workflows covered by the Hypermedia Development Method</i> .....	139
Figure 6-1	<i>Models built during the Design</i> .....	147
Figure 6-2	<i>Use Case Model for the Online Library Application</i> .....	149
Figure 6-3	<i>Class with additional Compartment Variants</i> .....	151
Figure 6-4	<i>Conceptual Model of the Online Library Application</i> .....	152
Figure 6-5	<i>User Model of the Online Library Application</i> .....	154
Figure 6-6	<i>User Model Packages</i> .....	155
Figure 6-7	<i>Navigation Class</i> .....	157
Figure 6-8	<i>Stereotype for External Node</i> .....	157
Figure 6-9	<i>Navigation Space Model of the Online Library Application</i> .....	158
Figure 6-10	<i>Index Class</i> .....	161
Figure 6-11	<i>Shorthand Notation for Index Class</i> .....	161
Figure 6-12	<i>Guided Tour Class</i> .....	161
Figure 6-13	<i>Shorthand Notation for Guided Tour Class</i> .....	161
Figure 6-14	<i>Query Class</i> .....	162
Figure 6-15	<i>Shorthand Notation for Query</i> .....	162
Figure 6-16	<i>Navigation Structure Model (First Step)</i> .....	163
Figure 6-17	<i>Menu Class</i> .....	165

Figure 6-18	<i>Shorthand for Menu Class</i> .....	165
Figure 6-19	<i>Navigation Structure Model (Second Step)</i> .....	166
Figure 6-20	<i>Pattern for Access Structures</i> .....	167
Figure 6-21	<i>Navigation Structure Model (Third Step)</i> .....	169
Figure 6-22	<i>Indexed Guided Tour</i> .....	170
Figure 6-23	<i>Navigation Context triggered by an Index</i> .....	171
Figure 6-24	<i>Grouped Context</i> .....	172
Figure 6-25	<i>Context Package and Context Change</i> .....	172
Figure 6-26	<i>Shorthand for Context Package</i> .....	172
Figure 6-27	<i>Presentation Class as Container other Presentation Modeling Elements</i> ..	175
Figure 6-28	<i>Presentation Class of Library Main Menu</i> .....	177
Figure 6-29	<i>Presentation Class of Composite Library and Author Menu</i> .....	177
Figure 6-30	<i>Presentation Class Author</i> .....	177
Figure 6-31	<i>Presentation Class Article</i> .....	177
Figure 6-32	<i>Presentation Class SearchAuthorByName</i> .....	178
Figure 6-33	<i>Presentation Class AuthorIndexByName</i> .....	178
Figure 6-34	<i>Window Class</i> .....	180
Figure 6-35	<i>Frameset Class and Frame Class</i> .....	180
Figure 6-36	<i>Pattern for Main Presentation Elements</i> .....	181
Figure 6-37	<i>Partial Presentation Structure Model</i> .....	182
Figure 6-38	<i>Page Presentation</i> .....	182
Figure 6-39	<i>Alternative Representation for Page</i> .....	183
Figure 6-40	<i>Presentation Flow Model of the "Search Author" Scenario</i> .....	186
Figure 6-41	<i>Object Lifecycle Model</i> .....	188
Figure 6-42	<i>Rule Class</i> .....	190
Figure 6-43	<i>User Behaviour Class</i> .....	190
Figure 6-44	<i>Pattern for Adaptation</i> .....	191
Figure 6-45	<i>Part of the Adaptation Model</i> .....	193
Figure 7-1	<i>Iteration Workflow</i> .....	200
Figure 7-2	<i>UML-based Web Engineering Process for One Release</i> .....	203
Figure 7-3	<i>Requirements Capture Workflow</i> .....	214
Figure 7-4	<i>Template for Use Case Description</i> .....	227
Figure 7-5	<i>Use Case Model for the Online Library Application</i> .....	228
Figure 7-6	<i>Part of the Glossary</i> .....	229
Figure 7-7	<i>Analysis and Design Workflow</i> .....	231
Figure 7-8	<i>Conceptual Model of the Online Library Application</i> .....	239
Figure 7-9	<i>User Model of the Online Library Application</i> .....	241
Figure 7-10	<i>Navigation Space Model of the Online Library Application</i> .....	243
Figure 7-11	<i>Navigation Structure Model of the Online Library Application</i> .....	244
Figure 7-12	<i>Presentation Structure Model of the Online Library Application (Partial View)</i> .....	245
Figure 7-13	<i>Presentation Class of Library Main Menu</i> .....	246
Figure 7-14	<i>Presentation Class of Composite Library and AuthorMenu</i> .....	246

Figure 7-15	<i>Presentation Class Author</i> .....	247
Figure 7-16	<i>Presentation Class Article</i> .....	247
Figure 7-17	<i>Abstract User Interface Model of one Online Library Page</i> .....	247
Figure 7-18	<i>Presentation Flow Model of the Online Library Application (Partial View)</i> .....	248
Figure 7-19	<i>Adaptation Model of the Online Library Application</i> .....	249
Figure 7-20	<i>Architecture of the Online Library Application</i> .....	250
Figure 7-21	<i>Class Description</i> .....	251
Figure 7-22	<i>Subsystems of the Online Library Application</i> .....	252
Figure 7-23	<i>Implementation Workflow</i> .....	253
Figure 7-24	<i>Risk Management Workflow</i> .....	263
Table 7-25	<i>Risks for the Online Library Project</i> .....	267
Table 7-26	<i>Actions to Mitigate Risks for the Online Library Project</i> .....	267
Figure 7-27	<i>Iteration Planning Workflow</i> .....	268
Figure 7-28	<i>Iteration Evaluation Workflow</i> .....	272
Figure 7-29	<i>Sample Iteration Report for the Online Library Project</i> .....	274
Figure 7-30	<i>Validation Workflow</i> .....	276
Figure 7-31	<i>Verification Workflow</i> .....	280
Figure 7-32	<i>Testing Workflow</i> .....	283
Figure 8-1	<i>Architecture of SmexWeb</i> .....	290
Figure 8-2	<i>The SmexWeb Server</i> .....	291
Table 8-3	<i>Activities performed during the Inception Phase</i> .....	297
Figure 8-4	<i>Frameset proposed for SmexWeb Applications</i> .....	299
Table 8-5	<i>Activities performed during the Elaboration Phase</i> .....	300
Table 8-6	<i>Activities performed during the Construction Phase</i> .....	302
Figure 8-7	<i>Formal Description of the Exercise's Task of the EBNF-Application</i> .....	304
Figure 8-8	<i>Pragmatic Description of the Exercise's Task of the EBNF-Application</i> .....	305
Figure 8-9	<i>An Interactive Exercise of the EBNF-Application</i> .....	306
Table 8-10	<i>Activities performed during the Transition Phase</i> .....	307
Table 8-11	<i>Activities performed during the Maintenance Phase</i> .....	308
Figure 8-12	<i>Use Case Model of the EBNF-Application</i> .....	310
Figure 8-13	<i>Conceptual Model of the EBNF-Application</i> .....	311
Figure 8-14	<i>User Model of the EBNF-Application</i> .....	313
Figure 8-15	<i>Exercise View of the Navigation Space Model</i> .....	314
Figure 8-16	<i>General View of the Navigation Space Model</i> .....	315
Figure 8-17	<i>Exercise View of the Navigation Structure Model</i> .....	316
Figure 8-18	<i>General View of the Navigation Structure Model</i> .....	317
Figure 8-19	<i>Presentation Structure Model</i> .....	319
Figure 8-20	<i>Presentation Flow Model</i> .....	320
Figure 8-21	<i>Abstract User Interface Model of the Exercise "Building Rules"</i> .....	321
Figure 8-22	<i>Adaptation Model of the EBNF-Application (Partial View)</i> .....	324



"As We May Think"  
Vannevar Bush,  
The Atlantic Month,  
July 1945

# 1 Introduction

The World Wide Web (WWW or Web) has changed the way we work and the way we live. The Web is currently the most successful hypermedia system in existence, but “the Web is far from done” as Berners-Lee (1999), the inventor of the WWW stresses. We are moving to a more “intelligent”, collaborative, and personalised Web. It is becoming more a personal environment for collaborative creation than just for browsing. The future “semantic Web” proposed by Berners-Lee is a Web of data with meaning in the sense that software can learn about what the data means, to process it.

This work focuses on the development of “*personalised*” *hypermedia* applications (Web applications are a special case of hypermedia applications, i.e. hypermedia applications for the Web)<sup>1</sup>. Personalisation, also called customisation or adaptation, is the process, which – when applied to software – consists of a change in the behaviour of the system based on the knowledge the system has of the user. This knowledge can be supplied by the user herself<sup>2</sup> or by the software system, which is prepared to observe and register the user’s behaviour. Software systems with the capability to acquire information about the user, to build a user model with it, and

---

<sup>1</sup> In this work the more general term hypermedia is used.

<sup>2</sup> The feminine form is used in this work for for *users* and *customers* and the masculine form for *developers*.

to utilise the user model to dynamically adapt themselves are called *adaptive systems*.

*Adaptive hypermedia systems (AHS)* are both, adaptive and hypermedia systems. They combine hypermedia with Intelligent Tutoring Systems (ITS) guidance through the adaptation of the information presented, the layout of the presentation or the way in which the information units are visited, i.e. how navigation is performed.

The concept of *hypertext*, later *hypermedia* was created by Nelson (1960), who defined hypertext as “non-sequential writing – text that branches and allows choices to the reader, best read at an interactive screen”. Nelson assumed that both the reading and writing process would be supported by hypertext. However, the first person to define the concept of hypertext (without using the word hypertext) was Busch in his famous article “As we may think” (1945). There – referring to scientific publications – he says: “A record, if it is to be useful to science, must be continuously extended, it must be stored, and above all it must be consulted... Our ineptitude in getting the record is largely caused by the artificiality of systems of indexing... The human mind does not work that way. It operates by association”. Busch proposed a solution called a memex: “A memex is a device in which individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory”.

A more pragmatic description of hypertext stems from Conklin (1987): “The concept of hypertext is quite simple: Windows on a screen are associated with objects in a database, and links are provided between these objects, both graphically (as labelled tokens) and in the database (as pointers)”.

The essential feature of hypermedia is the concept of a *network* of nodes connected by links. A *node* is a unit that contains text and/or multimedia elements, such as images, video, audio, or animations. A *link* is usually directed and connects two nodes: the *source node* and the *destination node*. A link is associated with a specific part of the content of the source node, e.g. a word, a phrase or an image. This part of the source node is called an *anchor*. It is the linking capability which allows a non-linear organisation of the text or multimedia content. The activity whereby the user accesses a node by following links is known as *browsing* or *navigation*. Hypertext basic concepts are formally described by the Dexter Hypertext Reference Model in the specification language Z (Halasz & Schwartz, 1990). Adaptive hypermedia applications are those that adapt the content or presentation of their nodes and/or their links to the user.



The advantage of the hypermedia style of structuring and accessing information has, however, some limitations and shortcomings for users and developers. The disadvantages for the users are an easy disorientation – the so called “lost in the hyperspace” syndrome – and the cognitive overload. Thus, the developer’s challenge is the management of the cognitive burden which is placed to the user, producing high quality hypermedia applications that reduce the overhead of remembering hyperlinks and support the users through navigation. If a hypermedia application is well developed the user does not require to understand the entire information space; to navigate she has only to understand the local context to find a suitable link destination (Lowe & Hall, 1999).

The typical design problems of hypermedia, such as how to increase local and global coherence of a hypermedia applications, how to improve orientation and how to facilitate navigation are addressed by many researchers. Among others the following design principles are suggested: the use of higher order information units, the visualisation of the structure of the hypermedia system (e.g. overview or local maps), the provision of additional navigation facilities (e.g. forward and backward navigation, providing indexes, history trails, landmarks and bookmarks), and the use of a stable screen layout (Thüring, Hannemann & Haake, 1995 and Linard & Zeiliger, 1995).

*Adaptive hypermedia systems* seek to solve the disorientation and cognitive overload problems in a different way, i.e. by adopting a user-centred approach. The user is observed by the system, a user model is built for the individual user, and the system adapts visible aspects of the system to the user. More precisely, the adaptation of the content and presentation avoids cognitive overhead by showing the appropriate information with the adequate layout to the individual user. Adaptive navigation solves the disorientation problem by limiting browsing space, providing annotations for the links, hiding some irrelevant links or suggesting the best link to follow. From the commercial point of view, personalisation has the advantage to draw new visitors, to turn visitors into buyers, to increase revenues and to increase advertising efficiency.

Eklund and Zeiliger (1996) summarise the characteristics of adaptive hypermedia systems as follows: AHS are “explicitly based on hypertext (or hypermedia), and use a model of the user’s knowledge or goals to modify links or content to present individualised instruction or guidance”. Adaptive hypermedia gives the Web intelligence in the sense that these systems have the ability to “understand” the user and to customise the application.

Most of the current adaptive hypermedia applications have been implemented as prototypes and improved in successive steps. We need a process to perform the development in an effective and efficient way. This process must address project

management, development and test techniques, metrics for evaluation, etc. Such a process supports, the human mind's planning activities during the development of complex software systems through knowledge representations and guidelines.

In this work an engineering approach for adaptive hypermedia systems is presented. It focuses on the process and on object-oriented modeling. It is not concerned with implementation. The main motivation for this decision is the fast evolution of the implementation technologies and platforms.

The proposed software engineering approach consists of a reference model, modeling techniques and a development process. The Unified Modeling Language (UML, 1999)<sup>3</sup> was chosen for all models, techniques and notations used. This decision was taken based on the fact that the UML is an international standard since 1997 (OMG, 2000). The Object Constraint Language (OCL)<sup>4</sup>, that is part of the UML, is used for the formal specification and the process is based on the Unified Software Development Process (Jacobson, Booch, Rumbaugh, 1999).

In summary, the software engineering approach is a Unified Process and UML-based approach. It is named UWE, acronym for UML-based Web Engineering.

The main characteristics of UWE are:

- It is an entirely object-oriented approach.
- It presents a reference model visually represented in UML and formally specified in OCL.
- It supports visual modeling techniques.
- It provides a UML extension (profile) for adaptive hypermedia applications.
- It defines a development process that covers the whole lifecycle of adaptive hypermedia applications.

The UML-based Web engineering approach was validated using several case studies. This work shows how the guidelines and techniques of UWE have been used in the development process of an adaptive hypermedia exercising system for computer science students. The SmexWeb framework (Albrecht, 1998 and Tiller, 1998) was used for the implementation of this exercising system.

---

<sup>3</sup> UML Version 1.3

<sup>4</sup> OCL Version 1.3

## 1.1

## Motivation

---

Adaptive hypermedia applications are complex software systems, whose development process demands an exhaustive feasibility study, adequate planning and experience in the construction of hypermedia applications, user modeling and adaptation techniques.

Software engineering is always a knowledge-intensive process. It requires different types of knowledge from the software developers: procedural, semantic and episodic. Procedural knowledge is related to the developer's ability to interact with the environment, e.g. finding the appropriate people to interview (potential users and customers) or using a case tool. Semantic and episodic knowledge are based on information. The former is related to the meaning of descriptions, such as processes, notations and adaptation techniques. The latter consists of experience with such knowledge, e.g. the usefulness of certain diagrams, patterns, user models or programming constructs. Semantic knowledge is obtained through learning, while episodic knowledge is obtained through experience based on the topics learned (Robilliard, 1999). Adaptive hypermedia developers and adaptive Web developers as well as general software developers all require semantic, episodic and procedural knowledge.

Software development involves processing a large amount of information belonging to a set of different domains. As we are not able to register all this information, we require assistance to manage it. A specific methodology for adaptive hypermedia applications, such as the engineering approach presented in this work allows for the reuse of information and knowledge gained in the development of a wide spectrum of hypermedia and adaptive applications. It supports a more effective and efficient development of such applications. Adaptive hypermedia and adaptive Web development is different because between others, it requires a great deal of communication and team synergy, tasks are often done in parallel and the goal is to take into account the needs, preference and knowledge of each user. The complexity of adaptive hypermedia applications is generally underestimated. Managers, developers and academics still consider hypermedia development as an authoring activity rather than an application development to which well-known software engineering practices could apply (Murugesan, Deshpande, Hansen & Ginige, 1999).

This work was motivated by the lack of an software engineering approach for adaptive hypermedia systems. The main aim of this approach was the use of current object-oriented techniques. General object-oriented software engineering approaches, such as the Unified Process (Jacobson, Booch & Rumbaugh, 1999) and the Rational Unified Process (Kruchten, 1998) or specific methodologies for

hypermedia like RMM (Isakowitz, Stohr & Balasubramanian, 1995), OOHDM (Schwabe & Rossi 1998), and HFPM (Olsina, 1998) are not sufficient as they do not cover user modeling and adaptation issues. Wu, Houben and de Bra (1999) cover specific adaptive aspects, but they neither address a process that allows the systematic development of adaptive applications nor use object-oriented techniques.

In addition, the choice of UML as the modeling language for this work was taken before the UML became an OMG standard. Kobryn (1999) stresses that the major benefits of international standardisation for a specification include wide recognition and acceptance, which typically enlarge the market for products based on it. Despite the problems with UML, having a standard is a step in the right direction. The modeling community can focus now on improving one modeling language instead of a few dozen OO languages (Siau & Cao, 2001). The use of UML for modeling purposes is a must, as there is a guarantee that UML is updated and improved, it is supported by tools, conferences and books, and, even most importantly, UML improves the communication between people involved in a software development project as they “talk” the same language.

## 1.2

## **Goals and Results**

---

By way of an analogy to hypermedia engineering (Lowe & Hall, 1999), *engineering for adaptive hypermedia applications* can be defined as a systematic, disciplined and measurable approach that supports the entire life cycle of adaptive hypermedia systems. This life cycle goes from conception through the elaboration, construction, delivery and maintenance to the cessation of the application.

The goal of the software engineering approach is to support developers during these different phases in organising mental activities, working at various levels of detail and abstraction, generating visual representations adapted to the designers level of experience, presenting the solution’s constraints, building representations of the application and finally outlining plan structures and strategies (Robilliart, 1999).

The main results of the present work – the UWE approach – are a reference model for adaptive hypermedia systems, modeling techniques for the analysis and design of such applications and a development process that covers the entire lifecycle of these applications. The validation of the engineering approach was done using a case study of an adaptive learning-system for student use.

The reference model for adaptive hypermedia applications was elaborated in order to identify the features that characterise adaptive hypermedia and personalised Web applications as a previous step to the definition of appropriate modeling techniques. It is named after the place where it was developed: Munich Reference Model. The use of UML allows for a graphical representation of the reference model and the use of OCL for a formal description of the functionality of the model (Gogolla & Richters, 2000).

The main characteristics of the Munich Reference Model are:

- It is based on the Dexter Hypertext Reference Model.
- It includes a user model and an adaptation model.
- It is formally specified in OCL and visually represented in UML.

Existing user modeling techniques and the more relevant methodologies for general hypermedia have been compared in this work prior to the definition of a specific development process for adaptive hypermedia applications. The survey analyses how these engineering approaches cover the hypermedia applications lifecycle, which techniques and notations they use, and what their strengths and weaknesses are.

The modeling techniques of the UWE approach for the methodical analysis and design of adaptive hypermedia applications comprise modeling elements, notation and a method. The notation and semantics of these elements define a “lightweight” UML extension (profile); the method supports the systematic construction of adaptive hypermedia applications. The aim is to obtain a method that allows as many steps as possible to be performed in an automatic way.

The main characteristics of the UWE modeling techniques are:

- It supports visual and systematic modeling.
- Hypermedia issues, such as content, navigation and presentation are treated separately from user modeling and adaptation issues.
- It provides a UML profile based on the extension mechanisms of the UML and uses it for the construction of the analysis and design models.

These modeling techniques were developed during the initial phase of this project and are also known as UML-based Hypermedia Design Method (UHDM). They are now integrated in the analysis and design workflows of UWE development process for adaptive hypermedia.

The development process of UWE tailors the Unified Process for the hypermedia (Web) domain and for adaptive hypermedia, in particular. At the same time it extends the Unified Process to include project management and quality management support. UWE includes a maintenance phase and changes the idea of quality control management incorporating workflows for requirements validation and design verification in addition to the testing of the implemented software.

The main characteristics of the development process of the UWE approach are:

- It is an object-oriented, workflow-based, iterative and incremental process.
- It specialises the Unified Process for the development of adaptive hypermedia applications describing which “experts” (workers) are required, which activities they perform and which specific artifacts they produce.
- It extends the coverage of the Unified Process development cycle including a maintenance phase.
- It adds development process supporting workflows for project management and quality management.
- It changes the idea of quality control management incorporating workflows for requirements validation and design verification in addition to testing.

In summary, this work presents UWE, an object-oriented engineering approach for adaptive hypermedia systems. Special emphasis has been put on visual modeling and the definition of an appropriate UML profile. The modeling techniques are embedded in the development process, which aims to cover the entire lifecycle from inception to maintenance of adaptive hypermedia applications. This work also focuses on an object-oriented, formal specification of a reference model for these applications in UML and OCL.

## **1.3 Organisation of the Work**

This work is organised in nine chapters.

The current Chapter 1 provides the introduction to this work and focuses on its motivation and goals.

Chapter 2 introduces the main concepts relating to adaptive hypermedia, adaptation methods and adaptation techniques.

Chapter 3 describes the characteristics of user models, classifies them and outlines the user modeling process.

Chapter 4 presents an object-oriented reference model for adaptive hypermedia applications.

Chapter 5 gives a comparative overview of methods for the development of hypermedia applications. It constitutes a basis for the design techniques and the development process for adaptive hypermedia applications presented in the next two chapters.

Chapter 6 presents modeling techniques for the analysis and design of adaptive hypermedia applications. It includes the definition of a set of modeling elements and methodical steps for the construction of each model.

Chapter 7 presents a development process covering the whole lifecycle of adaptive hypermedia applications.

Chapter 8 describes how the modeling techniques and the development process are validated with case studies.

Chapter 9 outlines conclusions on the results of this work and provides ideas about future work to be done in this field.

Appendix A includes the UML profile defined for the adaptive hypermedia domain.





*“The greatest strength and weakness of a hypermedia system lies in the issue of navigation”  
John Eklund & Romain Zeiliger,  
AusWeb '1996.*

## **2** Adaptive Hypermedia Systems

“Adaptive hypermedia systems are hypermedia systems which reflect some features of the user in a user model and use this model by adapting various visible aspects of the system to the user” (Brusilovsky, 1996b). They have the advantages of both user-model-based adaptive systems and hypermedia systems. Classical hypermedia applications, are thus enhanced by an agent that improves the system behaviour based on the analysis of the user behaviour.

According to this definition an adaptive hypermedia system (AHS) must fulfil the following requirements:

- it must be a *hypermedia system* allowing navigation through the hyperspace of the application domain,
- it must include a *user model* to describe the user, and
- it must provide an *adaptive mechanism* for the dynamic adaptation of the hypermedia on the basis of the state of the user model.

The adaptation consists in changes of the content and/or the presentation of nodes and links.

What does the user model contain? Let us call a user characteristic the user’s knowledge, preference, interest, tasks or goals. A user model can therefore be defined as a representation of characteristics, which the system “believes” that a user possesses (Benyon & Murray, 1993). It contains characteristics, which are

different from both, the actual characteristics the user has and the characteristics the designer of the system believes the user has and he employs in the design of the system. The representation of the user can only be a partial and incomplete representation. It is a less complete knowledge than the knowledge another person could acquire of the user because a live person can use different input channels – audio, visual, tactile – simultaneously to get information. The system may, however be able to reason faster and more precisely than people. User models are usually very pragmatic as they limit the number of characteristics to those that are strictly required for a specific application.

A clear distinction must be made between hypermedia systems that are customisable – called adaptable systems – and adaptive hypermedia systems. In both cases the user plays a central role and the ultimate goal is to offer a personalised system. They differ in the way the adaptation is performed.

- An *adaptable hypermedia system* allows the user to configure the system by changing some parameters and the system then adapts its behaviour accordingly. It is an external system or the user who decides when her user model should be changed, e.g. at the beginning of a session. This configuration consists of setting preferences.
- An *adaptive hypermedia system* is a hypermedia system that adapts autonomously (Bulterman, Rutledge, Hardman & van Ossenbruggen, 1999). It monitors the user's behaviour, registers this behaviour in a user model and adapts the system dynamically to the current state of the user model. The system uses the user's browsing actions, her answers to questionnaires and the initial information the user may provide, to adapt the nodes and the navigation. These adaptations can be made by changing predefined presentations or constructing them out of pieces of information. In the latter case, where a dynamic generation of pages is performed, such systems are also known as dynamic hypermedia systems (De Bra, 1999).

Most of the applications that make use of adaptive hypermedia are currently in the area of educational hypermedia, because in this area it is easier to build a detailed user model (Patterno & Mancini, 1999).

This chapter presents an overview of adaptive hypermedia systems. The first section outlines the objectives, benefits and risks of adaptation. Section 2 presents an adaptation lifecycle and Section 3 defines three different adaptation levels. For each of these levels of adaptation methods and techniques are briefly described in Section 4. Sections 5, 6 and 7 delineate the role of the user model, acquisition techniques and user interaction possibilities in adaptation, respectively. Section 8 presents a classification of application areas for which adaptive systems have been

implemented. Finally, Section 9 provides a list of possible measures for adaptive hypermedia systems.

## **2.1 Adaptation and Feasibility**

Many questions arise related to adaptation in hypermedia systems.

- Do we need hypermedia systems that are adaptive?
- What are the goals of these systems?
- What benefits do this type of system offer to the user?
- Which additional risks do adaptive hypermedia systems have?

Every hypermedia system is adaptive in some sense as the user decides what link to follow in each browsing step. Users use the same hypermedia application in different ways by the selection of different navigation paths. But navigation sometimes offers too much freedom or insufficient guidelines (Eklund & Zeiliger, 1996), as the user does not know which path to select, i.e. which path is most appropriate for her. Adaptive hypermedia techniques can be a powerful tool to support users while browsing on the WWW. These techniques increase the functionality of the Web.

### **2.1.1 Objectives**

The objective of a hypermedia system is to make information easily accessible to the user. An adaptive hypermedia system has the enhanced objective of improving the flexibility and the comfort, which a hypermedia system provides to each individual user. Hypermedia systems improve the human computer interaction, operational speed and accuracy as well as they enhance the user learning process, the user satisfaction and they increment the number of users. This goal is achieved by gradually adapting content and the functionality to the level of competence and interests of the user. Adaptive hypermedia systems therefore, seek to learn as much as possible about the users while they interact with the system. The communication can be seen as a dialog between user and adaptive system, which improves with time as the system's knowledge of the user becomes more precise and the user gets used to the system (de La Passardiere & Dufresne, 1992).

Consequences of an adaptive and flexible system are that a wider group of users can be reached, i.e. for each user the system behaves as if it had been built for her

interests and competence. And last but not least, if the hypermedia system is easy to use, e.g. fewer clicks to reach useful information, this is a factor in motivating users to continue using the system.

### 2.1.2 **Benefits**

---

Adaptive hypermedia systems improve human computer interaction, operational speed and accuracy as well as they enhance the user learning process and the user satisfaction. They improve comprehension of the content and decrease search and navigation time. There are two other benefits that adaptive hypermedia systems offer: they are useful for a heterogeneous group of users and reduce the risk of being “lost in hyperspace”. They allow for a good combination of an active, self-directed participation of the user and some guidance or help provided by the system.

The adaptation is performed in a similar way to a tutor adapting his lessons based on feedback received from the students sitting in front of him. In both cases, the adaptation plays an important role in the success of the system or lessons, respectively. Users’ goals, knowledge, tasks, interests and preferences can vary at an initial point in time as well as through the period of time during which they are using the system. A solution, therefore, is to implement a system that takes into account these changing user characteristics. The same applies to frequently used Web systems, which seek to attract the attention of a heterogeneous group of users with varying interests, knowledge and tasks.

“Lost in hyperspace” means to be unable to find the path to information needed, or to be unable to find the way back to information that has already been read or seen (sometimes a mere couple of minutes ago). The risk of “lost in hyperspace” is reduced with adaptive hypermedia systems as they reduce the navigation space, thus eliminating links that are not of interest to the user or which might confuse her.

### 2.1.3 **Risks**

---

Adding intelligence to the hypermedia system and moving partial control from the user to the systems is contrary to the philosophy of the hypertext paradigm, which is supposed to give the user full control to explore the hyperspace. An adaptive interface is often perceived by the user as limiting, disorienting, unpredictable and incoherent. The design of an adaptive interface therefore, means that special

attention must be paid to such risks so that techniques are applied that are non-intrusive, motivating, non-disorienting, and helpful.

One of the problems of adaptive systems is that the interface is less stable for the user, making it more difficult for her to orient herself. The major risk is therefore, changing navigation and a presentation that confuses the user. For example, if the user goes back to take another look at pages she has already seen, these pages very often look different to the first time she saw them, as they are generated dynamically according to the current state of the user model. The user may be irritated by incomplete and/or hidden information or anchors, because she wants to decide for herself which links to follow. This risk is eliminated by systems like SmexWeb (Albrecht, Koch & Tiller, 2000), which keep a history of changes to the user model. In this way one page has the same look and feel throughout a session for a particular user.

Adaptive systems tend to become complex and are therefore systems that are expensive to build and maintain. Nevertheless, industry observers see the personalisation of the Web as the next important step in e-commerce applications, although one of the main problems is privacy. The concept of personalisation is subject of ethical debate. Some people fear the misuse of personal information gathered by organisations with or without the permission of their customers. Industrial observers, however, maintain that people will get use to this and will begin to see it less as an invasion of privacy and more as the benefit of software offering improved service and faster access to preferred information (Waters, 2000).

#### **2.1.4** Difficulties

The main difficulty consists of building adaptive functionality. The design and implementation of such systems represent a time-consuming and difficult job as adaptive hypermedia systems are complex and expensive systems. Another difficult is to determine how correct is the user model. Most of the existing adaptive hypermedia systems or frameworks have been developed by computer scientists and so have the first applications of these systems. There is a need for methodologies and case tools that make the construction of adaptive hypermedia systems easier, mainly for authors that are not computer experts.

## **2.2** Adaptation Lifecycle

Adaptive hypermedia systems and indeed adaptive systems in general, go through different steps during utilisation or stay in different states so as to interact with the user, adapt the presentation and update the user model. By contrast, non-adaptive software systems only oscillate between presentation and interaction. Jungmann and Paradies (1997) outline a four-steps lifecycle model. The steps are presentation, interaction, analysis and synthesis.

A slightly different lifecycle model for adaptation is shown in Figure 2-1. It is graphically represented with a UML state diagram (UML, 1999 and Harel & Gery, 1997), which depicts the states of the lifecycle model and the possible transitions between these states.

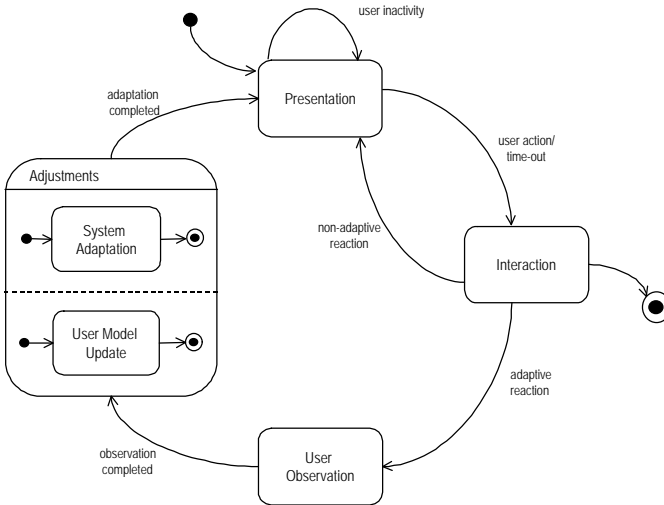


Figure 2-1: Lifecycle Model of Adaptation

The states are: *presentation*, *interaction*, *user observation*, and *adjustment*. The adjustment state is refined by using two concurrent states: *system adaptation* and *user model update*. Two adjustment alternatives are possible: user model update before or after the systems adapts according to the content of the user model. In addition to the four sequential transitions (*user action*, *adaptive reaction*, *acquisition completed* and *adaptation completed*), the model includes the transitions *user inactivity* and *non-adaptive reaction*. The former depicts the system waiting for a user action. The latter, *non-adaptive reaction* transition,

shows that the system can also react without adjusting the user model and the presentation.

The cycle starts with an initial presentation and a default user model. Stereotypes or interviews are usually used to provide the information for the first user model. These states have the following semantics:

- *Presentation.* The system presents to the user presentation elements or a page appropriate to the properties the system knows about the user. The system remains in this state until the user becomes active or it receives a time-out signal.
- *Interaction.* The system decides how to react to user action. Two alternatives are represented with two outgoing transitions: a non-adaptive and an adaptive reaction.
- *User observation.* This is a state the aim of which is to evaluate the information obtained from the user interaction with the system.
- *Adjustments.* This state comprises two sub-states: the user model update and the user interface (UI) adaptation, performed concurrently.
- *User model update.* In this state the result of the acquisition is used by the system to update the user model.
- *System adaptation.* The user model is utilised to adapt the presentation, content or links, i.e. to modify the user interface or generate a presentation that takes into account the user's goals or characteristics.

A *user action* produces the change from the state presentation to the state interaction, i.e. the user is then waiting for an activity of the system – usually the presentation of another page or interface object. In some systems a *time-out* can *also* trigger the transition from the state presentation to the state interaction. The system stays in the interaction state for as long as it needs to decide whether it will perform an *adaptive* or a *non-adaptive* cycle, and then changes to state acquisition or state presentation, respectively.

The transition to the adjustments state occurs after the *user observation* activities are completed. As soon as the user model is *updated* and the system adjusted for the next presentation the system proceeds to the next presentation. To note is that the system's adaptation can be performed before or after the user model update (pre and post mode). The finalisation of the adjustments mark the transition from the adjustments state to the presentation state.

During these states errors can be introduced due to an inappropriate acquisition of information about the user, incorrect assumptions by the user model update or errors by the adaptation process (Brusilovsky, 1998).

## 2.3 Levels of Adaptation

Content, structure and presentation are important issues in hypermedia systems. These issues are mostly treated separately by authoring processes and methods for hypermedia applications (see Chapter 5). Adaptive hypermedia systems also benefit from this separate treatment as they allow for adaptive content, adaptive navigation and adaptive presentation as defined in the next section. The scope of each of these issues is given below.

- *content*: The content consists of pieces of information included in hypermedia applications. They may be either time independent – called passive elements – such as text and images, or time-dependent – active elements – like video clips, audio tracks and animations.
- *structure*: This is the organisation of the content with a specification as to *which* content items will be visited and *how* they will be visited through navigation.
- *presentation*: This is the visualisation of the content and of the interactive elements that support the functionality of the hypermedia system. There are thus two different aspects to the presentation: the static layout and the description of the user interaction possibilities.

Interactive elements are those, which make it possible to access other elements (“navigate to”), to show passive elements (“display”) or to activate multimedia elements (“play”).

Adaptive systems tailor the information presented to the user’s preferences, knowledge or interests. This customising process may include changes such as the selection of pieces of information that are appropriate to the knowledge level of the user, or some guidance performed through the removal of links that the system considers of little use to the state of the user model at a given point in time. An adaptive system can adapt for example, on the basis of the user variability, the help, error messages, formatting, search strategies, task offer, input devices, dialogue style, content, etc.

Two different forms of hypermedia adaptation (technologies) are distinguished by Brusilovsky (1996a):



- adaptive presentation (at content-level) and
- adaptive navigation support (at link-level).

Another possible adaptation is a change at presentation-level, i.e. changes to the layout that do not affect the content, such as colours, font type or font size. If these changes to the layout are distinguished from content adaptation, then the following classification for adaptation is presented (Patterno & Mancini, 1999):

- *adaptive content* consists of selecting different information, such as different text, images, videos, animation, etc. depending on the current state of the user model. For example, the adaptive hypermedia system provides an expert in a certain domain with more information than a novice.
- *adaptive navigation* changes the link appearance, the link target or the number of links presented to the users as well as the order in which these links are presented.
- *adaptive presentation* shows different layouts of perceivable user interface elements, such as different type of media, different ordering or different colours, font size, font type or image size.

The first classification in content-level and link-level adaptation is based on the structure of the hypermedia and consists of nodes and links. The second classification is based on the three main aspects to be considered when developing hypermedia applications: content, navigation structure and presentation.

## 2.4 Adaptation Methods and Techniques

Different methods can be used to achieve adaptation. A method is determined by an adaptation idea defined at conceptual level. Adaptation methods are defined by Brusilovsky (1996b) as an abstraction of adaptive techniques. A technique is defined by a user model representation and an adaptation algorithm.

An adaptation method can be implemented by using different techniques and a technique can be used to implement more than one method (conceptual idea). The techniques mentioned above have been implemented by one or more existing adaptive hypermedia systems.

In the following subsections adaptation methods and techniques for content, navigation and presentation are outlined. A detailed description of adaptation methods, techniques and systems, which use these methods and implement

techniques, is presented by Brusilovsky (1996b). He distinguishes methods and techniques for adaptive presentation and adaptive navigation. In this work three methods and techniques are distinguished based on the definitions presented above. There are methods and techniques for adaptive content, adaptive navigation and adaptive presentation.

## 2.4.1 Adaptive Content

The objective of content-level adaptation methods is to increment the application usability for a wide group of users that have different knowledge or background. The content-level adaptation consists of providing additional, comparative or alternative content as well as hiding content.

The methods for adaptive content are:

- *additional content*

This is the most frequently used method for adaptive content. It consists of showing only relevant parts of information (hiding irrelevant parts) according to the user's level of knowledge, her goal, interests or preferences. This method is used to show

- *additional explanations,*
- *prerequisite explanations, or*
- *comparative explanations*

to be applied to concepts (terminology that comes from applications in educational areas).

- *content variant*

This method can be seen as a variant of the showing/hiding content method as it consists of showing a part of the information while at the same time hiding another part of the information. This method is also known as *explanation variants*.

The following techniques can be used to implement the methods described above, i.e. to manipulate content to be adapted to the user's characteristics. Most of these are used in adaptive hypertext systems, i.e. they are used for content of type text. But most of them can also be applied to multimedia content in general.

The techniques for adaptive content are:

- *stretchtext*  
The content is organised as a set of visible placeholders. Instead of moving to a new page, an activation of a placeholder will replace the activated placeholder extending the text (Höök, 1998). The adaptive hypermedia system determines which fragments are “stretched” (expanded) and which are “shrunk” (collapsed) for the initial presentation. The user can then decide which placeholder she will stretch, and which she might want to shrink. It should be noted that this technique allows both the user and the system to adapt the content.
- *conditional fragments*  
The user model and the concept relationships of the domain model provide the information that allows the system to determine which chunk of information should be presented to the user. The chunk of information may also consists of fragment variants, i.e. fragments related by an “or-exclusive” relationship.
- *page variants*  
This is a very simple technique, which consists of keeping two or more alternative pages with adapted content, e.g. one for each knowledge level: beginner, intermediate and expert.
- *frame-based approach*  
This technique allows the inclusion of all related information in a frame. Frames can be shown, hidden, presented alternatively or ordered. The frameset includes rules to decide which frames are presented to a user.

Table 2-2 shows which technique can be combined and used for the implementation of each adaptive content method.

Methods / Techniques	stretchtext	conditional fragments	page variants	frame-based approach
additional content	x	x		x
content variant		x	x	x

Table 2-2: Methods and Techniques for Adaptive Content

## 2.4.2 Adaptive Navigation

The objective of link-level adaptation is to support navigation preventing users to follow navigation paths that are irrelevant with their tasks or goals (Brusilovsky, 1997). Methods for adaptive navigation provide global or local guidance, support global or local orientation and generate personalised views. The adaptation consists of changes to the navigation structure or how this navigation structure is presented to the user.

Link-level adaptation can be applied to contextual links as well as to non-contextual links. Additional navigation support in form of site maps or trees, table of content, indexes and historical bookmarks lists also can benefit from adaptive navigation techniques.

The methods to support adaptive navigation are:

- *global guidance*  
The objective of the global guidance method is to assist the user in finding the shortest navigation path to the information she is looking for or wants to learn.
- *local guidance*  
The objective of the local guidance method is to assist the user in just one navigation step, i.e. to find the “best” link to follow from the current node.
- *global orientation*  
The objective of this method is to support the user in her knowledge of the hyperspace structure and her position in it.
- *local orientation*  
The objective of this method is to support the user in understanding what the different navigation possibilities of the current position mean and to help the user to follow the appropriate link.
- *personalised views*  
This method is an agent-based approach. It consists in the generation and update of a personalised view of a hyperspace. The agents are responsible for finding appropriate links for the user, thus maintaining the personalised view.

The following techniques for navigation adaptation manipulate anchors and links with the purpose of adapting navigation dynamically to the user characteristics given by the current state of the user model, i.e. they are used to implement the methods mentioned above.

These techniques for adaptive navigation are:

- *direct guidance*  
The user sees only one option to continue with the browsing activity, i.e. just one anchor or button to navigate to the “next” page is displayed. The destination of this “best” link is determined by the system.
- *link annotation*  
Anchors of links are “annotated”, that is they present a different visible aspect, such as a different colour, bullet or text to show the relevance of the destination. Even a Boolean adaptive annotation (visited/not visited) can be quite useful. Special cases of link annotation are link *highlighting* and link *hiding*. Highlighting of links is used even in non-adaptive applications. Hidden links are present but their anchors are not visible as they are annotated in the same way as text is presented in their surroundings, i.e. these anchors cannot be recognised as anchors of links. The “traffic-light” annotation is a well-known example where red, yellow and green icons are presented together with the anchored text of a link to indicate the degree of appropriateness.
- *link removing*  
Links that the system considers inappropriate are removed, i.e. they are not longer available. Anchors of these links are replaced by text, for example.
- *sorting of links*  
This consists of the ordering of a set of anchors, so that links are presented in decreasing order of relevance to the user. The disadvantage of adaptive ordering is that each time the user enters the same page, the ordered anchors may be different.
- *passive navigation*  
This consists of the addition of non-explicit links (without anchors) that are used by the system to offer assistance to the user when the system identifies a user behaviour pattern, e.g. the user remains inactive during a certain period of time or the user navigates back and forward.

Another technique described in the literature is map-adaptation (e.g. Brusilovsky, 1998 and De Bra, Houben & Wu, 1999). It consists of a combination of the other techniques, the only difference being that it is applied to a graphical visualisation of the navigation (link) structure. The map is usually presented in a separate frame.

Adaptive navigation techniques reduce the navigation space either by eliminating anchors (direct guidance, link hiding and link removal) or by guiding the user's attention to a reduced group of anchors (annotated and sorted links). A user-specific limitation of the navigation space prevent users from getting "lost in the hyperspace".

These five techniques for adaptive navigation can be combined for optimal navigation support. Table 2-3 shows which technique can be used for the implementation of each navigation content method.

Methods / Techniques	direct guidance	link annotation	link removing	link sorting	passive navigation
global guidance	x			x	x
local guidance	x	x	x	x	x
global orientation		x	x		
local orientation		x	x	x	
generation of personalised views	x	x	x	x	x

*Table 2-3: Methods and Techniques for Adaptive Navigation*

### 2.4.3

### **Adaptive Presentation**

The objective of presentation-level adaptation is to adapt the layout to the visual preferences or needs of the user. Methods for adaptive presentation assist the user with an appropriate layout or language. The adaptation consists of changes to the presentation. Sometimes these changes happen simultaneously with adaptation of content. Methods and techniques for adaptive presentation are often grouped with those for adaptive content. In this work they are presented separately so as to distinguish between methods and techniques that produce modifications in the layout and techniques that change the content shown to the user

The methods for adaptive presentation are:

- *multi-languages*

The objective of the multi-language method is adaptation to the language preferred by the user. This may be also context dependent.

- *layout variants*

The layout variants method includes all possible alternatives required in a presentation, e.g. colours, font size or font type, maximum size of images, text orientation, ordering of content fragments, etc.

The same techniques as for adaptive content, with the exception of stretchtext, can be used for presentation adaptation. These techniques are *page variants*, *conditional fragments* and *frame-based approach*. In addition, the *styleguiding* technique is used to implement the methods mentioned above.

- *styleguiding*

This consists of the definition of different styleguides that are used alternately for layout variants.

Table 2-4 shows which technique can be used for the implementation of each adaptive presentation method.

Methods / Techniques	page variants	conditional fragments	frame-based approach	styleguiding
multi-languages	x	x	x	
layout variants	x	x	x	x

Table 2-4: Methods and Techniques for Adaptive Presentation

## 2.5

## The Role of the User Model

Adjusting information (adaptive content), individualising layout (adaptive presentation) or providing the user with navigation support (adaptive navigation) are performed within the system on the basis of the information kept in the user model. Hence, the user model is an important part of an adaptive hypermedia system. It is defined as the system's representation of certain user characteristics and attitudes (Eklund & Zeilinger, 1996 and Paiva, Self & Hartley, 1995).

According to Eklund & Zeiliger (1996) the five main features that are represented inside user models are:

- the user's current goal or task,
- the user's knowledge on the domain presented in the hypermedia,
- the user's background or general knowledge,
- the user's experience, e.g. in the use of similar applications, or in hyper-space, and
- the user's preferences or interests.

A detailed description of user models and user modeling is presented in Chapter 3.

## 2.6 Acquisition Techniques

The features included in user models (listed above) need to be initialised and updated, i.e. the system needs to acquire information about the user. Different techniques are used to perform this acquisition (Kobsa, Müller & Nill, 1994), such as:

- *stereotypes*, that are defined by the designer at design-time. At run-time the system assigns a stereotype to each user.
- *interviewing* is performed by the system at run-time; the information supplied by the user is used for example to initialise the user model.
- *observation of user behaviour* consists of the analysis of the user's actions, plan recognition or inference mechanisms.

The process of acquiring information concerning the user behaviour comprises the steps of capturing the appropriate data, selecting the relevant information from data and inferring, i.e. interpreting the user interactions activities. How effectively user models can represent users is still controversial. Ramscar, Pain and Lee (1997) formulate this doubt as follows: Do we know what the user knows, and does it matter?" Kay (1993) remarks that modeling cannot be anything but a guess if it attempts to model the user's knowledge. Self (1996) stresses that "...the power of a student model does not lie in its fidelity but in the differences it indicates". Höök, Kaelgren and Waern (1995) and others promote the idea of a user model of the "glass box" type, i.e. a user model that is visible to the user and potentially manageable by the user.



## 2.7 User Interaction Possibilities

User behaviour can only be observed through her actions. According to Schneiderman (1998) the actions a user performs while interacting with a system are: menu selection, form filling, direct manipulation, natural language and command language. For hypermedia system the following actions are possible:

- *link following or browsing* is the typical action a user performs in order to navigate the hyperspace,
- *menu selection* consists of the selection of an item from a list of items, often it has the same effect as following a link,
- *form filling-in* allows the user to use the keyboard to enter a sequence of characters; it is an approach that gives the user a feeling of control over the dialog,
- *direct manipulation* is not permitted in the hypermedia paradigm directly; it requires the use of JavaApplets, for example.

No action by the user, i.e. no action during a certain period of time (timeout) can also be registered as user behaviour and is then used by the system for *passive navigation*. Passive navigation is a concept that was introduced by (Albrecht, 1998 and Tiller, 1998) in the implementation of the SmexWeb – Student modelled **exercising on the Web**. It allows the system to take control over the process of navigation when the user remains inactive, offering her some help or guidance.

User actions are usually stored in log files. The entries in the log file are used as triggers for the adaptation mechanisms defined on the basis of a set of adaptation rules. User behaviour is determined by the analysis of these log files.

## 2.8 Applications Areas

There is no restriction on developing adaptive hypermedia systems for any type of application area. These systems are normally built for applications where the hyperspace is large enough and the target users are known as a heterogeneous group of users who differ in their knowledge, interests, preferences and/or tasks. In the past an important number of adaptive hypermedia systems have been developed for the educational purposes, thus offering alternatives to intelligent tutor systems.

Adaptive hypermedia systems can be classified according to application areas into: instructional hypermedia systems, hypermedia search engines, online information

systems, online help systems and personal assistants. Some examples are given for each application area. A comparison of adaptive hypermedia systems can be found in (Brusilovsky, 1996b).

## **2.8.1 Instructional Hypermedia Systems**

The most popular adaptive hypermedia systems are systems related to the educational area. They are known as adaptive teaching, tutoring, learning and training systems or the recent term e-learning systems. These systems are mainly based on the utilisation of user knowledge for adaptation. They assume the adaptive system will be used by a heterogeneous group of students or learners in terms of knowledge. Students are observed during their work or learning process. These systems then adapt to the student's improvements.

One of the most interesting works in this area is the ELM-ART tutoring system that supports learning of the programming language LISP (Brusilovsky, Schwarz & Weber, 1996a and Weber & Specht, 1997). Adaptation is implemented by direct guidance (the system selects the next best step) and link annotation. The annotation follows a traffic light metaphor, where the colour green is used to indicate that a section is ready to be learned and recommended, yellow is used for ready to be learned but not recommended and red indicates not ready to be learned yet. The adaptation (the link annotation in this case) is performed whenever a learning unit is finished after all units that are a prerequisite to the current unit have been reviewed.

Other frameworks for adaptive teaching systems are INTERBOOK, TANGOW, KBS Hyperbook, SmexWeb, AHA, ISIS-Tutor and DCG.

INTERBOOK (Brusilovsky, Schwarz & Weber, 1996b) is a system for authoring and delivering adaptive electronic textbooks on the Web. All INTERBOOK-served electronic textbooks have generated table of content, a glossary and a search interface. The online books – in the same way as ELM-ART – use coloured bullet annotation to inform the user about the status of the node behind the link.

TANGOW (Carro, Pulido & Rodriguez, 1999) structures Web courses by means of teaching tasks and rules. It differs from ELM-ART in that uses a dynamic tree to restrict the set of teaching tasks to be reviewed. This is achieved by including in each dynamic generated page only those subtasks (fragments), which are considered to be relevant by the system at run-time. In addition, rules are used to analyse prerequisite conditions. KBS Hyperbook (Henze & Nejd, 1999) is another goal-driven approach that uses a Bayesian network technique for its user model.

SmexWeb (Albrecht, Koch & Tiller, 1999) is a framework that permits the development of teaching applications through the instantiation of a collection of abstract and concrete classes. Similar to TANGOW the authoring process consists mainly of the definition of concepts (tasks) and adaptation rules. All types of adaptation are supported by SmexWeb: adaptive content, adaptive navigation, adaptive presentation and passive navigation.

AHA (Adaptive Hypermedia Applications) is a generic hypermedia system based on the adaptation of pages using conditional fragments (De Bra & Calvi, 1998). The structure of the domain is similar to the SmexWeb structure. Concepts are related to other concepts through weighted links.

ISIS-Tutor system (Brusilovsky, 1997) uses different forms of adaptive navigation, such as direct guidance, hiding and annotation. The goal is to highlight the links corresponding to the student's goal and to hide concepts that belong to future learning targets.

The Dynamic Course Generation (DCG) proposed by Vassileva (1997) represents a quite different approach. It consists of a concept domain structure represented as a plan, which relates known concepts for the learner with the goal-concept of the course. The plan is then adapted dynamically according to the student's learning progress. This results in changes to the subtasks and steps the learner has to follow.

## **2.8.2** Hypermedia Search Engines

Hypermedia search engines combine traditional retrieval systems with hypermedia features. The objective is to obtain a manageable set of responses to a query put to an information hyperspace as opposed to a query database. The responses are a set of links calculated by the search engine. These systems usually limit the navigation choice and give hints, as to which are the most relevant links.

A very successful example is the Adaptive HyperMan designed by Mathé and Chen (1996). This system helps NASA Space Shuttle flight controllers access and organise large amount of information. The user can mark any part of a document as interesting and index parts with user-defined concepts. She can then retrieve marked portions of documents. This system provides long-term user models.

The Personal WebWatcher (Mladenic, 2000) based on the WebWatcher (Armstrong, Freitag, Joachims & Mitchell, 1995) is an agent that watches the user "over the shoulder", i.e. without asking the user for keywords, preferences or evaluations. It has a learning offline phase, in which it analyses requested pages of

the recent past. Adaptation consists of highlighting (annotation) anchors that the system believes will be of the user's interest. Another example of an adaptive information filtering for the WWW is the case-based approach presented by Marinilli, Micarelli and Sciarrone (1999).

### 2.8.3 Online Information Systems

Online information systems provide reference access to information on a hyperspace. This group includes e-commerce applications, recommendation systems, digital libraries, electronic catalogues and all classes of online documentation. The objective is to offer the concepts or information requested to the user in an appropriate way, i.e. according to her objectives, her background knowledge and preferences. A typical example is the administration of an adaptive bookmark menu. The bookmarks can be selected, sorted and/or annotated according to the absolute, relative or sequential frequency of the documents usage, i.e. for the entire time the system is used, in the recent past or successive use.

Knowledge management systems are also information systems but usually related to the information available in an organisation. This is the "knowledge" an organisation and its employees have to perform and to organise in their daily work. Adaptive knowledge management adapts navigation, thus reducing the whole hyperspace to the subset, which users need to accomplish their work. An adaptive knowledge management systems can adapt presentation to the background level of the user. Adaptation is mainly task-oriented.

Applications of this group differ from adaptive hypermedia systems in the educational sphere, as they do not present a systematic introduction to a learning subject. Examples of applications in this area are: PUSH, Swan, MetaDoc, AVANTI, CiteSeer and commercial products, such as Amazon.com and FindMe systems.

The Plan and User Sensitive Help (PUSH) is an information system that only adapts content to the user applying the stretchtext technique (Höök, 1998). It does not affect how the user can navigate between pages, it only affects how much information is presented within a page. The content is related to a software development method, called SDP, and consists of processes and objects.

Swan (Garlatti, Iksal & Kervella, 1999) is an adaptive and navigational Web Server for online information systems about nautical publications. The structure of its user model is similar to the Hynecosum user model (Vassileva, 1996) based on stereotypes (Rich, 1979). It consist of a user's class, a task model and an individual model. The sailor's class is used for adaptive presentation. Adaptive navigation

support is achieved by means of a task model, which uses the vessel's class, an individual model and the navigation context.

MetaDoc (Boyle & Encarnaçao, 1994) is a hypertext reading system that makes use of the stretchtext technique presenting to the user only the relevant extensions in uncollapsed form. Another example is CiteSeer, an automatic generator of digital libraries of scientific literature. CiteSeer generates a database, which downloads Web publications that responds to the user model (Ballacker, Lawrence & Giles, 2000).

The AVANTI system provides hypermedia information about a metropolitan area, such as places of interest, transportation and public services, for a variety users, including tourists, residents, elderly people, blind persons and wheel-chair-bounded people (Fink, Kobsa & Nill, 1997). The system is to be used at people's home, public information kiosks and in travel agencies. It adapts the information to the user taking into account motor functions and sensory abilities, interests and preferences, domain knowledge and competence of the user.

## **2.8.4 Online Help Systems**

Online help systems are always attached to tools or other systems, i.e. they are not independent systems. In some ways they are also online information systems, but the objective is to assist the user when she has difficulties with the tool or system. This assistance consists of presenting help information when requested (as in online information systems) and automatically recognising when the user needs some help. Adaptive online help systems have the advantage of knowing the context in which the user is working. The host system of the online help system provides information on the user's goal, thus allowing for context-sensitive help.

Greer et al. (1998) are working on peer-help systems that have been applied as intelligent help desks. A well-known but not very effective nor well-accepted adaptive help system is the Microsoft Word assistant.

ORIMUHS (Encarnaçao, 1997) is a framework for adaptive online help systems. It is based on action-level discourse management, statistical and probabilistic evaluation and user modeling techniques, such as stereotypes and overlay models. The intelligent user support is initiated and can be controlled by the end-user, but the actual presentation is computed by the system.

### 2.8.5 Personal Assistants

Personal Assistants are developed to manage a huge and dynamically changing hyperspace in a personalised way. They are helpful to users who need access to certain types of information in a hyperspace, such as the Web, frequently. Personal assistants are agents that search in a defined space identifying “helpful” information for the user. To decide which information is “helpful” for the user, the system bases itself on a user model. These agents search the information space at a certain frequency looking for new information, updating already identified items and eliminating items that are no longer up to date.

The ifWeb system is a user-model-based adaptive agent that supports user in their navigation in the WWW, i.e. an hypermedia search assistant. It performs autonomous navigation, collects documents and classifies them. The adaptive mechanism allows to maintain the model in such a way that it includes user’s interests and non-interests (Asnicar & Tasso, 97).

Hynecosum (Vassileva, 1996) is a hypermedia information system for hospitals. It is therefore also an institutional system, if we follow the classification presented by Brusilovsky (1996a). It allows adaptive navigation support for users of different level of experience. The main idea in Hynecosum is the use of task-hierarchies to restrict views of information and to make certain links visible or hide them, based on the experience level and individual needs and preferences.

*Personalised e-commerce assistants* are a special type of assistants, which aim to increase the acceptance of e-commerce offering a personalised service. Examples of adaptive electronic Web shops are AMPres (Rössel, 1998), TELLIM (Jörding, Michel & Popella, 1998), the Web Shop (Åberg & Shahmemehri, 1999) and SETA (Ardissono & Goy, 1999).

## 2.9 How to measure adaptive hypermedia systems

How can adaptive hypermedia systems be measured? Many evaluation criteria that are used in traditional software systems (ISO/IEC 9126, 1991) can be applied to evaluate adaptive hypermedia systems (Schneiderman, 1998). Other criteria are specific to hypermedia systems, such as the criteria related to nodes and links or specific to adaptive systems, such as adaptivity and adaptability. A non-exhaustive list is presented below:

- *Accessibility*: express the facility to reach the nodes.

- *Adaptability*: is the facility of an application to be configurable according to a set of decisions taken by the user, which usually define her preferences and/or background.
- *Adaptivity*: denotes the capacity of the application to alter the user model according to the user behaviour during the application run and adapt dynamically to the current state of the user model.
- *Assistance*: measures the amount of help in the form of additional information or link annotations is offered by the application to the user.
- *Availability*: indicates whether the content is updated, and whether information obtained e.g. from a database is always accessible.
- *Completeness*: measures the content for missing information and the structure for missing and dangling links.
- *Consistency*: measures the regularity of the application, i.e. similar treatment of similar aspects (at content, navigation and presentation level) and clear differences for nodes with different content, for different access structures, for different types of navigation or differences in the layout. This is considered to be the most important evaluation criteria, although it is difficult to define what a consistent hypermedia application is. In adaptive tutoring systems, for example, consistence improves quality in the same way as consistency is responsible for the success of a teaching book.
- *Functionality*: indicates how the application functions satisfy the users.
- *Implementability*: defines the overhead to providing adaptive features.
- *Maintainability*: defines the effort needed to make specified modifications.
- *Performance*: expresses the system's response time to user interaction as well as the amount of resources used by the system under stated conditions.
- *Predictability*: measures how easily the user can guess the reaction of the system to her interaction.
- *Portability*: indicates the ability of the software to be transferred from one environment to another.
- *Reliability*: measures number of crashes resulting for e.g. from SQL or JavaScript error messages or too many hits during peak periods of Web use.
- *Reuse*: defines the percentage of elements that are used for more than one purpose within the same application or in different applications. In hypermedia systems reuse means use of objects in different contexts, use

of the same interface objects or navigation elements for different nodes. Reuse promotes consistency, accessibility and predictability.

- *Richness*: denotes the amount of information nodes contained in the application.
- *Satisfaction*: shows the user's subjective impression of the adaptive system.
- *Self-evidence*: expresses how well the user can guess the meaning of the visualised content or the navigation elements.
- *Usability*: measures the effort the user needs to use the system and individual assessment of such use.
- *User-retention-overtime*: indicates how long the user remains using the application.

Studies to measure one or more of these criteria usually compare user's handle an adaptive system and its non-adaptive variant. An example is the study performed by Höök (1998) for the evaluation of the usability of the PUSH system.



*"Do we know what the user knows,  
and does it matter?"*

*Michael Ramscar, Helen Pain and John Lee,  
User Modeling, June 1997.*

## **3 User Models and User Modeling**

Paiva, Self and Hartley (1995, page 509) have defined the nature and goal of a student model as “representations of some characteristics and attitudes of the learners, which are useful for achieving the adequate and individualised interaction established between computational environments and students.” Replacing the term learner by user this definition is also applicable to a user model. A user model is constituted by descriptions of what is considered relevant about the actual knowledge and/or aptitudes of a user, providing information for the system environment to adapt itself to the individual user.

What is user modeling? What is a user model ? What is a model?

- A *model* is defined as an abstract representation of something of the real world. This representation is abstract because only some relevant properties for the application are included in the model.
- For a *user model*, the real thing is the user, who is represented as a collection of data. It is the explicit representation of user aspects. Mainly the system’s belief about the user is portrayed.
- *User modeling* is a process covering the whole life cycle of a user model including acquisition of knowledge about the user, construction, update, maintenance and exploitation of the user model.

In case of learning applications, the user model is called student model. Self (1988) defines a student model as a four-tuple of procedural knowledge, conceptual

knowledge, traits and history. Student modeling differs from general user modeling in the diagnosis and representation of subtleties on the user knowledge that are crucial to the adequate performance of an adaptive learning environment. It is more difficult to recognise changes in student knowledge, as there is usually little evidence and the learners behaviour is difficult to interpret as it responds to creative, unexpected and novel actions (Greer, 1996). Nevertheless, in this work the terms user modeling and user models are still used to refer to models that include knowledge or experience of the user. In some paragraphs student models are explicitly mentioned when the aspects described are only valid for them and not for user models in general.

Why do we need a user model? Without a user model a system will perform in exactly the same way with all users, since there is no basis to behave in a different manner. But users are different: they have different background, different knowledge about a subject, different preferences, goals and interests. To individualise, personalise or customise actions a user model is needed that allows for selection of individualised responses to the user.

Even a very simple user model can improve the usability of a hypermedia or Web application as it is shown with the SmexWeb sample application (Albrecht, Koch & Tiller, 1999). On the other hand, adaptation can produce disorientation just because of the different aspect of revisited pages. The SmexWeb framework presents a solution to this problem: *navigation consistency*. Therefore it maintains a personal history for each user including the already visited pages and the corresponding user model states (Albrecht, 1998 and Tiller, 1998).

Although user modeling offers clear advantages, many researchers are sceptical of its worth. They believe that the cost-benefit for user modeling is too high (Chin, 1993). The main difficulty of applying user modeling to systems is the acquisition of user models, i.e. the initialisation and the maintenance of the user model due to the lack of user data, the imprecise measurement of user variables and the unpredictable change of the user behaviour over time (Stein, Gulla & Thiel, 1997).

User modeling plays an important role in adaptive hypermedia systems as “Adaptive hypermedia is a direction of research on the crossword of hypertext (hypermedia) and user modeling.” (De Bra, Brusilovsky & Houben, 1999). The information kept in the user model of an adaptive hypermedia system belongs to the representation of the following user’s properties (Brusilovsky, 1996a):

- preferences,
- tasks, goals or interests,
- domain knowledge,
- experience, and

- background.

What information the user model should be designed to contain depends on the set of tasks to be supported as well as on the subject matter of the application. As already said, a user model is a representation of the knowledge and personal characteristics which the system believes that a user possesses. It is different from both the actual knowledge possessed by a user and the knowledge employed by system designers (Benyon & Murray, 1993). Systems designers form mental models of the users based on their domain studies and analysis. They use their knowledge about the users to guide their designs.

A user model can never be completely accurate, it is usually a rough approximation. Though, incomplete user models can be useful. The effort to improve accuracy has to be compared to the benefits of the improvement. Thus, user models have always to be considered in the context they will be used and the goals of the user model-based application.

The main purposes of user modeling are :

- to assist a user during learning of a given topic,
- to offer information adjusted to the user,
- to adapt the interface to the user,
- to help a user find information,
- to give to the user feedback about her knowledge,
- to support collaborative work, and/or
- to give assistance in the use of the system.

The user modeling process requires techniques to gather relevant information about the users, to construct the user model and to use it for at least one of the purposes listed above. A protocol of the user actions, interpretations of these actions or explanations of the behaviour are required for user modeling.

At present, general techniques for user modeling do not exist. Most of the existing and promising techniques have been implemented and tested in different prototypes and are often restricted to certain domains.

In the following sections of this chapter user model characteristics, foundations, classification and purposes are presented as well as user model internal organisations, techniques for acquiring data and updating processes for user models are outlined.

### 3.1 Characteristics of User Models

User models may be classified as adaptive or adaptable models (depending on who updates the user model), as being obtained by querying or observation (depending on how the update is performed), as static or dynamic (depending on when the update takes place), as grouped or individual, internal or external, explicit or implicit (depending on where it is located), as visible or opaque (depending on the visibility), as fine grained or coarse grained (depending on how detailed it is), as short term or long term user models (depending on its duration).

A user model is called *adaptive* when the model is updated automatically by the system according to the information obtained from the user's behaviour like mistakes, use of help systems or the dialogue between user and computer. *Adaptable or configurable* user models are only modified by the user and therefore it is a user controlled process, i.e. the user is who decides when the application will be adapted to the newest state of her user model. Rich (1979) calls these user models explicit and implicit. Here these terms are reserved for another classification (see below).

Information for initialising or updating a user model can be obtained by *querying* the user or by *observing* him. Greer (1996) calls these options: intrusive and passive observation. Queries may cover user's interests, preferences, expertise in some topics, knowledge of domain subjects. Observations have as subject among others user's interactions, problem's solving abilities, performance on tests of declarative knowledge and response to ability tests. Very often a combination of both techniques is chosen with the aim to acquire as much information as possible about the user.

*Dynamic* user modeling consists in dynamic knowledge acquisition of the user based on user interactions with the system. The user model is continuously updated with the acquired information. The usage of this kind of model implements dynamic applications that adapt themselves at each step to the user. In a *static* user modeling information is collected by queries or by observations and actualisation is done only in an initial phase or in regular intervals.

A user model represents *positive* or *negative* information of the user, such the adaptation in the ifWeb system (Asnicar & Tasso, 1997) is based on interests and non-interests of the user.

Models for *groups* represent user characteristics based on stereotypes while *individual* models are specially generated for each user. Often stereotyped models

are used in the initial phase and further they are refined and individualised during the interaction of the user with the system.

A user model is *internal* if it is embedded in the application; otherwise it is *external*. It may have a separate representation (*explicit*) or be included in the domain model representation (*implicit*).

User models are *visible* by the user, when the assumptions the system makes about the user is open to inspection to her. The user model is said to be *visible* and *modifiable* by the user, if she can change it explicitly (Höök, 1998 and Kay, 1995). Espinoza and Höök (1996) in her “glass box approach” stresses that the user must be able to inspect why a certain adaptation was generated, and correct the behaviour if the result is not what the user wanted. If the user neither is allowed to see nor to modify the user model, the user modeling is *opaque* to the user. The visibility requires a user interface that interrogates the user about some properties or presents the complete user model or part of it.

Another classification factor is the *granularity* (Collins, Greer, Kumar & McCalla, 1997). Some user models are described as *fine-grained* or *coarse-grained* depending on the detail level of the issues represented by the model. Some models also provide many levels of granularity by using a hierarchy.

*Short-term* models are available as long as a session lasts. *Long-term* models are kept from one session to another.

## 3.2 User Model Foundations

A user model is defined as the representation of the system’s beliefs or knowledge that the system has about the user. The notation used in this chapter is based on the formal foundation given by Self (1991), which represents a user model based on beliefs and knowledge the system and the user have. Beliefs are represented by formulas in the propositional calculus. They can be assessed as true or false. The objects of belief are called propositions. Beliefs are related to the behaviour of agents (A), such as a user (U) or a system (S). In the remainder of this section definitions are restricted to one user, but they are applicable to any number of them.

To formalise beliefs let  $p$  be a proposition, then  $B_{Ap}$  holds if agent A believes  $p$ .

$B_A = \{p \mid B_{Ap}\}$  is the set of beliefs of the agent A

As  $B_{AP}$  are propositions itself, that means that beliefs can be nested. Then,

$B_{SU} = \{p \mid B_S B_U p\}$  is the set of propositions the system  $S$  believes are believed by a user  $U$ .

But as the user's beliefs are unknown for the system, all reasoning has to be done on the system's beliefs. More interesting are the subsets of system's beliefs about the user; they constitute the user model (UM). To give a formal description of a user model, let  $p(U)$  denote a proposition related to the user  $U$ .

$UM = B_S(U) = \{p \mid B_S p(U)\}$  is the set of propositions the system  $S$  believes about the user  $U$ .

Belief could be replaced with knowledge as it can be described in terms of beliefs: an agent knows  $p$  if  $K_{AP} = B_{AP}$ ,  $p$  is true and there is a source for knowing  $p$  is true.

For separating domain dependent and domain independent aspects of a user model, it is useful to distinguish among the following types of propositions:

- propositions related to the domain, which a student acquires in a learning system (problem knowledge) or which are subject for searches (**D**),
- general propositions that are domain-independent, also called background knowledge (**I**), and
- propositions that describe cognitive or personal characteristics of the user or of her actions, also known as behavioural knowledge, such as preferences, tasks, goals and experiences (**C**).

Student models focus on domain-related propositions, while general user models are built mostly on behavioural knowledge.

Thus,  $D_S$  and  $D_S(U)$  can be defined as follow:

$D_S = \{p \mid B_S p \wedge p \in D\}$  is the set of propositions related to the domain  $D$  that the system  $S$  believes

$D_S(U) = \{p \mid B_S p(U) \wedge p \in D\}$  is the set of propositions related to the domain  $D$  that the system  $S$  believes the user  $U$  has.

$I_S$ ,  $I_S(U)$ ,  $C_S$  and  $C_S(U)$  can be defined in analogy.

As every proposition believed by the system belongs to one of these three types (D, I or C), then

$$\mathbf{B}_S = \mathbf{D}_S \cup \mathbf{I}_S \cup \mathbf{C}_S \quad \text{and} \quad \mathbf{UM} = \mathbf{D}_S(\mathbf{U}) \cup \mathbf{I}_S(\mathbf{U}) \cup \mathbf{C}_S(\mathbf{U})$$

Giangrandi and Tasso (1997) include a temporal aspect describing the status and the evolution over time of a temporal student model. It allows to monitor the learning process in a more fine-grained way. By the introduction of temporal constraints they handle the problem of possible conflicts in the student's beliefs.

### 3.3 Types of User Models

The criteria that are most frequently used to classify user models are: the nature of the contents, the type of representation and the methods used to initialise, construct and exploit user models. In this section only a user model classification based on the contents is presented. The next sections outline the other criteria.

Three types of content-based models are distinguished: the domain-knowledge model, the domain-independent-knowledge and the psychological or cognitive model. Other names for these models are: student model, background-knowledge, and user profile (Murphy & McTear, 1997).

The *Domain-knowledge model* ( $\mathbf{D}_S(\mathbf{U})$ ) contains knowledge the system assumes that the user has about the domain. In addition, a sub-classification of propositions contained in the models may be done. Benyon and Murray (1993) distinguish three levels within this model: task, logical and physical level. The task level describes user goals in the domain. The logical level records what the system believes the user understands about the logical concepts embodied in the domain and the physical level records the user's inferred knowledge. (e.g. task level: paragraph formatting in a text editor, logical level: meaning of paragraph, physical level: how to perform indentation).

Vassileva (1990) also distinguishes between general characteristics, which describe or evaluate the user domain knowledge and the actual domain knowledge given by facts, procedures, misconceptions and mal-rules.

Ragnemalm (1995) distinguish two types of  $\mathbf{D}_S(\mathbf{U})$ : *Subject-matter* and *Pedagogic-content* model. The former corresponds to a domain knowledge base. The latter is organised according to instructional goals storing evaluations of the user knowledge. The knowledge base can include correct as well as incorrect knowledge (misconceptions).

The *Domain-independent-knowledge or Background-knowledge Model* ( $I_S(U)$ ) contains general or not domain-specific knowledge as well as areas of interests and background of the user. This data is not of psychological nature but may be important for the overall user model.

The *Psychological or Cognitive Model* ( $C_S(U)$ ) is concerned with preferences, (dis-)abilities and personality traits. For example the system has to know the student's preferences (Vassileva, 1990; Ragnemalm, 1995) if it adapts its instruction to:

- the learning style or strategy preferred by the user (holistic vs. serialised, exploratory vs. directed learning),
- the motivation technique that is more effective for the user (curiosity, competition or confidence),
- the type of thinking – inductive (learning by examples) or deductive (logical deductions, problem-solving),
- the degree of concentration (recognised e.g. by typing errors, misuse of commands).

Users that differ in these cognitive skills require different adaptations. Usually the characteristics of cognitive model are long-term characteristics as they are more resistant to changes.

If the user's model is represented as a subset of the expert model then it is called an *overlay model*, i.e.  $UM \subset B_S$ . An overlay model only allows to diagnose missing pieces of knowledge, but cannot represent deviations from correct knowledge or bugs (Sleeman & Brown, 1982).

Another approach, also based on the nature of the contents, is the modeling of characteristics and misconceptions. Models that also include propositions about misconceptions or bugs are called *Perturbation Models* or *Bug-models*. Here it must be distinguished between misconceptions, bugs and errors. A misconception is a discrepancy between the system's and the user's representation of knowledge, a bug (sometimes called mal-rules) refers to a discrepancy at the behavioural level and an error is the difference between the user's and system's behaviour related to the problem domain. Bugs or erroneous sequences are sometimes given or automatically generated by the system using some rules or mechanisms (Vassileva, 1992).

To illustrate the above definitions some propositions related to the SmexWeb adaptive hypermedia exercising application (see Chapter 8) on the subject EBNF (Enhanced Backus-Nauer-Formalism) are chosen. EBNF is a grammar-like



formalism used for describing the syntax of programming languages. It is a topic of introductory courses in computer science. Therefore, students may have little experience with computers in general and browsing in particular. This adaptive Web-based application was implemented on the basis of the framework SmexWeb (Albrecht, Koch & Tiller, 1999) to assist students with different background in solving EBNF exercises.

The pedagogic-content model contains the EBNF-rules to built correct EBNF expressions and pedagogic rules useful to teach EBNF. The domain-independent-knowledge model consists of a list of propositions related to general computer and browsing knowledge. The cognitive model describes some learning and layout preferences of the user. Some of the propositions of each model are listed below.

$D_S = \{ \text{EBNF rules are of the form non-terminal = expression,}$   
 expressions that comprise terminals and non-terminals,  
 EBNF language comprise expressions without non-terminals,  
 the symbol “|” is used for alternatives,  
 the symbol “\*” indicates any number of repetitions,  
 the symbol “+” indicates at least one repetition,  
 distinction between “\*” and “+”, ... }

$I_S = \{ \text{any link can be followed,}$   
 help can be requested at any time,  
 experience in browsing, ... }

$C_S = \{ \text{preference of explanations with examples,}$   
 small font, non-formal details, ... }

$B_U$  contains propositions believed by the user, which are related to EBNF, to browsing and the user preferences. The  $UM$  consists of the propositions the SmexWeb system believes about the user and that are related to the mentioned subjects. It can be observed that there are some differences in both models. First, the user considers herself an expert in browsing while the system has categorised her as fairly good. Second, the user has an erroneous concept about the EBNF language.

$B_U = \{ \text{EBNF rules are of the form non-terminal = expression,}$   
 EBNF language comprise expressions without non-terminals,  
 the symbol “|” is used for logical “and”,  
 expert in browsing,  
 explanations with examples,  
 small font, ... }

**UM** = {EBNF rules are of the form non-terminal = expression,  
 EBNF language comprise expressions with terminals  
 and non-terminals,  
 the symbol “|” is used for logical “and”,  
 fairly good experience in browsing,  
 explanations with examples,  
 small font, ... }

### 3.4 Objectives of User Modeling

Applications of different kind can profit from user models. Traditionally, user models are found as a component of learning systems. User modeling is usually tightly coupled with the instructional component of an adaptive learning environment or plays an important role in user assistance, such as in search engines or help systems. User models are included now in a wider spectrum of applications. The following is a non-exhaustive list of objectives for which user modeling is used for: help users to learn about a topic, to support collaboration and assistance, to find and tailor information and/or to improve man-machine communication.

#### 3.4.1 Help to Learn

The instruction and training field is one of the major applications fields for user modeling. Learner or student models are built to support the systems in e.g. choosing suitable learning material, selecting appropriate exercises, differentiating skills of trainees, providing feedback on their knowledge or predicting and correcting student’s answers.

The learner model in an instruction or training system aims at identifying misunderstandings and support the student in the learning process eliminating these misconceptions.

Some adaptive learning environments need to support *dynamic planning* (Vassileva, 1995 & Wasson, 1990), i.e. the online creation and revision of the instructional plan according to the user’s behaviour. Plans can not be created from scratch, that means there are always a set of assumptions made. Therefore, plans are only dynamic to a certain degree. A mapping function can be defined that creates a new user model based on the current user model and the users inputs as well as the system’s outputs.

$$\mathbf{UM} \times \{i_1, o_1, i_2, o_2, \dots, i_n, o_n\} \rightarrow \mathbf{UM} \quad (\text{a})$$

where  $\{i_1, o_1, i_2, o_2, \dots, i_n, o_n\}$  is a sequence of events: inputs (i) and outputs (o). Usually  $n$  is kept small to reduce complexity. As the user's inputs are not known in advance, some kind of predictions have to be made. The plan may also depend on other factors, such as constraints or organisation of the subject matter.

*Diagnosis* is the process of finding and interpreting student's misconceptions or bugs. Planning and diagnosis are in some cases very closely related. The results of the diagnosis are represented in the user model and are the subject of the remediation.

*Remediation* can be performed in different ways. From the pedagogical point of view it has to be determined, which is the appropriate form. These forms are for example:

- *re-teaching*: is an option to be applied when the system can not exactly identify the user's difficulty, e.g. a set of concepts can be re-thought.
- *emendation*: if the system has identified a proposition  $p$ , where the user believes a faulty version of  $p$ , then the system can deny the faulty version of  $p$ , assert  $p$  and justify  $p$ .
- *exemplification*: the system can generate examples that address the missing or faulty knowledge of the user.

### 3.4.2 Support Collaboration and Assistance

In a collaborative environment, where the user has greater freedom to select her own goals, the role of the system is different. It plays the role of an assistant, offering hints, helps and comments. The system is in this case a co-operative partner or adviser and no longer a traditional tutor. Emphasis is put on goals, tasks, and interests of the user, instead of knowledge. In applications supporting collaborative learning or work the user model is built by a more direct interaction with the user rather than through some internal system reasoning.

A mapping function, as shown in (a), is applied in this case to obtain an updated user model. It takes into account the last  $n$  inputs from the user. The number of parameters of the function may be restricted to  $n = 1$ , i.e. the system in this case does not predict or plan at all.

$$\text{UM} \times \{i_n, i_{n-1}, i_{n-2}, \dots, i_1\} \rightarrow \text{UM} \quad (\text{b})$$

### **3.4.3 Find and Tailor Information**

Examples of user models used for finding and tailoring information for the user are: reminding users of previous Web navigation paths (Maglio & Barret, 1997), recommending Web pages, selecting documents of interest to the user, filter documents, such as e-mails or articles according to the user's interests (Cas & Bingler, 1998), select appropriate collaborators (Collins, Greer, Kumar & McCalla, 1997), recommend form of collaboration between students (Bull & Smith, 1997), guidance in solving exercises (Albrecht, Koch & Tiller, 1999), etc.

### **3.4.4 Improve Man-Machine Communication**

User modeling is also an important subject in the human-computer communication field. The dialogue management and the natural language generation are own research areas. Dialogue modeling is traditionally motivated by the assumptions that interaction is not an arbitrary exchange of messages, but that it follows certain rules and can be described by certain patterns. For example, Miracle is a logic-based information retrieval system that builds a dialogue model which describes potential development of the interaction recommending problem-solving steps (Stein, Gulla & Thiel, 1997).

## **3.5 Initialisation of User Models**

The first time a user runs an adaptive application the user model is empty. A user model can be initialised in two ways: by explicit questioning or by default assumptions. Most of the times a combination of both is implemented.

### **3.5.1 Explicit Questioning**

The system can acquire initial knowledge about the user asking her to fill in a questionnaire. The problem that arises is how many questions the user would be willing to answer. It is not possible to give a unique response as it depends on others on the application and the type of users.

If there is a finite set of independent propositions  $\{p_1, p_2, p_3, \dots\}$  such that  $B_{sp_i}$  may be true, the system may ask the user questions such as "Do you believe  $p_i$ ?" or "Do you know  $p_i$ ?". Hence the user is asked to indicate through the dialogue an

assessment of interests, estimation of her capabilities determining a first domain knowledge, etc. If the propositions are not independent, an appropriate order in the questions has to be determined (Self, 1991).

Some answers can be used to deduce an increment of the system's knowledge. Therefore, *inference rules* are defined as follows:

$$B_S K_u p_i \rightarrow B_S K_u p_j \text{ for some pairs of } i, j$$

where  $B_S K_u p_i$  is the system's belief about the user's knowledge about  $p_i$ .

Questions can also be related to domain independent concepts as well as to cognitive characteristics. These questions may be restricted to one proposition  $p_i$ , such as "Do you prefer material with examples?" or related to a set of classes of users "Do you believe yourself to be an expert, novice or beginner?" These classes permit through application of inference rules the determination of propositions.

$$B_S c_j \rightarrow B_S K_u p_1 \wedge B_S K_u p_2 \wedge B_S K_u p_3 \wedge \dots$$

The user model relies on the user input, i.e. on her own assessments and beliefs about her knowledge, which may not always be correct. The initial interview is a primary and valuable source of information about the user. Sometimes it is used to assign the user to certain classes, as mentioned above. These classes are so called stereotypes. In the remainder of this work they are called *user stereotypes* to mark a difference to UML stereotypes used in Chapter 6.

### 3.5.2 Default Assumptions

If no explicit questioning is possible or only a limited set of questions is tolerable, but more information is needed, then some default assumptions can be made. For example, the assumption that a user of a learning system knows nothing of the subject to be learned:

$$\forall p (p \in D_S(U) \wedge p \rightarrow B_S \neg K_u p)$$

or every user that uses a certain system responds in a first phase to some class of user (stereotyped), as for example to be assisted with help information without explicit request until she gives a signal to reduce this assistance.

$$\forall U (p = \text{„requires help“} \wedge p \in I_S(U) \wedge B_S B_u p)$$

If user stereotypes are arranged into a hierarchical structure, it is possible to organise sequences of questions and inheritance of inference rules. A set of propositions can be associated to each node, which represents a user stereotype. If a user is identified or assumed to belong to a user stereotype group, then all the propositions of the node are included in its model and she inherits all propositions of the parents nodes (stereotype activation). Figure 3-1 shows an example of a user stereotype hierarchy.

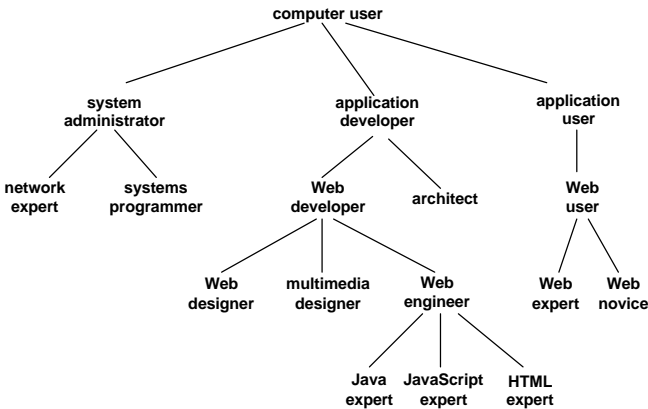


Figure 3-1: A User Stereotype Hierarchy

*User stereotypes* are useful in the initialisation process of user models, sometimes during the whole user modeling process. But they do not offer enough information in the case of student models because they do not permit the necessary fine-grained analysis. User stereotypes for student models are seldom used beyond the initial stage. E-commerce applications often use user stereotypes that are based on predefined user segmentation groups, such as the Internet Values and Lifestyles (iVALS, 2000).

### 3.6 User Model Internal Structure

The heart of students models is a set of concepts (also called topics, knowledge elements, propositions, objects) mostly related to the domain. Depending on the type of relation between these concepts Brusilovsky (1996a) distinguishes between *level one*, *level two* and *level three* models. Level one is the simplest form and is a

set of independent concepts referred to as the *domain model*. If the topics are linked to each other thus forming a kind of *semantic network*, this network domain model is said to be a level two model. Level three is a frame-based domain knowledge representation, where each topic is represented as a set of attributes with the particularity that different kinds of topics usually have different sets of attributes.

These domain models provide the structure for the user model. In this context the knowledge a system has about the user can be represented in different ways, such as overlay models, semantic nets, user profiles, stereotype-based models, bayesian networks or fuzzy logic models. The most frequently used representations are overlay models, stereotyped models and bayesian networks.

### 3.6.1 Overlay Model

In an overlay model the user's knowledge is represented as a subset of the system's knowledge. The system's knowledge may be given by the expert knowledge, the domain knowledge, the expected student knowledge or a perturbation model. The overlay model is the most predominant type of user model usually represented as a hierarchical or semantic network of nodes related directly to the domain concepts. Boolean or discrete values are used as estimation for the user's knowledge. The value of overlay models has been questioned and criticised, such as by Self (1991) and Vassileva (1990). Others, such as Brusilovsky (1996a) stresses that overlay models are powerful and flexible as they can measure independently user knowledge on different topics. One way of implementing overlays is to assign a numerical weight to each concept in the curriculum, which indicates how sure the system is that a user knows the concept.

There are many examples for overlay models in the literature; the following systems organise their user models as overlay models including not only domain knowledge but also user tasks and goals: Orimuhs (Encarnação & Stork, 1996), PUSH (Espinoza & Höök, 1996), HyperTutor (Gutierrez, Perez, Usandizaga & Lopistéguy, 1996), ELM-ART (Brusilovsky, Schwarz & Weber, 1996a), HYNECOSUM (Vassileva, 1996), ADAPS (Brusilovsky & Cooper, 1999), etc.

For example the student model of HyperTutor contains information about the student learning characteristics, the domain knowledge, the didactic material and about the learning process history. Also the media preferences for learning (video, sound, animation, etc.) are considered as part of the learning characteristics. The domain knowledge contains the concepts acquired by the student during the learning process. It is represented as an extended overlay model of the pedagogical

domain. The learning date of a concept is also stored; it allows the system to know in which order the concepts are learned and how much the student may have forgotten according to the time elapsed since then. Information about the didactic material presented to the student is kept in order not to repeat examples or exercises. Recording the history of the learning process allows the system to know what happened in the last session and over the whole student learning process.

### 3.6.2 User Profile

The terms user model and user profile are often used as synonymous. But sometimes they are distinguished to indicate that user profiles are simple user models. They are used to represent user's cognitive skills, intellectual abilities, intentions, learning styles or preferences.

In a user profile each aptitude is assigned to a value; these values usually belong to a range of valid values. The values can be a boolean (true for *known* and false for *unknown*), discrete (e.g. 1 for *low*, 2 for *middle*, 3 for *high*) or probabilistic value (e.g. 0 for *none*, 0,5 for *some*, 1 for *all*). Therefore the model of the user can be represented by pairs of "topic-value", one pair for each concept. Every representation with a finite number of values for each aptitude can be converted to one with only boolean values (Kay, 1995 and De Bra & Calvi, 1998). These aptitudes may be transient (e.g. interest in politics, no Web experience) and some permanent (e.g. blindness, age). Some may be situational-dependent (e.g. preference for audio).

Examples of systems that implement user profiles are the EPK – an electronic product consulting system – developed by Timm and Rosewitz (1998) and SmexWeb – Student Modelled Exercising System (Albrecht, Koch & Tiller, 2000).

### 3.6.3 Stereotyped Model

In a stereotyped model the properties and knowledge of the users are also represented with pairs of item and value. The difference is that different combinations of pairs are assigned to stereotypes, such as novice, intermediate, expert. One user then inherits all the properties defined for one stereotype. Stereotyping introduced by Rich (1979) is simpler but less flexible and powerful than other user modeling techniques.



Examples of systems that use stereotyped models are the tourist information system developed by Schuhbauer (1998), the CALL system (Murphy & McTear, 1997) and GRUNDY of Rich (1979).

Good results are obtained with the combination of stereotyping techniques and overlay modeling. The initialisation of the user model is done by assigning a stereotype to the user, which is refined at each step implementing the overlay model. An example is the user model implemented in ARCADE (Encarnação & Stork, 1996).

Stereotype models are enough when modeling the interface or choosing the type of instruction. They are insufficient when individual adaptation requires a more fine-grained description of the user or specific help or advising is required.

### 3.6.4 Bayesian Networks

Numerical techniques have become more popular in the last decade for modeling user's knowledge, user goals, recognising plans and identifying the best actions to take under uncertainty. One of these approaches used to manage uncertainty in user modeling is Bayesian Networks (BN).

A Bayesian network is a directed, acyclical graph in which the nodes correspond to variables (user properties) and the *links* correspond to probabilistic influence relationships (Jameson, 1998). These variables may belong as well to the domain-knowledge, background-knowledge and/or cognitive model. Each *node* represents the system's belief about the possible values (levels, states) of the variable. Thus, the conditional probability distribution (CPD) must be specified at each node. If the variables are discrete, they can be represented as a table, which lists the probability that a child node takes on each of its different values for a given combination of values of its parents.

For example the following node represents the User knowledge of the concept "inheritance" in object-oriented techniques; possible values are *known* and *unknown*. This node is a child of User expertise in object-oriented software development and Difficulty of the concept "inheritance". The states for the parent node User expertise in object-oriented software development can be defined as different levels of expertise such as *expert*, *intermediate*, *beginner*, *novice* (see Figure 3-2).

Links between nodes are defined to infer about the system's belief about the user. To derive beliefs for *known* and *unknown* the network requires a conditional probability table that specifies each of the 24 combinations of possible values of

the variables in the parent nodes and the child node, how likely the value of the child variable is, given the values of the parent variables.

Where do these probabilities come from? The difficulty lies in specifying these conditional probabilities. They can be derived from empirical data, estimated by a domain expert, and/or based on a more general theory about the relationship among variables of these types. General theories applicable to specify conditional probabilities can be found, for example, within the psychological test theory (Jameson, 1995). Some frameworks also allow conditional probabilities to be characterised as unknown parameters (Mislevy & Gitower, 1995). But in most of the systems these numbers have been entered by the designers themselves on the basis of intuitive judgement. Most of the researches do not indicate how accurate numbers can be obtained in practice.

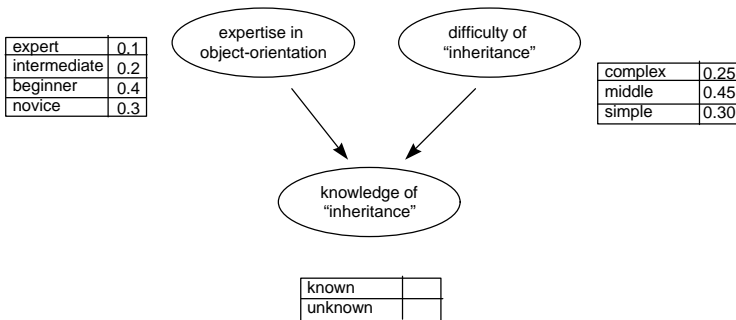


Figure 3-2: Bayesian Network

Applying Bayes' Theorem in its pure form becomes unmanageable as the number of variables in a problem increases. Different kinds of algorithm improvement's have been tested in many applications. Computation techniques for bayesian inference networks are discussed by Jameson (1995). There are many examples of adaptive systems based on user modeling relying on bayesian networks, such as the KBS hyperbook system (Henze & Nejd, 1999), the agents of Horvitz (1997) and the systems mentioned in Jameson (1995).

The updating process requires to acquire information about the user's behaviour and to adjust the user model if there is evidence that it is inaccurate. The user model is inaccurate if the user acts in a different way than predicted by the user model.

The acquisition is the process of collecting input from the user in whatever form it is available – mouse click, typed or voice input, screen touch, time elapsed – corresponding to user interactions in a process, pages visited in a hypermedia application, steps in a student's problem solving, etc. Based on this information the aim is to infer about what the user knows, or does not know, what she prefers or aims at. The problem is the interpretation of the data (mouse click, keyboard input, etc.) into actions or propositions, which is not trivial (Ragnemalm, 1995).

### 3.7.1 Acquisition techniques

The user model acquisition techniques can be characterised along several orthogonal dimensions, as follows (Chin, 1993):

- *Active or passive*, based on the participation of the user in the acquisition.
- *Automatic or user initiated*, based on who is the initiator of the acquisition.
- *Direct or indirect*, depending on the length of the inference chain.
- *Explicit or implicit*, based on the type of user feedback.
- *Logical or plausible*, according to the results produced.
- *Online or offline*, based on when the acquisition is performed.

Active techniques interact directly with the user, e.g. using online forms via CGI-scripting to query the user. Passive techniques, instead, construct user models based on inferences from observations, such as visited pages or the analysis of the user clickstream; information is obtained from HTTP log files, CGI data, cookies or Java applets. SmexWeb (Albrecht, Koch & Tiller, 1999), for example, uses both techniques; an active one for the initialisation of the model and a passive acquisition technique for the update of the user model.

User initiated are those techniques in which the user decides when to change the user model. In automatic techniques, instead, the user has no influence on when she is observed and when the model is updated. System initiated techniques are the more frequently used. The UM toolkit (Kay, 1995) is an exception that allows users to build a model in a graphical format.

Explicit techniques are those in which the user consciously provides information. Implicit techniques instead are based on unobtrusive observation of the user behaviour.

Logical and plausible techniques differ on the degree of plausibility of the results. Plausible techniques require the explicit representation of uncertainty in the user model and need mechanisms to maintain consistency in the user model. Uncertainty in user modeling can be managed, e.g. by Bayesian networks (Jameson, 1995). Overlay models, instead, are an example for the result of a logical acquisition technique.

An acquisition technique is direct, if the system derives directly from the user feedback information that is used for the update of the user model. Indirect acquisition techniques build upon the results of direct ones; they often take the form of inference rules. One of the most well known inference rules are those to define stereotypes. Stereotypes are widely used after they have been introduced by Rich (1979) in the GRUNDY system.

Most of the acquisition techniques are applied online. Exceptionally, users can be observed offline, to infer stereotypes, e.g. extracting information of certain customer databases. It is questionable if this is an appropriate acquisition technique for user modeling, since the information obtained from the databases may not correspond to real users of the adaptive system.

### 3.7.2 Acquisition process

The acquisition process consists of three phases: *collection*, *diagnosis* and *consistency*. In the *diagnosis* process two steps can be distinguished: *transformation* and *evaluation*.

#### Collection of data

The main problems related to the *collection* of data are: the reliability of the data, the amount of data available and the level of detail of the data. What amount of data is needed depends on the granularity of the model. In hypermedia-based systems there is the additional difficulty to register user interactions mainly due to the stateless Hypertext Transfer Protocol (HTTP), which provides little support in the process of collecting data (Ragnemalm, 1995). Thus, data is obtained through additional channels of communications.

Granularity or level of detail of data vary from application to application. For example, the following states of level of detail are defined for student models: final, intermediate and mental to express just the result of final exercise, intermediate results or every step during solving the exercise.

---

### **Diagnosis**

As mentioned above, the diagnosis is the process of finding faults. It consists of two steps: a *transformation* of the collected data and a comparison (*evaluation*) of the resulting user's behaviour with some "correct" behaviour.

*Transformation* consists of extracting the relevant information from the data collected in order to judge the user's skills. This can be done in two directions according to Ragnemalm (1995) and Vassileva (1990):

- the user's input from her behaviour can be converted to a representation closer to the model. The techniques used for this conversion may be domain-dependent.

User's input has to be mapped into a set of propositions (Self, 1991). The problem is to find a function:

interpret  $(\{i_1, i_2, i_3, \dots, i_n\}) = \{p_1, p_2, p_3, \dots, p_m\}$  such that

$B_s B_{Up_j}$  for  $j= 1$  to  $m$

In the simplest case the function is the identity, thus  $i_j = p_k$ . But the interpretation is usually more complex as it has to reason about what the inputs mean in terms of user's beliefs. These interpretations are very often domain or application dependent. Some general *acquisition rules* can be formulated, such as if the user states a proposition then the system believes that she believes it and all components of it (Kass, 1989).

- the user's properties contained in the model can be converted to a format closer to the user's input, i.e. giving criteria or recognition patterns for the user's input. They can be given in advance or generated by the system. Handling of contradictions, misconceptions and instability of user and student models are aspects to be considered.

It can be defined in analogy to the deduction of recognition patterns:

deduce  $(\{p_1, p_2, p_3, \dots, p_m\}) = \{r_1, r_2, r_3, \dots, r_n\}$  such that

$B_s B_{Up_j}$  for  $j= 1$  to  $m$

and the set of recognition patterns  $\{r_1, r_2, r_3, \dots, r_n\}$  is compared during *evaluation* with the user's input  $\{i_1, i_2, i_3, \dots, i_n\}$ .

*Evaluation* refers to the process of comparing the user's behaviour or knowledge to some conception of "correct" behaviour or knowledge, which is represented explicitly or implicitly in an expert model.

**Summarising:** the diagnosis process aims at matching data (i) with a model (UM) or inversely to match the model to the data. Thus, diagnosis processes are ranged between two extremes:

- a purely data-driven approach: the diagnosis is built based on the user's behaviour without reference to a predefined model,
- a purely model-driven approach: this method generates predicted models and matches them to the user's behaviour.

Data-driven approaches are appropriate for simple domains, more for user than learner models. Model-driven models easily result in problems with combinatorial complexity. Most *diagnosis methods* are in between these two extremes, such as reconstruction, model tracing, or condition induction. A brief definition is given of some of them. The following works can be referred to for a detailed description of diagnosis methods: Self (1991), Dillenbourg and Self (1990), Jameson (1998) and Ragnemalm (1995).

- *Reconstruction* is the inferring process that tries to reconstruct a set of propositions the user "has used" for an answer "a" (answer for a solution with no intermediate steps). This is based on the assumption that the user uses only knowledge that is included in  $\mathbf{D}_s(U)$  and provided that there is a unique set of  $\{p_i\}$ .

reconstruct (a) =  $\{p_1, p_2, p_3, \dots, p_m\}$  such that  $p_i \in \mathbf{D}_s$  for  $i = 1$  to  $m$

This method is applied in Web systems, where the system only receives information from the user when the user activates a link. Intermediate steps or inputs are inaccessible to the system.

- *Model tracing* is a technique that can be chosen if more than one set  $\{p_i\}$  is given, such that  $\{p_1, p_2, p_3, \dots, p_m\} \rightarrow a$ . It assumes that the user input reflects the mental steps in the user's problem solving process. Model tracing selects states that can possibly be applied and transforms them for comparison with the user input states. As input for the user mental steps it is required an extremely detailed procedural user modeling. Classical examples are the Lisp Tutor (Anderson & Skwarecki, 1986), the ACT Programming Tutor (Corbett & Anderson, 1995) and PAT Online (Ritter, 1997).

The EBNF application mentioned before implements applets, that for example permit to record every movement the user performs during the construction of EBNF expression.

$$\text{match} (\{\text{step}_1, \text{step}_2, \text{step}_3, \dots, \text{step}_k\}, \{p_1, p_2, p_3, \dots, p_k\})$$

for  $k \leq m$  and for each candidate sequence  $\{p_i\}$ .

It is not necessary to wait for the complete user's input sequence to begin the comparison with the propositions. Based on this comparison the system determines the matching propositions and can predict next steps. This model tracing process has no advantages unless the user makes mistakes (useful for student models). As a result of this process a set of propositions  $\{p_i\} \in D_S$  is added to the UM fulfilling  $B_S B_{U p_i}$ .

If the user model is an *overlay model* then,  $\forall i (B_S B_{U p_i} \rightarrow p_i \in D_S)$ .

- *Condition induction* is similar to model tracing as it assumes that the user input is described with mental steps and the model is represented by propositions. Difference is that instead of looking for an appropriate proposition from the user input a proposition is generated and then compared to the propositions of the domain model. The direction is the opposite of the model tracing (Rangemalm, 1995).

interpret  $(\{i_1, i_2, i_3, \dots, i_n\}) \rightarrow \{q_1, q_2, q_3, \dots, q_k\}$  and then

$$\text{match} (\{q_1, q_2, q_3, \dots, q_k\}, \{p_1, p_2, p_3, \dots, p_k\})$$

for each candidate sequence  $\{p_i\}$

- Other techniques such as *Decision Trees*, *Generate and Test* as well as *Interactive Diagnosis* use knowledge about misconceptions, bugs and mal-rules to predict student's input. Dillenbourg and Self (1990) describe these concepts on the basis of discrepancy between the system's knowledge and the knowledge about the user represented in the user model. Misconceptions refer to a discrepancy in the conceptual level while bugs, also called mal-rules, are differences in the behavioural level.

Different techniques are used to construct and update probability-based inference networks, such as deductive, inductive and abductive reasoning described by Jameson (1995) and Mislevy and Gitower (1995) .

- *Deductive reasoning* flows from general to particulars, within an established framework of relationships among variables – from causes to effects, from knowledge to observable behaviour. It is also called predictive inference.

- *Inductive reasoning* flows in the opposite direction – from effects to possible causes, from user’s solution to likely configurations of knowledge. It is also called diagnostic inference.
- *Abductive reasoning* generates new hypotheses, new variables or new relationships among variables from observations. Model improvement, i.e. modifying the network in response to unexpected or unsatisfactory outcomes, requires abductive reasoning.

---

### **Model’s consistency**

The maintenance of the model’s *consistency* is a subject that is not so intensively researched as the diagnosis techniques are. When incorporating new propositions to the user model, some inconsistencies with already existing conceptual or behavioural propositions may appear. One interesting approach is presented by Huang, McCalla, Greer and Neufeld (1991). When conflicts are detected among the assumptions about the user, their systems determine which assumptions should be retracted to resolve the conflict.

## **3.8**

---

### **Sharing User Models**

Kobsa and Wahlster (1989) stress that a user model is a knowledge source which is separable by the system from the rest of its knowledge. Adaptive Web systems can access through the World Wide Web to external user models, but in practice it has been proven to be difficult to find domain problems sharing a user model. Some research has been done in this direction of shared or server user models.

BGP-MS (Belief, Goal, Plan Management System) and “Doppelgänger” (Wahlser & Kobsa, 1989) are examples of server systems developed for user modeling, that offer their services to a number of applications. BGP-MS is a prototypical implementation of the AsTRa (Assumption Type Representation) framework for logic-based user model representation and reasoning (Pohl, 1999).

Fink (1998) sees the following as the main advantages of the central maintenance of user modeling activities compared to local and application-oriented user modeling scenarios:

- current user model information is available to every application,
- synergistic effects with respect to acquisition and usage,
- low redundancy in individual user model content,



- availability of stereotypes for different applications, and
- design, implementation and maintenance can be performed independently of the application.

Paiva & Self (1995) present the workbench TAGUS that aims at providing a set of services to be used by applications and by user model developers. It has identified the basic mechanisms in user modeling and establishes general modeling cycle. This cycle includes two main stages: the acquisition activities and the maintenance of the model. The architecture of TAGUS is composed of a User or Learner Model (ULM), a set of maintenance functions, an acquisition engine, a reason maintenance system, a meta-reasoner and two interfaces. This user model server is independent of the applications that use it. The remaining problem is the transfer of information between the application and the ULM as the user or learner model depends usually on the domain. TAGUS is only partially general as it is useful for applications that satisfy some constraints as to communicate with the application using a language especially defined for this purpose. Other shell systems are UMT (user modeling tool) implemented by Brajnik and Tasso (1992) and GUMAC (General user model acquisition component) presented in Kass and Finin (1988) and the AsTRa (Assumption Type Representation) framework described by Pohl & Höhle (1997).

Many commercial user modeling servers are available. A list of some major players as of press time is shown in the article of Waters (2000). Fink and Kobsa (2000) present and discuss a group of selected user modeling servers: Group Lens, Personalization Server, Learn Sesame and FrontMind. They describe among others the architecture, the functionality, data acquisition, representation methods, software and hardware platforms, the company and product profile.

### **3.9 Development of User Models**

Part of the software engineering process for adaptive hypermedia is concerned with the development of an appropriate user model for the application. The classification and description of characteristics of user models done in this chapter are used as basis for the definition of a development process for user models (Chapter 6 and 7). The structure of a general user model is based on the division of the user model in three sub-models as suggested in Sub-section 3.3.



*“..an attempt to capture, both formally and informally,  
the important abstractions found in a wide range of  
existing and future hypertext systems.”  
Frank Halasz and Mayer Schwartz  
NIST Hypertext Standardization Workshop,  
January 1990.*

## **4 An Object-Oriented Reference Model**

In this chapter a reference model (metamodel) for adaptive hypermedia applications is presented. It is called the Munich Reference Model for Adaptive Hypermedia Systems, continuing with the tradition to choosing names of places for the reference models related to the hypermedia field, such as in the case of the Dexter Reference Model for Hypertext Systems (Halasz & Schwartz, 1990), the Amsterdam Hypermedia Model (Hardman, Bulterman & van Rossum, 1994) or the Dortmund Family of Hypermedia Models (Tochtermann & Dittrich, 1996). An exception to the rule is the Adaptive Hypermedia Application Model (AHAM) of Wu, Houben and De Bra (1998), which does not include the name of a city in the title. AHAM is a first approach to a reference model for adaptive hypermedia applications, and is, to author’s knowledge, the only one that has been published to date.

The model presented in this work is a Dexter-based reference model, similar to most of the models mentioned above. It augments the Dexter model with features supported by existing adaptive hypermedia systems or systems under construction. It improves the AHAM approach by including a detailed specification of the user model and the adaptation model (called the teaching model by AHAM). The Munich Model focuses on object-orientation, visual representation and formal specification.

The Dexter Model was formalised in the specification language Z, a specification language based on set theory (Spivey, 1992). Since then, object-oriented models and programming have increased in importance and dissemination. In addition,

more emphasis is now put on visual modeling languages, which improve intuitive comprehension of models.

The Munich Reference Model is based on the Unified Modeling Language (UML) and the Object Constraints Language (OCL). “The UML is a language for specifying, visualising, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems” (OMG, 2000). UML provides the notation and the object-oriented modeling techniques for the visual representation of the reference model. OCL (Warmer & Kleppe, 1999) is used for the formal specification of invariants on the model elements and attributes as well as pre-conditions and post-conditions on the functions.

The visual representation has the advantage that of showing at a glance the relevant concepts, how they are organised and how they are related to each other. This graphical representation is missing in a pure Z or Object-Z specification. UML has been chosen because it has become a standard modeling language. The semi-formal graphical representation is supplemented with semantic information formally written in OCL. The integration of formal and diagrammatic approaches is recommended, e.g. by Pastor, Insfrán, Pelechano, Romero and Merseguer (1997) that propose a mixed approach called OO-Method. Evans, France, Lano and Rumpe (1998) as well as Wirsing and Knapp (1996) stress that the formal meaning of diagrams and model elements is required for rigorous analysis. Richters and Gogolla (1999) report on considerable improvements by using OCL constraints instead of English text eliminating ambiguous interpretations of UML models.

In this work UML combined with OCL allows for an object-oriented formal specification that is equivalent to the original Z specification of the Dexter Model.

This chapter is structured as follows: The first section gives an overview of the state of the art of reference models for hypermedia applications. The second section presents the architecture of the Munich Reference Model for Adaptive Hypermedia Applications. In addition, this section outlines the changes to the Dexter model and lists differences to other reference models for adaptive hypermedia systems. The third section presents the formal specification of the model.

## **4.1 Reference Models for Hypermedia Systems**

The objectives of hypertext or hypermedia references models are to capture important abstractions found in current hypermedia applications, to describe the

basic concepts, such as the node/link structure of these systems, to provide a basis to compare the systems, and to develop a standard.

For the specification of reference models, formal, semiformal or informal techniques can be used. Formal techniques are those specification languages that are based on mathematics, logic or algebra and for which syntax, semantics and manipulation rules are explicitly defined. Semiformal techniques indicate the use of diagram techniques and tabular techniques, which present information in a structured form. Informal techniques are those that use only natural language. Wieringa, Dubois and Huyts (1997) stress the importance of integrating semiformal and formal specification techniques. The Munich Reference Model presented in this chapter seeks to achieve this aim.

In practice, the majority of specifications use a combination of two or three techniques. Several abstract models have been developed in the area of hypermedia. The existing models fall into two main categories:

- informal – semi-formal models, such as
  - the Hypertext Abstract Machine (HAM) of Campbell and Goodman (1988),
  - the Tower Model (De Bra, Houben & Kornatzky, 1992),
  - the Amsterdam Hypermedia Model (AHM) of Hardman, Bulterman and van Rossum (1994),
  - the Devise Hypermedia Model (DHM) of Grønbaek and Trigg (1996), and
  - the Adaptive Hypermedia Application Model (AHAM) of De Bra, Houben and Wu (1999), the description for which is based on tuples.
- formal models, such as
  - the Trellis Model (Furuta & Stotts, 1990), the specification for which is based on petri nets,
  - the Dexter Reference Model, the original specification for which was written in Z. Two additional specifications of the Dexter Model are the Object-Z (van Ossenbruggen & Eliëns, 1995) and the UML-OCL specification (Koch, 2000b);
  - the Dortmund Family of Formal Models of Tochtermann and Dittrich (1996), the specification for which is based on VDM (Vienna Development Model).

These models also differ in their objectives. The Dexter Model defines a common vocabulary, the hypertext abstract (HAM) is an architectural description and the Trellis model is a formal specification of hypertexts. The four most relevant reference models are outlined in the following subsections.

#### **4.1.1 The Dexter Hypertext Reference Model**

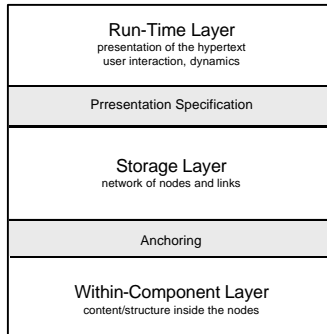
The Dexter Hypertext Reference Model was the result of the discussions of a small workshop on hypertext at the Dexter Inn, Sunapee, New Hampshire in October 1988. The purpose was to find a common language for the people involved in hypermedia development and to obtain common abstractions for the hypermedia systems existing at that time (Halasz and Schwartz, 1994). The Dexter Model has been proven to be useful and has since then been used as a basis for discussions and improvements of hypermedia systems. For example, the Devise Hypermedia Model (DHM) of Grønbaek and Trigg (1994,1996) is an extended Dexter Model that covers systems that represent and store links as objects separate from the components they connect and at the same time models systems whose links are embedded in the contents of documents. Although DHM covers similar territory to the original Dexter Model it introduces the concepts of location specification and reference specification that capture the two styles of linking in a more intuitive and integrated manner.

There is no doubt that the Dexter Reference Model – formalised by Halasz and Schwartz (1990) in the specification language Z – is one of the most important milestones in the history of hypermedia development. It uses the word “hypertext” to refer to both text-only and multimedia systems; in the same way is done in this section. The Dexter Reference Model divides a hypertext system into three layers. These are the Run-Time Layer, the Storage Layer and the Within-Component Layer connected by the interfaces Presentation Specification and Anchoring. The model focuses mainly on:

- the Storage Layer,
- the mechanisms of Anchoring (interface between the Storage Layer and the Within-Component Layer),
- the Presentation Specification (interface between the Storage Layer and the Run-Time Layer), and
- some aspects of the Run-Time Layer.

The Within-Component Layer is purposely not elaborated within this reference model. Figure 4-1 shows these layers as presented in the work of Halasz & Schwarz (1994).

The main goal of the reference model is to describe the network of nodes and links in the Storage Layer, i.e. the mechanisms by which these links and nodes are related. The nodes are treated in this layer as general data containers. The content and structure within the hypertext nodes are described in the Within-Component Layer. The Run-Time Layer contains the description of the presentation of nodes and links, user interaction and the dynamics of the application. But the Dexter Model only provides the realisation of a set of interfaces; it does not attempt to cover all the details of the user interaction with the hypertext.



*Figure 4-1: Layers of the Dexter Hypertext Reference Model*

As regards the general containers of data in the Within-Component Layer, no details are given about their content, e.g., text, graphics, animation, etc., or about the structure and the mechanism to deal with this structure.

In addition, the model includes the interfaces between the Run-Time Layer and the Storage Layer (Presentation Specification) and between the Storage Layer and the Within-Component Layer (Anchoring). The presentation interface is described in detail e.g. by the Standard Reference Model (SRM) by Bulterman, Rutledge, Hardman and van Ossenbruggen (1999). It can be observed that this separation of the contents, structure and presentation aspects of hypermedia systems is the basis of most of the hypermedia design methods.

The Dexter Model describes the Storage Layer as the structure of a hypertext system that consists of a finite set of *components*. A component is either an *atom*, a *link* or a *composite* entity. Atoms in the Dexter Model terminology are the “nodes” of the hypertext system. Links, also called link components, are entities that represent relations between other components. Composite entities are constructed out of other components. Each component includes component information and a content specification. The component information is thus a list of attributes, a presentation specification and a list of anchors.

- With *attributes* arbitrary properties can be included, for example attaching keywords to a component.
- The *list of anchors* provides a mechanism for specifying the end points of the links that relate this node to other nodes in the network.
- The *presentation specification* is used as the interface to the Run-Time Layer.
- The *content specification* is used as the interface to the Within-Component Layer.

Every component has a unique identifier (UID) associated with it. These UIDs are assumed to be unique in the whole universe of discourse.

The content of a *link component* is a list of two or more specifiers. Each specifier contains a component specification, a presentation specification, an anchor identification (id) and a direction. The field direction can be either “*from*”, “*to*”, “*bidirect*” or “*none*” with the following semantic: source of a link, destination, both, or neither source nor destination.

Anchoring is the mechanism that provides the functionality to allow for linking between nodes or documents but also for addressing (referring to) locations within the content of a component. An *anchor* is an indirect addressing entity, which has two parts: *anchor ID* and *anchor value*. The *anchor value* is an arbitrary value specifying a location, an item or a region. This anchor value is a variable and interpretable field within the content of the component. It is part of the Within-Component Layer. Otherwise, the *anchor ID* remains constant and identifies its anchor uniquely within the scope of its component or uniquely across the whole universe through a pair “UID-anchor ID”.

The UML class diagram representing the Storage Layer of the Dexter Model presented by Koch (2000b) is reproduced in Figure 4-2. It shows the model elements mentioned above and the relationships between these model elements. All the classes depicted in Figure 4-2 are part of the package “Storage Layer” with



the exception of *Content* and *Anchor Value*, which are classes of the package “Within-Component Layer”.

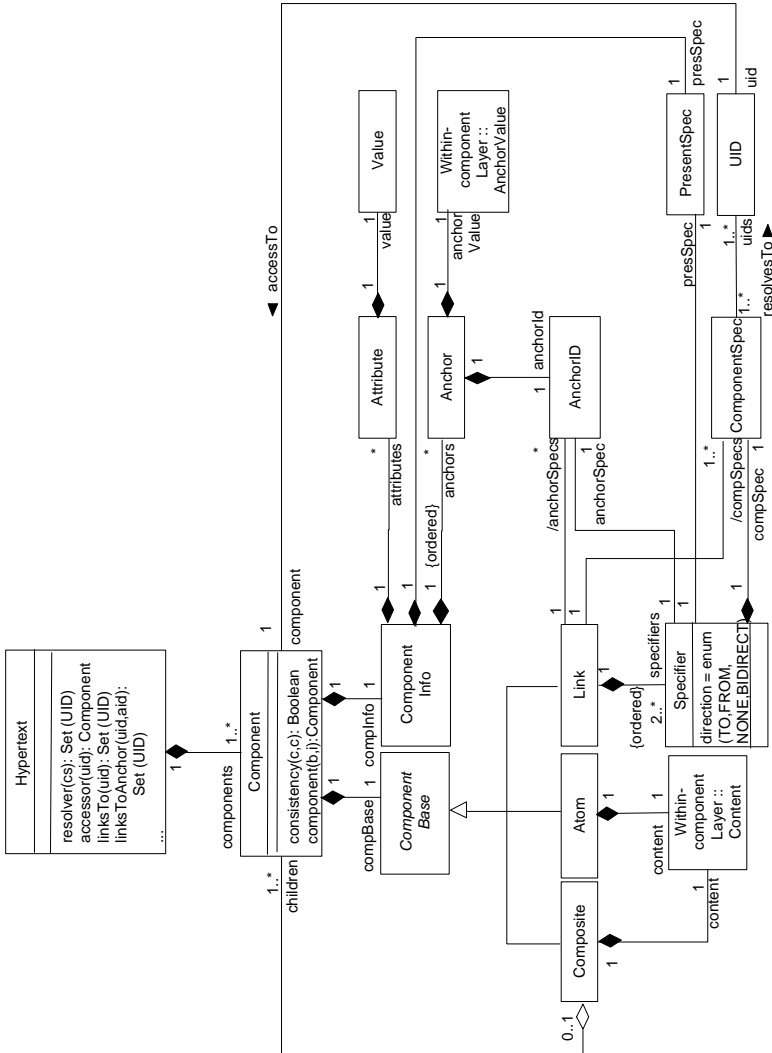


Figure 4-2: UML Class Diagram for the Storage Layer of the Dexter Hypertext Reference Model

The functionality of the Storage Layer is supported by a *resolver* function and an *accessor* function. Together they are responsible for mapping specifications of components into the components themselves, i.e. retrieving the components. The resolver function “resolves” the component specification into a component UID or set of UIDs, which is used by the accessor function to “access” the correct component(s). The accessor function may find out that no component exists for a UID. We are then in the presence of a dangling link. But, the Dexter Model requires link consistency. Therefore, when a component is deleted, the system has to guarantee that also all links resolving to that component are also deleted. This requirement has been widely criticised and is not implemented in actual hypermedia systems.

In addition to the data model the Dexter Model defines a set of operations to access or modify the hypertext structure: to create an atom, a link or a composite component, to delete or modify components, to set values of attributes and to get a component (using the accessor function) as well as an operation to get all attributes of a component. Two other operations help to determine the accessibility of the network. They are *linkToAnchor* and *linkTo* operations. In case of the former, given a component and an anchor contained in the component, the operation returns the set of links that resolve to this anchor. In the case of the latter, given a hypertext and a component UID, the operation returns all links resolving to that component.

The Run-Time Layer describes how the components are presented to the user. This presentation is based on the concept of instantiation of a component, i.e. a copy of the component is cached to the user. If the user modifies the instantiation, it is written back into the Storage Layer. The copy receives an instantiation identifier (IID). It should be noted that more than one instantiation for a component may exist simultaneously and that a user may be viewing more than one component.

Instantiation of a component also results in instantiation of its anchors. An instantiated anchor is known as a link marker. In order to follow the same structure as in the Storage Layer, the instantiation is a complex entity that consists of a base instantiation, a sequence of link markers and a function mapping link markers to the anchors they instantiate. Base instantiation is a primitive in the model and represents the presentation of a component to the user.

In order to keep track of all these instantiations the Run-Time Layer uses an entity *session*. The user will open a session by the action *present Component* of a given hypertext. She can edit the instantiation, save the modifications, create a new component or delete a component. The most common action is *follow Link*, which takes the IID of an instantiation together with the link marker contained within that instantiation and then presents to the user any component resolved according to the content of a link component specifier, i.e. components that are the end point

destination of all links. The user can also remove an instantiation and close the session.

Figure 4-3 shows the UML class diagram of the Run-time Layer and the related part of the Storage Layer (Koch, 2000b).

### 4.1.2 AHAM: Adaptive Hypermedia Application Model

The Adaptive Hypermedia Application Model (AHAM) presented by Wu, Houben and De Bra (1998) divides adaptive hypermedia systems into the same three layers as the Dexter Reference Model does for hypermedia systems. These are: the Run-

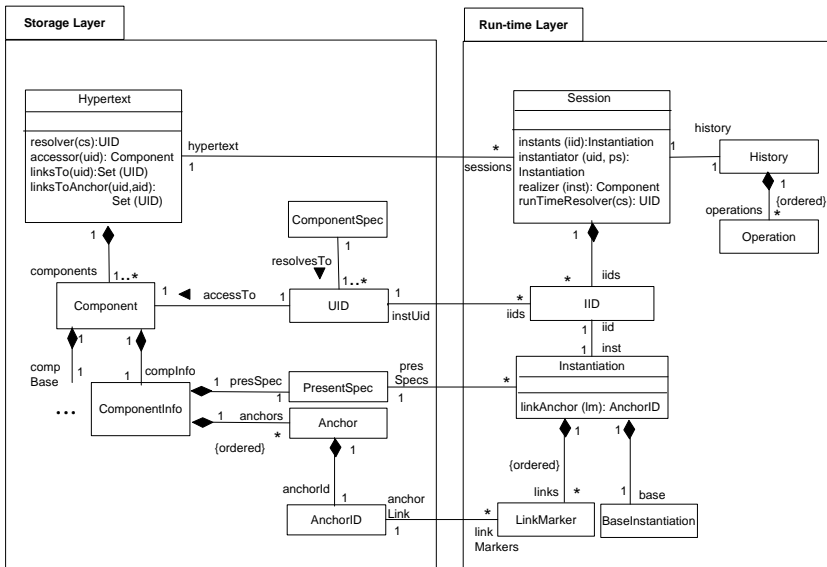


Figure 4-3: UML Class Diagram for the Run-time Layer and Part of the Storage Layer

Time Layer, the Storage Layer and the Within-Component Layer connected by the interfaces Presentation Specification and Anchoring.

The Storage Layer of AHAM is also a network of nodes and links. It consist of the same three components – *domain model*, *user (student) model* and *teaching model*

– as most intelligent tutoring systems do (Nwana, 1990). The domain model represents the author's view of the application domain. The user model contains information which the system records about the user. The teaching model consists of pedagogical rules, which define how these models, the domain model and the user model, are combined for adaptation. The Within-Component Layer specifies the content and the structure inside the nodes.

Although this reference model is not restricted to teaching applications, the terminology used is related to the instructional field. The adaptation is performed within the adaptive engine, part of the presentation specification, which is responsible for the adaptation or dynamic generation of the nodes. The anchoring is the mechanism for addressing locations or items within the content of an individual component.

The AHAM Storage Layer treats the nodes of the structure in the same way as the Dexter Model treats general containers of data. It also describes the activities of the adaptive engine. At the present time only a few details of the Run-Time Layer are given. No details are given about the content of the containers of data, such as text, graphics, animation, etc.

Based on the layers specified above, an adaptive hypermedia application is defined as a 4-tuple  $\langle DM, UM, LM, AE \rangle$  where DM is a domain model, UM is a user model, TM is a teaching model and AE is an adaptive engine.

The domain model describes the structure of an adaptive hypertext system as a finite set of concept components (in contrast to the Dexter components). Two types of concepts components are distinguished: concepts and concept relationships. Two types of concepts are distinguished: atomic and composite. And two types of composites: abstract composites and pages.

The user model based on the user's knowledge about concepts is updated only on the basis of the browsing behaviour of the user. A table representation is used as a conceptual representation of the user model. The most common attributes are:

- concept UID.
- user knowledge related to each concept, the user knowledge-value indicates how much the user knows about the concept.
- read, which indicates whether the user read some fragment, page or set of pages about the concept. This value may be Boolean or a list of access times.
- ready-to-read, which indicates if the user fulfils the knowledge prerequisites that enables her to read about this concept.

The basis for the adaptive functionality can be found in the *teaching model*, which combines information from the Run-Time-Layer about the user's behaviour, the domain model and the user model in rules that determine how information is changed in the user model and which information will be presented to the user.

The teaching model is described as a finite set of pedagogical rules. Two types of rules are defined: generic and specific rules. These rules use and change variables which denote concept UID's, attributes, anchors, parts of presentation specifications and user-model attributes for concepts and concept relationships. Adaptation may be performed based on the current state of the model or using a new state, which has been reached by applying a rule.

### 4.1.3 AHM: Amsterdam Hypermedia Model

The Amsterdam Hypermedia Model (AHM) is an informal extension of the Dexter model. It adds the notions of time, high-level presentation attributes and link context to the Dexter Model. It was developed by Hardman, Bulterman and van Rossum (1994) to support the design of hypermedia systems that use dynamic media such as audio, video and animations. These kinds of media require a model that supports the specification of temporal relationships between data items. AHM was defined by combining the Dexter Hypertext Reference Model and the CMIF Multimedia Document Model (Hardman, Bulterman & van Rossum, 1994).

The most important improvements to the Dexter Model are:

- The presentation specification of an atomic component that includes the specification of the attributes channel and duration.
- The presentation specification includes temporal layout, spatial layout and style information. The presentation specification of a composite component is extended with a list of synchronisation arcs, which include the components ID of the related components and a timing relation.
- Composite component do not include content. They only act as containers, i.e. they do not contain any data directly.
- Anchor values in composite components are replaced by a list of indirect addresses (component ID and anchor ID).
- For each child the AHM indicates a pair of component Ids and a start time.
- The composite type is specified as either parallel or choice. Parallel composite components display all their parts, whereas choice composites

display one or more of their children. The Run-Time Layer will implement the selection mechanism.

AHM supports synchronisation of multimedia components. This synchronisation consists of constraints defined between the children of a composite component – the relative starting time of each child of a composite for example. This information is given explicitly within a composite.

#### **4.1.4 DFHM: Dortmund Family of Hypermedia Models**

The Dortmund Family of Hypermedia Models supports different variations of data types. It can thus be regarded as a family of interrelated models rather than one model with fixed data type specification. The DFHM is formalised in the specification language VDM (Tochtermann, 1994).

DFHM describes the hypermedia fundamentals, such as node, component (multimedia content of node), link, anchor and hyperdocuments. These basic data types of hypermedia have been extended using hypermedia structuring concepts that organise and categorise hyperdocuments.

The structuring concepts introduced by DFHM are: link structures, views, view nodes and folders. These structuring concepts are relevant to the Munich Reference Model since they can be considered as the first approach to model customised hypermedia applications.

A *link structure* is defined in this model as a set of links that interconnects defined parts of a hyperdocument. It allows different links to be assigned in the same document. Conceptually link structures are defined as an identification, a set of links and optional attributes. DFHM suggests using the link structure to define different contexts for different users.

*Views* of hyperdocuments are defined in similarity with views over databases. They will be used to hide information that is not interesting or relevant to the user. A view allows for simplification of the user interface and for data security. A *view node* is a node containing only chosen components or nodes of underlying nodes or view nodes.

A *folder* is a container object containing nodes, links and other folders. Folders support links between folders documents and documents outside the folders.

## 4.2 The Munich Reference Model

The main objectives guiding the elaboration of this model have been the following:

- to develop a model for adaptive hypermedia applications based on the Dexter Hypertext Reference Model,
- to include the user models in the reference model,
- to model the adaptation rules and functionality,
- to produce an object-oriented approach,
- to elaborate a formal specification of this metamodel, and
- to use general terminology, i.e. independent of the application field.

The result is the Munich Reference Model for Adaptive Hypermedia Applications. It is an object-oriented specification based on UML models and the Object Constraints Language (OCL).

The UML class diagrams show a visual representation of the metamodel augmenting the intuitive comprehension. OCL (UML, 1999) supports the formal specification through invariants for model elements such as classes and attributes as well as by pre-conditions and post-conditions for the operations that describe the functionality of the metamodel<sup>5</sup>.

### 4.2.1 Architecture of Adaptive Hypermedia Systems

An adaptive hypermedia system is first of all a hypermedia system. The Dexter Reference Model is therefore used as basis for a reference model for adaptive hypermedia systems. The three-layer structure and the names of these layers are kept unchanged, even though the “Storage Layer” has more functionality than just storing information about the hypertext structure. The layers are the Run-Time Layer, the Storage Layer and the Within-Component Layer as illustrated in Figure 4-4. To support adaptation the Storage Layer consists of three models: the Domain Model, the User Model and the Adaptation Model.

---

<sup>5</sup> The specification presented in this chapter was checked with the UML-based Specification Environment (USE). USE was developed at the University of Bremen (Richters & Gogolla, 2000). <http://www.db.informatik.uni-bremen.de/projects/USE>

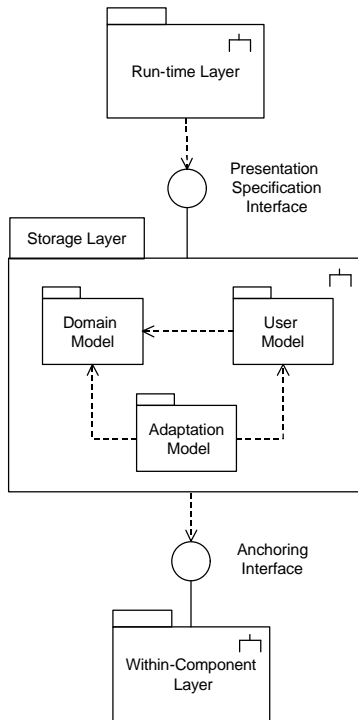


Figure 4-4: Architecture of Adaptive Hypermedia Systems

The Domain Model models the basic network structure. The User Model includes the user attributes and attribute-values that are relevant to the adaptive application. User attributes are classified as domain dependent and domain independent. A user attribute that belongs to the former group has a value for each domain component. The Adaptation Model consists of a set of rules and containers for user behaviour. Rules are triggered by the user behaviour or by other rules. Different types of user behaviour are modeled: browsing, user input and user inactivity.

The model includes two interfaces: the presentation specification and anchoring. Anchoring is the mechanism for indirect addressing that provides a fixed point of reference for use by the Storage Layer (anchor ID) combined with a variable field (anchor value) used by the Within-Component Layer. The Within-Component Layer is neither modeled in this reference model nor in the Dexter or AHAM



models. The adaptive mechanisms are defined in the adaptation model and they are responsible for the adaptive presentation, i.e. for the adaptive content, adaptive links and adaptive presentation. The presentation specification builds pages out of page fragments, taking into account the adaptive mechanisms defined in the adaptation model.

The Run-Time Layer manages different sessions for the users generating and presenting the instances of pages and storing the modifications in the Storage Layer.

In the same way as in the Dexter Model, the data model is supplemented by a set of functions. Two types of functions are distinguished:

- *Authoring functions*, needed by adaptive hypermedia applications to update components, rules and user attributes, i.e. to create an atom, a component relationship or a composite component, to create a rule, to add a user attribute to the model, to delete or modify components, rules or user attributes.
- *Retrieval functions*, are required to access the hypermedia domain structure and the user model, i.e. to get a component (using the accessor function), to get all attributes of a component, get all rules triggered by a user's behaviour or another rule, to get the value of a user attribute, etc. This functionality is determined by some of the functions defined in the next section, such as accessor, resolver, constructor, evaluator, trigger, executor, etc.

Note that the separation of the contents, structure and presentation aspects of hypermedia systems presented in the reference model is the basis of hypermedia design methods and of the modeling techniques of the UML-based Web Engineering approach presented in Chapter 6.

#### **4.2.2 Extensions to the Dexter Hypertext Reference Model**

The changes made to the Dexter Reference Model are due to the adaptive aspects of the systems that are modeled. The main extensions are:

- The hypertext in the Dexter model is the domain in the Munich approach.
- Components may be related not only by navigational relationships (links), but also by other conceptual relationships, such as part-of, prerequisite-of, inhibitor-of, variant-of and on-same-page (Boyle, 1997).

- The Domain Model models the conceptual level of the application and the hypermedia representation of the application. The latter is done by pages and links. The former using concepts, composite concepts, atoms and the concept relationships previously mentioned.
- The semantic of composite is simplified, as a composite concept can only have all children of type composite or all children of type atom.
- A User Model and an Adaptation Model are added as part of the Storage Layer.
- The User Model includes a user manager and a model for each user, consisting of user attributes and attributes values.
- Two different types of user attributes are considered: attributes that are dependent on the domain and those that are independent of the domain.
- The Adaptation Model is defined by a set of rules. These are the core of the adaptive functionality.
- Rules are classified in construction rules, acquisition rules and adaptation rules (content adapter, link adapter, and presentation adapter).
- The Adaptation Model also models user behaviour, i.e. browsing, user input and user inactivity.
- The page constructor guarantees dynamic page generation.
- Additional functions, such as “constructor”, “evaluator” and “trigger” are defined (similar to “resolver” and “accessor”).
- The model includes UIDs for all components, not only for pages.
- The Run-Time Layer only instantiates pages.

This work builds partially on the Storage Layer architecture presented by AHAM (Wu, Houben & De Bra, 1998). The approach presented in this chapter focuses on a detailed description of the User Model and Adaptation Model as well as the Run-Time Layer. It does not separate the adaptive rules from the adaptive functionality as AHAM does with the teaching model and the adaptive engine. The Munich Reference Model uses composites, which only act as containers as proposed by HAM, but do not detail the typical multimedia aspects, such as synchronisation. The aim is to build specific views for the user, as with the view nodes of DFHM, in these case by adaptation.

Another difference compared with the reference models mentioned above are the modeling techniques that are used, i.e. the object-oriented paradigma and UML. The advantage of an object-oriented approach is that it allows for integration of data and functionality. In addition, the Munich Model presents a complete formal

specification in OCL with the same scope as the Z-specification of the Dexter Reference Model (Halasz & Schwarz, 1990).

From the methodological point of view the Munich Reference Model for adaptive hypermedia systems is:

- an object-oriented approach,
- in UML standard notation,
- represented by a metamodel that shows all model elements and how they are related,
- based on general terminology applicable to every application field, and
- formally described by OCL constraints.

### **4.3 Formal Specification of the Munich Model**

The focus is on the specification of two of the three layers: the Storage Layer and the Run-Time Layer as well as parts of the anchoring and presentation specification interfaces. Each model of the Storage Layer: the Domain Model, User Model and Adaptation Model is described in detail, specifying classes, their attributes, relationships and main operations. This description consists of a graphical representation based on UML class diagrams and a complementary OCL specification. Note that the UML-OCL specification is simplified for the presentation in this work, as follows:

- Classes in UML diagrams only include – due to space problems –some attributes and operations.
- Spaces are included in invariants' names to augment readability.
- Non-side effect-free operations are included in the constraints – for abbreviation – whenever is it is possible to replace them by an expression.
- The operation “oclAsType” is sometimes omitted to increase clarity of constraints.

### 4.3.1 The Domain Model

The Domain Model describes the structure of the application. The class *Domain*<sup>6</sup> is defined as a finite set of *Components* together with the three functions, a *resolver*, an *accessor* and a *constructor*. These components model the elements of the conceptual level (concepts) and the presentation of the concepts in the hyperspace (pages). The metamodel structure has many similarities with the model of the Dexter Storage Layer.

The modeling of the components has evolved since the Dexter Model was presented. In the original Dexter Model a component consists of component information and a component base, where a component base is either a composite,

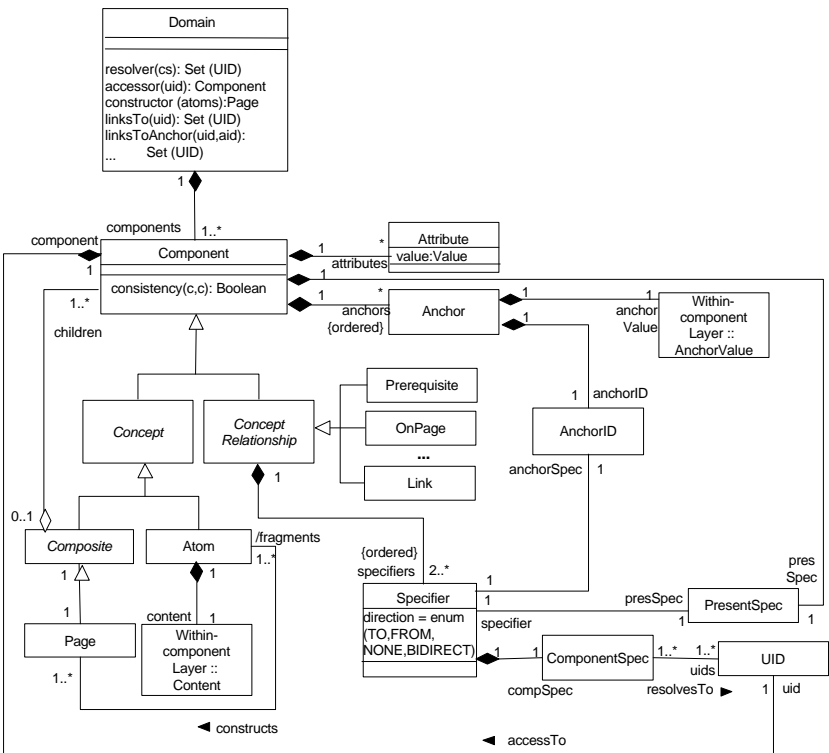


Figure 4-5: UML Class Diagram of the Domain Model

an atom or a link. This hierarchy is shown above in Figure 4-2. It allows for a composite to be composed of an atom and a link. The component base and component information are not needed in this object-oriented representation.

Grønbaek and Trigg (1994) proposed a Dexter-based metamodel where a component can be a node or a link and in both cases has some information associated to it. These changes are carried out in the UML representation of the Dexter Model (Koch, 2000b). AHAM adds the notion of domain concept to this structure, establishing that a component is either a concept or a concept relationship. And a concept can be a composite concept or an atomic concept. A composite concept whose children are all atomic is called a page concept. A fragment is an atomic content unit, it has associated an atomic concept. Every atomic concept must be a sub-component at least of one page concept.

In addition to links, other types of concept relationships are included (Boyle, 1997), such as part-of, variant-of, prerequisite-of, on-same-page, etc. If two concepts are related through a relationship of type variant-of, it means that only one of them is chosen to be presented to the user. On-same-page indicates that the related concepts have to be shown simultaneously. Prerequisite-of specifies that the source node has to be known before the target node is accessed. Another change is the use of one class to represent attributes and their values. The result is the graphical representation of the Domain Model shown in Figure 4-5.

Adaptive hypermedia applications are dynamic applications, i.e. they require the dynamic generation of pages. To perform this generation a *constructor* function is added to the model. It builds pages out of items of information, also called fragments.

As in the Dexter Model a UID is a unique identifier; it is a primitive in the model. UIDs are assumed to be unique in the entire universe of discourse. They provide a guaranteed mechanism for addressing any component in the hyperspace. The addressing process is accomplished in an indirect way based on the class called *Anchor*. An anchor has two parts: an *AnchorID* and an *AnchorValue*. Note that although the anchor value is depicted in Figure 4-5 it belongs to the Within-Component Layer. This anchor value is an arbitrary value that specifies some location within a component. The anchor ID is an identifier that uniquely identifies the anchor within the scope of the component. Together with the UIDs it makes it possible to uniquely identify the anchor within the scope of the hypermedia.

The resolver and accessor functions are, together responsible for mapping specifications of components into the components themselves, i.e. retrieving the components. The resolver function “resolves” the component specification into a component UID or set of UIDs, which is used by the accessor function to “access”

the correct component(s). Two resolver functions are included in the model; one in the Storage Layer and the other in the Run-Time Layer. The latter allows run-time aspects to be taken into account, i.e. history, for example, or adaptive aspects, such as the current values of user attributes by the resolving component specifications into UIDs. The accessor function may find out that no component exists for a UID. In such a case, it means that the domain includes a dangling link.

The functionality of the model is supported mainly by a *resolver* function, an *accessor* function and a *constructor* function. The constructor dynamically “constructs” the pages to be presented to the user on the basis of the fragments and the current state of the user model.

---

## Component

A component is an abstract representation of an information item from the application domain. It is represented with a class *Component*.

A component can either be a concept (*Concept*) or a concept relationship (*ConceptRelationship*). A concept, in turn, can either be an atom (*Atom*) or a composite (*Composite*). A concept relationship can be a link (*Link*) or a prerequisite (*Prerequisite*), or a is-part-of relation (*Is-part-Of*), etc. This inheritance hierarchy is shown in the UML class diagram (see Figure 4-5).

The model assures “type consistency” between components, i.e. two components are “type consistent” if they are both atoms or both links or both composites or both prerequisites, etc. The “type consistency” is specified by the following constraint.

```

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

This constraint has to be extended to include additional expressions if other types of concept relationships are defined, e.g. prerequisite, inhibitor or is-part-of.

A component has a component information that describes the properties of the component that are different to the content of the component. These properties are a sequence of anchors (*Anchor*), a presentation specification (*PresentSpec*) and, optionally, a set of arbitrary attribute/value pairs (*Attribute and Value*). The latter can be used to define any arbitrary property for a component and assign a value to it. The presentation specification contains information specifying how this component should be presented at run-time. It is part of the interface between the

Storage Layer and the Run-Time Layer. Anchors are part of the interface between the Storage Layer and the Within-Component Layer.

Note that a presentation specification always has some value. The component information is therefore initialised with no attributes, no anchors and a presentation specification, which is given as argument. The post-condition of the operation *init* indicates that a component instance has to fulfil these constraints.

```
context Component :: init (ps:PresentSpec)
post:   attributes → isEmpty
         and anchors → isEmpty
         and presSpec = ps
```

---

### Composite

A composite is constructed recursively out of other components. The component hierarchy is restricted to be a directed acyclic graph, i.e. a component may be a sub-component of more than one composite. No composite of the domain may directly or indirectly contain itself as a sub-component. This is an invariant that has to be fulfilled by the domain (see Sub-section Domain).

In contrast to the Dexter Model, this model imposes the restriction that all children of a composite are of type atom (this is the definition of page) or all of type composite. Composites that have composites as well as atomic children can be simulated by introducing extra intermediary composites. This restriction has already been set in the Amsterdam Hypermedia Model and in the AHAM Reference Model.

```
context Composite
inv   composite children's are all composite or all atoms:
       children → forAll ( ch : Component |
                          ch.oCllsTypeOf (Composite) )
       or children → forAll ( ch : Component |
                              ch.oCllsTypeOf (Atom) )
```

Another constraint that has to fulfil a new composite instance is the non-existence of children.

```
context Composite :: init ()
post:   children → isEmpty
```

---

**Atom**

An atom has a content, which represents the data of the component. The content of an object is a primitive of the model. It is the concern of the Within-Component Layer; therefore no details are described in the Storage Layer. The operation *init* connotes that an atom instance has no content after initialisation.

**context** Atom :: init ()  
**post:** content → isEmpty

Each atom belongs to at least to one page. The invariant that specifies this constraint is the following:

**context** Atom  
**inv** each atom belongs to at least one page:  
 Components.allInstances → exists ( c : Component |  
     c.ocIsTypeOf (Page) and  
     c.ocAsType(Composite).children → includes ( self ) )

---

**Page**

A *Page* component is a composite that has only children, which are components of type atom. The *accessor* function that translates the UIDs to components has to decide how these components are presented. For this purpose the accessor goes through the components hierarchy; whenever it reaches a page, it uses the constructor to build a page from a set of fragments, i.e. from atom components.

**context** Page  
**inv** page children's are all atoms:  
 self.children → forAll ( ch : Component |  
     ch.ocIsTypeOf (Atom) )

A derived relationship (UML aggregation), called */fragments* is included in the model just to show explicitly that a page is built as a set of atom components.

**context** Page :: /fragments(): Set (Atom)  
**post:** result = Atom.allInstances → select ( a : Atom |  
     ( self.children → includes ( a ) ) )



---

### **Concept Relationship**

The Dexter Model describes only one type of relationship between components of hypermedia systems, i.e. link components (link in this model). Instead, the components of the domain of an adaptive hypermedia application may be related by conceptual relationships, such as prerequisite, inhibitor or on-page or by a navigational relationship, i.e. by link.

The most common type of relationship is of course the link. Other types of relationship are used for adaptation together with the user attributes of the user model. A class is therefore included in the model and classes such as *Link*, *OnPage*, *PartOf*, *VariantOf*, *Prerequisite* and *Inhibitor* inherit from the class *ConceptRelationship*. The model can be extended by the definition of new types of relationships.

---

### **Anchor**

Anchoring provides a mechanism that allows for linking between nodes or documents and also for addressing (referring) to locations within the content of a component.

An anchor is defined as a pair consisting of an anchor ID (*AnchorID*) and an anchor value (*AnchorValue*). The anchor ID is an identifier, which uniquely identifies its anchor within the scope of the component of which it is part. Through the pair component UID – anchor ID, an anchor can therefore be uniquely identified across the whole universe. The anchor value is an arbitrary value that indicates some location, item or substructure within the component. The anchoring process is made possible by the decomposition of the anchor into two parts: the anchor ID is used by the Storage Layer, while the anchor value is a variable field for use by the Within-Component Layer.

Thus, to ensure that the anchor identifiers are unique within a component, the following invariant constraint must be fulfilled: The number of anchors must be equal to the number of different anchor identifiers.

**context** Component

**inv** number of anchors:

anchors → size = anchors.anchorID → asSet → size

---

### **Specifier**

Another type of component is a concept relationship that specifies which concepts are related. This consists of a sequence of at least two specifiers. A specifier

defines one single end point of a link. A specifier consists of a component specification (*ComponentSpec*) and an anchor identification (*AnchorID*), as well as two additional fields: a presentation specification and a direction.

The component specification together with the anchor identification specifies a component and an anchor within the component. The use of the component specification instead of the UID has the advantage of allowing indirect addressing, i.e. the UID of destination is resolved at run-time.

The direction encodes whether the end point is the source of the link (FROM), the destination (TO), both a source and a destination (BIDIRECT), or neither a source nor a destination (NONE). The direction of a specifier instance is initialised with NONE.

```
context Specifier :: init ()
post:    direction = #NONE
```

The presentation specification (*PresentSpec*) is a primitive value that is part of the interface between the Storage Layer and the Run-Time Layer.

---

### **Prerequisite**

A component relationship of type *Prerequisite* means that the specifiers with direction ‘FROM’ are a prerequisite, i.e. have to be visited or “known”, before a specifier with direction ‘TO’ is accessed.

The following constraint specifies that a component cannot be a direct prerequisite of itself.

```
context Component
inv a component can not be a direct prerequisite of itself:
    self.oclIsTypeOf(Prerequisite) and
    not self.oclAsType(ConceptRelationship).specifiers
        → exists (s1, s2 : Specifier |
            s1.direction = #FROM
            and s2.direction = #TO
            and domain.components → exists (c:
                Component |
                s1.compSpec.uids.components → includes (c)
                and s2.compSpec.uids.components → includes (c) ) )
```

Analogously invariants can be defined for the other concept relationships, such as *Part-of*, *Variant-of*, *Inhibitor-of* and *On-Same-Page*.

---

## Link

As already defined, a link consists of a sequence of at least two specifiers. Thus, the Dexter Model excludes dangling links. This was widely criticised; Trigg and Grønbaek (1994) argued that it makes sense to have links without specifiers at all. The Dexter Hypertext Reference Model allows for links with multiplicity greater than two. Binary links are the standard in hypertext systems.

If a specific application requires that all links should have at least one destination, this can be ensured by the following:

**context** Link

**inv** at least one specifier with direction TO:

specifiers → exists ( s: Specifier | s.direction = #TO)

Links are “first-class citizen” as they inherit from the component, which implies that links to a link component may be defined in the same way as to an atom or composite component.

Link includes two derived associations (*compSpecs* and *anchorSpecs*), establishing a direct association to *ComponentSpec* and to *AnchorID*. These associations are derived and thus annotated with a “/”. The association */compSpec* results in the set of component specifications for a link and */anchorSpec* in the set of anchor IDs for the link.

**context** Link

**inv** derived association */compSpecs*:

*/compSpecs* = specifiers.compSpec → asSet

**context** Link

**inv** derived association */anchorSpecs*:

*/anchorSpecs* = specifiers.anchorSpec → asSet

---

## Domain

The domain of an adaptive hypermedia system is represented by the class model shown in Figure 4-5. The root of the model is the class *Domain*. It consists of four parts:

- a set of components (*Component*) that represent concepts (*Concept*) and concept relationships (*ConceptRelationship*), i.e. “nodes” and “links”,
- a partial function called *resolver* that returns the UID for a given component specifier (more than one specifier may return the same UID)

- an *accessor* function, which, given a UID, returns a component (this function is total and invertible), and
- a *constructor* function that builds pages with atomic concepts.

---

### **The Resolver Function**

The *resolver* function is responsible for “resolving” a component specification into a UID. The UIDs are primitives in the model with attribute ID.

**context** Domain :: resolver ( cs : ComponentSpec ) : Set ( UID )  
**pre:** components → exists ( c: Component |  
           c.ocIsTypeOf ( Link )  
           and c.oclAsType( Link ). /compSpecs → includes ( c ) )  
**post:** result = UID.allInstances → select ( u: UID | cs.uids  
           → includes ( u ) )

---

### **The Accessor Function**

The *accessor* function is responsible for accessing the component corresponding to the resolved UID.

**context** Domain :: accessor ( uid : UID ) : Component  
**pre:** components → exists ( c: Component |  
           c.ocIsTypeOf ( Link )  
           and c.oclAsType( Link ). /compSpecs.uids → includes ( uid ) )  
**post:** result = uid.component

---

### **The Constructor Function**

The *constructor* function is responsible for gathering a set of fragments to build up a page.

**context** Domain :: constructor ( frag : Set ( Atom ) ) : Page  
**post:** result = components -> select ( p:Component |  
           p.ocIsTypeOf( Page ) and p.ocIsNew  
           and p.fragments -> forAll ( a:Atom | frag -> includes ( a ) )  
           and p.fragments = p.fragments@pre -> including ( a ) ) )  
           -> asSequence -> first

---

### **Finding all Links to a Component or an Anchor**

Two operations are included to access links and anchors, i.e. ensuring the navigation functionality of the hypermedia system. They are the *linkTo* and the *linkToAnchor* functions. The *linkTo* function determines the set of links that resolve to a specific component. The *linksToAnchor* obtains the set of links that resolve to a specific anchor of a component. Given a hypermedia system and the UID of a component in the system, the function *linksTo* (see definition of *Domain* above) returns the UIDs of all links resolving to that component.

To identify the set of links resolving to a component, as in the Dexter Reference Model, the function *linksTo* is introduced which, given a hypermedia system and the UID of a component in the system, returns the UIDs of all links resolving to that component. The inclusion of the operations *resolver* and *accessor* in the following constraints is possible, as their meta-attribute “isQuery” is true.

```

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

```

The function *linksToAnchor* returns the link components that are associated with a particular anchor of a component. The following is the OCL expression for *linksToAnchor*.

```

context Domain :: linksToAnchor (uid:UID, aid:AnchorID) : Set (UID)
post: result = linksTo (uid) → select (lid: UID |
      accessor (lid).ocIsTypeOf (Link)
      and accessor (lid)./anchorSpecs → includes (aid) )

```

Functions that modify nodes and links of the domain must ensure “link consistency”, i.e. all the component specifiers resolve to existing components. This is guaranteed by the following invariant:

```

context Domain
inv linkConsistency:
      Component.allInstances → forAll ( c : Component |
      c.ocIsTypeOf (Link) implies

```

Component.allInstances → exists ( comp : Component |  
 accessor(resolver(c./compSpec)) -> includes ( comp ) )

---

### **Domain Invariants**

There are five constraints that must be satisfied by every instance of the class *Domain* (invariants):

- The *accessor* function must yield a value for every component. As this function is invertible, every component must thus have a UID.
- The *resolver* function must be able to produce all possible valid UIDs, i.e. the range of the *resolver* has to be equal to the domain of the *accessor*.
- The anchor ID of a component must be the same as the anchor IDs of the component specifiers of the links resolving to the component.
- There are no cycles in the component/sub-component relationship, i.e. no component may be a sub-component (directly or transitively) of itself.
- Some concept relationships, such as *prerequisite* do not allow cycles, i.e. no component can be related by a prerequisite relationship to itself.

The first constraint is the “components accessibility” and ensures that all hypermedia components are accessible by means of the accessor function. This can be formalised as follows:

**context** Domain

**inv** components accessibility :

components → forAll ( c : Component |  
 UID.allInstances → exists (uid:UID | c = accessor (uid) ) )

The second constraint states that the set of UIDs obtained “resolving” component specifications (resolver range) is equal to the set of valid documents that can be retrieved by the accessor (accessor domain). It can be proved with the following two inclusions.

range of resolver  $\subseteq$  domain of accessor and

range of resolver  $\supseteq$  domain of accessor

The range of the resolver is included in the domain of the accessor by definition. The following OCL constraint thus proves then that the domain of the accessor is included in the range of the resolver.

**context** Domain

**inv** range of resolver  $\supseteq$  domain of accessor:

```

UID.allInstances → forAll (uid:UID |
  components.specifiers.compSpec → exists
    ( cs:ComponentSpec | resolver (cs) → includes (uid) ) )

```

The third constraint can be described in OCL using the previously defined operation *linkTo*. This constraint ensures that the set of anchor identifiers of a component should always be equal to the set of anchor identifiers of the links resolving to that document.

**context** Domain

**inv** anchors Ids of a component = anchors IDs of the links resolving to the component:

```

components → forAll ( c : Component |
  c.anchors.anchorID =
    Link.allInstances → select ( l:Link |
      UID.allInstances → exists ( uid: UID |
        l.specifiers.anchorSpecs =
          linksTo(uid).component.anchors.anchorID
        and accessor (uid) = c ) )

```

The fourth constraint guarantees that a component is not included in the transitive closure of sub-components of this component. It has to be proved that the transitive closure of the relation *children* does not contain a pair with two equal elements. To calculate the transitive closure, first the association *children* is transformed into an association class as depicted in Figure 4-6.

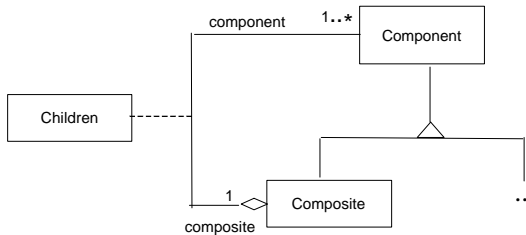


Figure 4-6: The Children Association Class

The OCL constraint that we are looking for is the following, where *transClos* is the transitive closure of the pairs of composites related by a *children* relationship:

```
not transClos → exists ( ch: Children | ch.component = ch.composite )
```

Unfortunately, OCL collections of collections are flattened, i.e. the *transClos* has to be defined as a sequence as proposed by Mandel and Cengarle (1999), of an even number of elements, where even positions belong to components and odd positions to composites. The expression written above can be replaced by:

```
not transClos → exists ( i : Integer | transClos → at (i*2-1) =
    transClos → at (i*2) )
```

The transitive closure can be calculated in two steps. First an operation called *subcomponents* is defined that builds a sequence of pairs of components (*sub\_comp*) including all components that have children of type composite.

```
context Domain :: subcomponents(): Sequence (Composite)
post: result = Children.allInstances → iterate ( pair: Children;
    sub_comp : Sequence (Composite) = Sequence{} |
    if pair.component.ocIsTypeOf (Composite)
    then sub_comp → append (pair.composite)
    → append (pair.component)
    else sub_comp
    endif )
```

In the second step an operation *transitiveClosure* is defined. It applies the Warshall's algorithm (Lang, 1988) to a given sequence of composites (pair of related composites) to calculate the transitive closure (*transClos*). The result is a sequence of all pairs of composites included in the transitive closure of the initial sequence.

```
context Domain :: transitiveClosure(initial:Sequence (Composite)):
    Sequence (Composite)
post: result = Composite.allInstances → iterate ( c3 : Composite;
    aux3 : Sequence (Composite) = initial |
    Composite.allInstances → iterate ( c2 : Composite;
    aux2 : Sequence (Composite) = aux3 |
    Composite.allInstances → iterate ( c1 : Composite;
    aux1 : Sequence (Composite) = aux2 |
    if Sequence {1..(aux1 → size) / 2} → exists ( i,j : Integer |
    aux1 → at (2*i-1) = c1 and aux1 → at (2*i) = c3
    and aux1 → at (2*j-1) = c3 and aux1 → at (2*j) = c2
```



```

then aux1 → append (c1) → append (c2) else aux1
endif ) )

```

The fourth invariant can thus be expressed using the above defined operation *subcomponents* and *transitiveClosure*. Thus, the constraint specifying that a composite may not contain itself as a sub-component can be formalised as follows:

```

context Domain
inv notItselfAsSubcomponent:
  let transClos : Sequence (Composite) =
    transitiveClosure (self.subcomponents())
  in  not transClos → exists ( i : Integer |
    transClos → at (i*2-1)
    = transClos → at (i*2) )

```

The fifth invariant specifies that concept relationships, such as *Prerequisite* do not allow cycles, i.e. no component can be related by a pre-requisite relationship to itself. First an operation *prerequisiteComp* is defined that builds a sequence of components. The principle that a pair is given by two elements of the sequence, one in the even position and one in the next odd position is used here again.

```

context Domain :: prerequisiteComp(): Sequence (Component)
post: result = ConceptRelationship.allInstances → iterate ( cr :
  cr:ConceptRelationship; prereq : Sequence (Component) =
  Sequence{} |
  if cr.oclIsTypeOf (Prerequisite)
  then
    let compPreq = components.allInstances → select (
    c:Component |
    ComponentSpec.allInstances → exists
    (cs:ComponentSpec |
    c = accessor (resolver(cs) )
    and cr.specifiers.compSpec → includes (cs)
    and cs.specifier. direction = #TO ) )
    in compPreq.allInstances → iterate (cp:compPreq;
    sub_prereq : Sequence (Component) = Sequence{} |
    sub_prereq → append (cr) → append (cp) )
    and prereq = prereq → union (sub_prereq)
  else prereq
  endif )

```

The invariant is then defined in a similar way to the previous constraint. It is required that the transitive closure be calculated for a sequence of components instead of a sequence of composites. Let *transitiveClosure2* fulfil this requirement.

**context** Domain

**inv** notItselfPrerequisite:

```

let transClos : Sequence (Component) =
  transitiveClosure2 (self.prerequisiteComp())
in not transClos → exists ( i : Integer |
  transClos → at (i*2-1)
  = transClos → at (i*2) )

```

Analogously, invariant for other concept relationships, such as *Part-of* or *Inhibitor* can be defined as well.

### 4.3.2

### The User Model

Adaptive hypermedia applications maintain a permanent User Model as part of the Storage Layer. The User Model describes the structure of the individual models of each user and how these models are administrated. User modeling comprises User Model initialisation, updating and retrieval.

The User Model package consists of a class *UserManager*, a set of users and three main functions. The functions are an *initialiser*, an *updater* and an *evaluator*. Initialisation can be performed on the basis of interviews or stereotypes. The most common updating procedure is based on the browsing behaviour of the user; it may also be performed on her answers to questions (see Chapter 3). The evaluation provides information about the current state of the user model.

A user of an adaptive hypermedia application is modeled by a user identification and a set of user attributes. The user identification identifies the user uniquely in the universe of the adaptive hypermedia application. With the attributes the adaptive hypermedia system provides a representation of the user's characteristics that are relevant for the application. We can distinguish different types of information contained in user models: user's knowledge, user's preferences, user's background experience, user's tasks, etc., summarised in two categories: dependent or independent of the domain. The values assigned to the attributes represent the beliefs the system has about the user.

Figure 4-7 shows the metamodel for the User Model package and its relationship to the classes *Domain* and *Component* of the Domain Model.

**User and the User's Manager**

The *UserManager* is responsible for the management of the set of users of the system. It consists of a set of users (class *User*) and three functions that allow for the management of the user models. These functions are an *initialiser*, an *updater* and an *evaluator*.

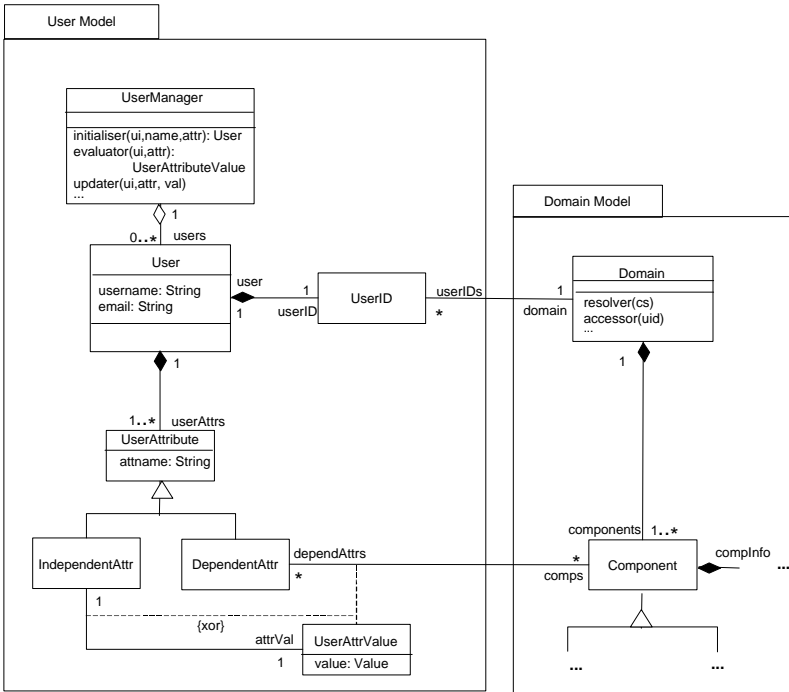


Figure 4-7: UML Class Diagram of the User Model and Associations to the Domain Model

Users (*User*) of an adaptive hypermedia system are modeled by a user identification (*UserID*) that identifies the user uniquely and a set of user attributes (*UserAttribute*). The following constraint ensures that a user is uniquely identified by her name and her email address.

**context** UserManager

**inv:** users  $\rightarrow$  forAll (u,v:User | u.username =  
v.username  
and u.email = v.email implies u = v )

---

### **Registering a New User**

When a user identifies herself to the system by her name and e-mail address, the system first checks whether the user has already been registered. If the user manager does not find the user, she is registered, i.e. a user identification is assigned, as a new user of the adaptive hypermedia application. Given a name and an e-mail address, the function *findUser* returns, if any, the user whose attributes name and e-mail are equal to the given parameters.

**context** UserManager :: findUser (n:String, e: String) : User

**pre:** users  $\rightarrow$  exists ( u:User |  
u.username = n and u.email = e )

**post:** result = users  $\rightarrow$  select ( u:User |  
u.username = n  
and u.email = e )  $\rightarrow$  asSequence  $\rightarrow$  first

The function *initialiser* creates a new instance of class *User* for each new user that registers for the adaptive hypermedia application and assigns a given set of user attributes to her.

**context** UserManager :: initialiser (userIdentification:UserID,  
n:String, e: String, defAttrs: Set(UserAttribute)) : User

**pre :** not users  $\rightarrow$  exists ( u:User |  
u.username = n and u.email = e )

**post:** let u = users  $\rightarrow$  select ( u:User |  
u.userID = userIdentification  
and u.username = n  
and u.email = e  
and u.userAttrs = defAttrs )  $\rightarrow$  asSequence  
 $\rightarrow$  first

in u.oclIsNew  
and users = users@pre  $\rightarrow$  including (u)  
and result = u

---

### **Retrieving User Model Information**

The *evaluator* is a function that, when given a user identification, a user attribute and a component, returns the value of the user attribute. The evaluator must thus

take into account whether the type of the user attribute is dependent or independent of the domain, i.e. if the value depends on a component or not.

```

context UserManager :: evaluator (userIdentification:UserID, comp:
    Component, attr: UserAttribute) : UserAttrValue
pre : users → exists ( u: User | u.userID = userIdentification
    and u.userAttrs → includes (attr) )
post : let uat = UserAttribute.allInstances → select
    (ua : UserAttribute |
        users → exists ( u: User |
            u.userID = userIdentification
            and u.userAttrs → includes (attr) )
        → asSequence → first
    in if uat.ocIsTypeOf (IndependentAttr)
        then result = uat.attrVal
        else result = UserAttrValue.allInstances → select
            ( uatVal:UserAttrValue |
                uatVal.comps = comp
                and uatVal.dependAttrs = uat )
            → asSequence → first
    endif

```

---

### Updating the User Model

The function *updater* modifies the value of a user attribute for a given user.

```

context UserManager :: updater (userIdentification:UserID, comp:
    Component, attr: UserAttribute, val:UserAttrValue)
users → exists ( u: User | u.userId = userIdentification
    and u.userAttrs → includes (attr) )
post: let uat = UserAttribute.allInstances
    → select (ua : UserAttribute |
        users → exists ( u: User |
            u.userID = userIdentification
            and u.userAttrs → includes (ua)
            and if ua.ocIsTypeOf (IndependentAttr)
                then true
                else UserAttrValue.allInstances ->
                    exists (uatVal:UserAttrValue |
                        uatVal.comps = comp and
                        uatVal.dependAttrs = ua )
            endif ) )
        → asSequence → first
    in if uat.ocIsTypeOf (IndependentAttr)
        then uat.ocAsType(IndependentAttr).attrVal = val

```

```

else uat.oclAsType(DependentAttr).attrVal = val
endif

```

---

### **Removing a User**

The function *deleteUser* allows the *UserManager* to eliminate a user identification and the user model of this user, i.e. all the user attributes and user attribute values related to her user model.

```

context UserManager :: deleteUser (userIdentification: UserID)
pre: users → exists ( u: User | u.userID = userIdentification)
post: let user = users → select ( u: User | u.userID
                                = userIdentification)
        in users = users@pre - user.

```

---

### **User Attribute**

User attributes can be classified in different ways. Some adaptive hypermedia systems distinguish three groups: attributes related to the concept knowledge, related to general knowledge or background knowledge and other attributes that describe preferences, tasks, goals, etc. It is also possible to divide the User Model into sub-models according to these criteria, such as Domain-knowledge model, User Profile and Cognitive model. Characteristics of these models have been detailed in Chapter 3.

For modeling purposes it is enough to include two groups of attributes within the User Model: “user knowledge related to the domain components” and “user general characteristics”. The first group includes domain dependent attributes while the attributes of the second group are domain independent. The second group also includes knowledge not related to the components, such as background knowledge. These classes are named *DependentAttr* and *IndependentAttr*.

The domain independent attributes can be shared with other adaptive hypermedia applications. AHAM suggests representing and implementing the second group in the same way as the first one. In the Munich Reference Model both groups are treated separately.

A dependent attribute is always related to a component. Examples of dependent attributes are:

- “knowledge”, which indicates how much the user knows about the component,
- “experience”, which can be treated similarly to knowledge,

- “confidence of belief”, which adds an estimated value of certainty to the systems belief,
- “read”, which is used to indicate if the user read something about the component,
- “ready-to-read”, which indicates whether the user is ready to read about this component,
- “time elapsed” from last reading. This is an indicator of how much the user may have forgotten,
- “solved”, which indicates if the user’s answer to a question or exercise is correct,
- “ type of errors”, etc.

Some of these attributes may be related to each other. Common attributes in educational adaptive systems are “knowledge” and “read”. All the pairs concept-UID-knowledge-value form an overlay model that represents the knowledge the system believes the user has.

Examples of domain independent attributes are listed below. These user attributes usually are adjusted by the adaptive system less often than domain dependent attributes. User behaviour, such as frequent soliciting of images or change to another language, may change the systems belief or initial settings of a language or of “no images”. Some examples of user attributes that are independent of the domain are:

- “goals”, such as searching, learning or exercising.
- “ images”, indicating with or without images,
- “examples”, with or without example material,
- “background knowledge”, such as computer experience, computer-based learning, experience, etc.
- “language”, such as English, Greek or French.

If all user attributes are independent of the domain, a domain independent User Model is obtained. This is seldom the case, but it has the advantage, that it can be used by different applications. A domain independent user model can be defined with the following constraint:

```

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

```

---

### User Attribute Values

A *UserAttributeValue* will be assigned to each attribute and each component in case of domain dependent attributes and just for each attribute by domain independent ones. Different forms of attribute values are possible, such as:

- Boolean, i.e. true and false, which means that for each component the user either knows or not knows the content of the component, has a preference or not.
- discrete, expressed by a small set of values such as “not known”, “learned”, “well learned”, “well known”, or values such as 1 for “high”, 2 for “middle” and 3 for “low” knowledge, or “s” for “exercise-solved”, “r” for “exercise-read”, “F” for “exercise-failed”, etc.
- probabilistic, given by a real number.

Domain independent attributes require only one value for each user and each attribute. User models can be implemented in different ways, such as log files, semantic nets, a table in a relational database, object-oriented classes, etc.

### 4.3.3 The Adaptation Model

The Adaptation Model consists of a set of rules and a set of functions to perform the adaptation functionality. The rules determine how pages are built and how they are presented to the user. The Munich Reference Model establishes how content-adaptation, link-adaptation and presentation-adaptation are performed and how user attribute values are changed, i.e. how the User Model is updated.

Rules are based on the information provided by the User Model and Domain Model as well as on the user interaction activities registered by the Run-Time Layer (shown in Figure 4-9).

The functions included in the Adaptation Model are an *adaptationResolver*, a *finder*, *trigger* and an *executor*. The *adaptationResolver* “resolves” a component specification into a UID, but into the UID of an adapted page of an appropriate concept. The *trigger* function implements a trigger mechanism that returns all the rules triggered by one given rule, i.e. the rules to be used at a given time. The first rule to be used is triggered by the user behaviour, such as browsing, some input or inactivity, which is provided by the Run-Time Layer. The *executor* function allows for the execution of the rules to select the appropriate concepts, obtain an adaptive content, presentation and linking as well as for the updating process of the User Model.



The user behaviour incorporated in the Adaptation Model models the different user activities, which can trigger a rule and therefore are part of the condition of a rule. The values for these user behaviours are provided by the Run-Time Layer, responsible for registering the current user activity of each specific user.

The Adaptation Model is a subsystem of the Storage Layer and has a dependency relationship with the Domain Model as well as with the User Model. Figure 4-8 shows the Adaptation Model and the classes of the Domain Model and User Model to which they are related. The different types of rules are visually represented as a hierarchy of rules.

---

## Rule

A rule is modeled as a class *Rule* that consists of one condition (*Condition*), one action (*Action*) and attributes. Attributes, such as phase and propagate, are suggested by De Bra, Houben & Wu (1999). Conditions and actions are expressions containing model elements and operators. Two types of rules can be distinguished depending on whether the rule is applicable to all instances of a domain class or just to one specific instance. The former is called global or generic rule; the latter local or specific rule.

*ModelElements* are defined by two attributes: an element identifier (*elementID*) and a Boolean value (*modified*) that indicates whether the model element is being modified in the actual action. The attribute *modified* has always value false in the case of condition objects. Only certain types of model elements, i.e. User Model attribute values and presentation specifications can have a *modified* value true. Rules can be specified using different languages, e.g. Prolog, scripting languages or Java.

A rule action (*Action*) has to include at least one model element, i.e. the constraint that one or more model elements are modified by the execution of a rule must be satisfied.

### **context** Rule

**inv** at least one model element is modified:

action.elements → exists ( m : ModelElement | m.modified )

A rule may have additional attributes. For example, the attribute *phase* determines which of two execution phases are chosen for adaptation, i.e. rules can be applied before or after the User Model is updated. If the phase attribute of the rule is “pre”, adaptation is performed on the current state of the user model. A value “post” indicates that the User Model is updated before the rule is applied, for example to generate some presentation pages.

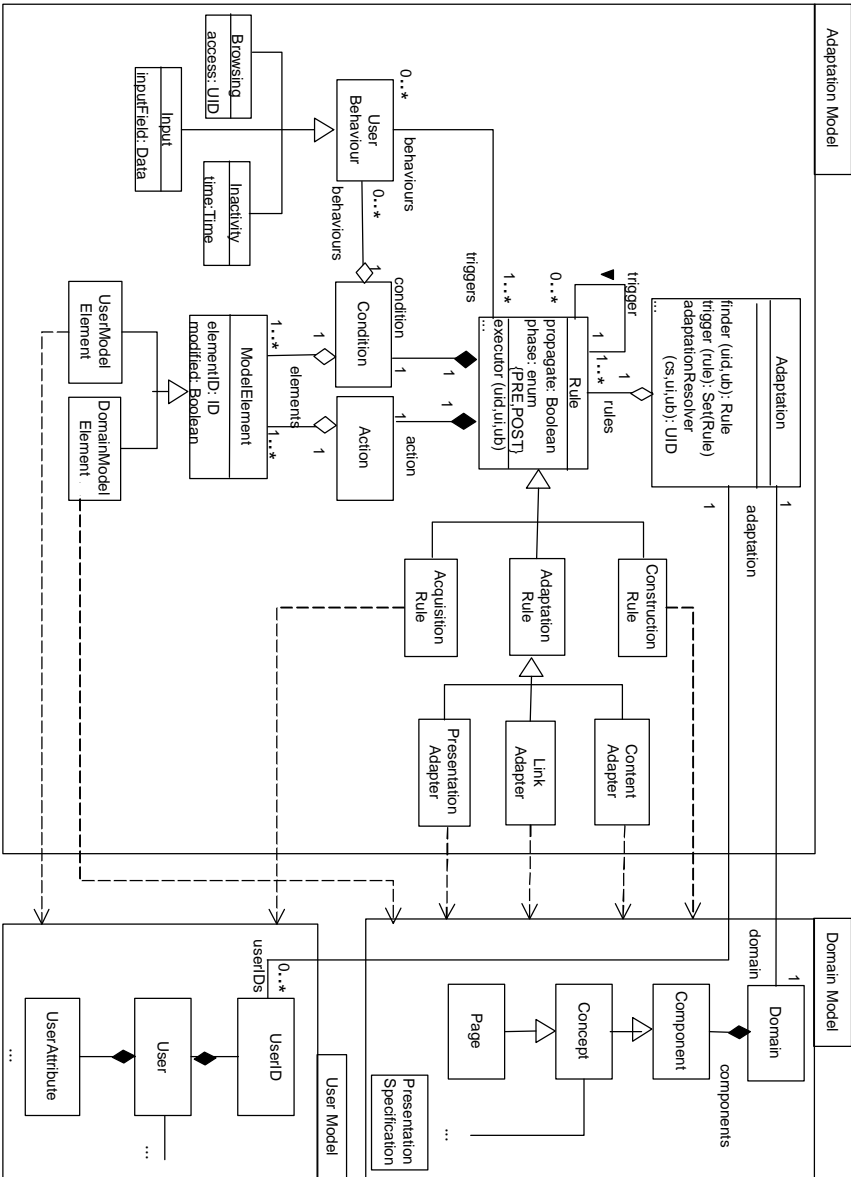


Figure 4-8: Adaptation Model

The following invariant has to be fulfilled: each rule is triggered by at least one user behaviour or by another rule. Based on the trigger two types of rules can be distinguished: *behavioural rules* and *content-based rules*.

**context** Rule

**inv** a rule is at least triggered by a user behaviour or by another rule:

```
UserBehaviour.allInstances → exists ( ub: UserBehaviour |
    self.condition.behaviours → includes (ub) )
or Rule.allInstances → exists ( r: Rule |
    r.action.elements → exists ( m: ModelElement |
        self.condition.elements → includes (m)
        and m.modified ) )
```

Rules are also classified according to their objectives into: construction rules, acquisition rules and adaptation rules. Rules that belong to the group adaptation are one of the three following types: content adapter, link adapter or presentation adapter. They differ in the executor method.

- The objective of the *ConstructionRule* is to find a concept on the basis of relationships, e.g. of type prerequisite as well as information provided by the user model. It returns the UID of the concept.
- The *AcquisitionRule*'s objective is to gather information about the user in order to build the user model. It includes a rule *executor* that returns a list of user attributes and value changes for these attributes. (The techniques for acquiring user models are explained in Chapter 3).
- The *AdaptationRule* is defined in order to adapt the pages based on the user model state. According to the three types of adaptation, three types of adaptation rules are defined:
  - *ContentAdapter* for the selection of different fragments for the page construction.
  - *LinkAdapter* for the application of different techniques of adaptive navigation, such as link annotation, link removing, link sorting, direct guidance, etc.
  - *PresentationAdapter* for the adjustment of the page presentation changing styles, fonts and sizes, for example.

Adaptation techniques are presented in Chapter 2. These different types of rules are represented using an inheritance hierarchy in the UML class diagram of the Adaptation Model as it is shown in Figure 4-8. They differ in the executor method.

---

### Executing Rules

The *executor* function of the class *Rule* is redefined for each type of rule. The user, her last interaction as well as the current concept are known parameters for the function, i.e. it is always based on a user identification, the user behaviour and the UID of a component. The executors of the different types of rules differ in the results they provide. The *ConstructionRule* returns the UID of a concept, the *AcquisitionRule* returns a set of user attributes and new values (may be relative to the existing ones) and for the three types of *AdaptationRule* the results are the fragments of a page, the adapted links and the modified presentation specification.

---

### The Core of the Adaptation

The class *Adaptation* models a finite set of rules and three main functions: an *adaptationResolver*, a *finder* and a *trigger*. The rules are used to update the User Model, find appropriate concepts for the user or change the presentation specification of model elements.

The following invariant assures dynamic update of the user model. For at least one user attribute a rule exists that modifies an attribute value of the user model.

```

context Adaptation
inv User Model is changed:
    Rules.allInstances → exists ( r:Rule |
        r.ocIsTypeOf (UMUpdater)
        and r.action.elements → exists ( m: ModelElement |
            m.ocIsTypeOf (UserModelElement)
            and m.modified ) )

```

---

### Finding Rules triggered by User Behaviour

The function *finder* identifies the rule that is triggered by a given user behaviour. It is the rule defined for a component, i.e. providing its component specification. Examples are a browsing activity accessing a component, an input activity for a component or a timeout while the user is looking at a component. Pre-condition of the function establishes that there is exactly one rule that satisfies the post-condition. This assumption can be changed allowing for a set of rules to satisfy the post-condition, but it requires the trigger function to be rewritten for a set of initial rules.

```

context Adaptation :: finder ( uid: UID, ub: UserBehaviour ) : Rule
pre : rules → select ( r:Rule |
    r.condition.behaviours → includes (ub)

```

```

    and r.condition.elements.elementsID
        -> asSequence -> first
        = uid ) → size <= 1
post : result = rules → select ( r:Rule |
    r.condition.behaviours → includes (ub)
    and r.condition.elements.elementsID -> asSequence -> first
    = uid ) -> asSequence -> first

```

---

### Triggering other Rules

The Boolean attribute *propagate* is used to allow the rule to trigger other rules. Triggered rules are all rules which condition includes model elements that are modified in the action of the given rule.

Thus, the trigger function identifies all rules that are triggered by one rule in the ‘PRE’ phase or in the ‘POST’ phase (*PhaseType*).

```

context Adaptation :: trigger ( rule : Rule, ub: UserBehaviour,
    ph:phaseType ) : Set (Rule)
pre : rules → includes (rule) and rule.propagate
post : result = rules → select ( r: Rule | r.phase = ph
    and r.condition.elements → exists ( m: ModelElement |
    rule.action.elements → includes (m)
    and m.modified ) )

```

---

### Resolving with Adaptation

The *adaptationResolver* resolves a component specification to a UID based on the component specification, the user identification and the user behaviour. The domain *resolver*, instead is used to obtain the component corresponding to the given component specification. The initial rule is triggered by the user behaviour, such a browsing activity, some input or the inactivity of the user. A finder function is defined for that purpose. Starting with this rule, the other rules to be applied are calculated by the trigger function.

```

context Adaptation :: adaptationResolver ( cs:
    ComponentSpec, userIdentification: UserID, ub:
    UserBehaviour, ph: phaseType) : UID
pre : rules.behaviours → includes (ub)
post : let uid = UID.allInstances → select ( ui:UID |
    domain.components → exists ( c: component |
    c = accessor (ui) and ui = resolver (cs) ) )
    in let i-rule = finder (uid, ub)
    in if not i-rule → isEmpty

```

```

then let r = trigger (i-rule → asSequence
                    → first, ub, ph)
in r.allInstances → forAll ( r:Rule |
                          executor (r, uid, userIdentification,ub) )
and result = uid
else result = uid
endif

```

---

## **User Behaviour**

Browsing is the most common user activity used to trigger adaptation rules. The browsing process is started with a user mouseclick that activates the *followLink* function of the Run-Time Layer. The *resolver* and *accessor* functions are responsible for identifying the page to be accessed.

In many adaptive hypermedia applications browsing is the only behaviour that is recorded. Additional information about the user can be obtained from her answers to questions, selection of options and other user input. User inactivity is under certain circumstances another helpful piece of information that can be used for adaptation, i.e. to trigger adaptation rules.

*UserBehaviour* is an abstract class of the user model. The classes *Browsing*, *Input* and *Inactivity* inherit from the class *UserBehaviour*. The following attributes are defined for these classes:

- an attribute *access* for the class *Browsing*, that indicates the component to be accessed,
- an attribute *inputField* for the class *Input*, that specifies the data entered or selected,
- an attribute *time* for the class *Inactivity*, that indicates time elapsed since last action. It is used for the time-out.

The user interaction activity is captured by the Run-Time Layer, which directs to the Adaptation Model. Typical events generated by user interaction are access to a page, timeout, mouseclick, keystroke, etc.

### **4.3.4 The Run-Time Layer**

The Run-Time Layer describes the mechanisms supporting the user's interaction with the adaptive hypermedia system. The fundamental concepts of this layer are the session and the instantiation. A session is established for each user and for each

new connection to the domain. It contains a history of all activities performed by the user. An instantiation is a presentation of the component to the user. It can be considered as a kind of run-time cache of the component as the user sees and edits a copy of the component. Thus, more than one instantiation for any given component can coexist.

A set of functions are included in the Run-Time-Layer to fulfil the presentation of the pages built according to a set of rules of the Adaptation Model with the concept pages contained in the Domain Model and adapted to the individual User Model. These functions perform the opening of a session, the opening of instantiations, modification and removal of instantiations, following a link, modification or creation of a component, closing a session, etc. Figure 4-9 shows the classes of the Run-Time Layer and the partial visualisation of the Storage Layer, mainly

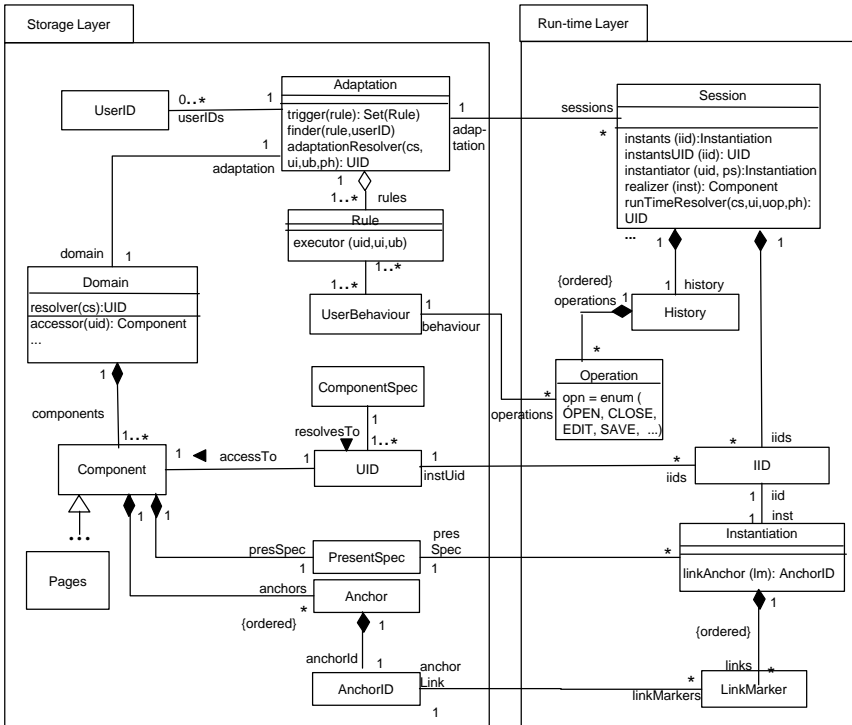


Figure 4-9: UML Class Diagram for the Run-Time Layer and Part of the Storage Layer

including classes that are related to classes of the Run-Time Layer.

Instantiation of a component also results in instantiation of its anchors. An instantiated anchor is known as a link marker (*LinkMarker*). The instantiation is an entity that consists of a sequence of link markers and a function mapping link markers to the anchors they instantiate.

---

### **Instantiation**

Each instantiation has a unique instantiation identifier from a given set of instantiations ID (*IID*). In addition, according to Halasz and Schwarz (1994), an instantiation consists of a base instantiation which “represents” a component, a sequence of link markers which “represents” the anchors of the component, and, as a minimum a function mapping link markers to anchor IDs called “link anchor” (operation *linkAnchor*).

**context** Instantiation :: linkAnchor (lm : LinkMarker) : AnchorID  
**pre:** links → includes (lm)  
**post:** result = lm.anchorLink

The invariant “dom linkAnchor = ran links” for the operation link anchor demands that for every link marker the function link anchor maps the link marker to an anchor ID.

**context** Instantiation  
**inv** dom linkAnchor = ran links:  
 links → forAll ( lm: LinkMarker |  
     AnchorID.allInstances → exists ( aid : AnchorID  
     | linkAnchor (lm) = aid  
 and LinkMarker.allInstances → exists ( lm :  
     LinkMarker | linkAnchor (lm) = aid  
 implies links → includes (lm) ) ) )

---

### **Session**

The session contains the hypertext being accessed, a history, a mapping from IIDs of the session’s current instantiations to the corresponding components of the Storage Layer, an *instantiator* function, a *realizer* function and a *run-timeResolver* function. This is represented by a class *Session* with an association to the class *Domain* and to a class *History* and a set of functions.

The history records all the operations (interactions) a user can perform during a session. The Dexter Hypertext Reference Model includes seven different types of



operations that a user can perform during a session. These operations are: open and close a session, present and unpresent an instantiation of a component, create a new instantiation during a session, as well as edit, save or delete an instantiation. The Munich Reference Model extends this list to include an additional operation, which is not a really operation as the user is inactive, i.e. she is just waiting. The aim is not to model every inactive period of time, but long periods of time, i.e. periods of inactivity that exceed a timeout limit. If the timeout is set, then the system reacts, for example redirecting to another node. The operation is called timeout.

The list of nine operations can be grouped into browsing, input and inactivity operations as shown below. These are the different user behaviours modeled in the Adaptation Model and used in the rule conditions. The user behaviour is an initial trigger in the execution of adaptive rules.

- *browsing*: open, close, present, unpresent
- *input*: create, edit, save, delete
- *inactivity*: timeout

The following invariant has to be fulfilled for every Session.

**context** Session

**inv** first operation in a session is OPEN:  
 $\text{history.operations.opn} \rightarrow \text{asSequence} \rightarrow \text{first} = \#OPEN$

A read-only session can be modeled as follows:

**context** Session

**inv** read only session:  
 $\text{not } \text{history.operations} \rightarrow \text{forAll (op: Operation |}$   
 $\text{op.opn} = \#CREATE \text{ or op.opn} = \#EDIT$   
 $\text{or op.opn} = \#SAVE \text{ or op.opn} = \#DELETE)$

For the manipulation of instantiations a mapping function is defined from instantiations to components. Instantiations are generated for a session. Given an instantiation identification, the function *instants* returns the instantiation of the component and the function *instantsUID* the UID of the corresponding component.

**context** Session :: instants (iid: IID) : Instantiation

**pre:** iids  $\rightarrow$  includes (iid)

**post:** result = iid.inst

**context** Session :: instantsUID (iid: IID) : UID

**pre:** iids  $\rightarrow$  includes (iid)  
**post:** result = iid.instUID

---

### Generating Instantiations

The *instantiator* is the core of the Run-Time model. Given a UID of a component and a presentation specification, the function returns, an instantiation of the component that is part of the session. The presentation specification is a primitive in the model, which contains information about how the component is to be presented by the system during instantiation.

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

The inverse function to the instantiator is the *realizer*. This takes an instantiation and returns a “new” component reflecting the recent changes due to editing the instantiation. This returned component is the input for the modifyComponent operation of the domain of the storage layer.

**context** Session :: realizer (ins: Instantiation) : Component  
**pre:** Instantiation.allInstances  $\rightarrow$  includes (ins)  
**post:** let new = adaptation.domain.components  $\rightarrow$   
 select (c:Component |  
 c.specifiers.presSpec.insts = ins  
 and ins.links  $\rightarrow$  forAll ( lm:LinkMarker |  
 ins.links  $\rightarrow$  includes (lm)  
 implies c.anchors.anchorID.linkMarkers  
 $\rightarrow$  asSequence  $\rightarrow$  first = lm ) )  
 $\rightarrow$  asSequence  $\rightarrow$  first  
 in new.ocllsNew  
 and adaptation.domain.components  
 = adaptation.domain.components@pre  $\rightarrow$  including (new)  
 and result = new

The following invariant assures that the set of components accessible by the accessor function is equal to the set of components realised from instantiations.

**context** Session

**inv** range of accessor = range of realizer:

```

UID.allInstances → forAll ( uid : UID |
  PresentSpec.allInstances → exists ( ps : PresentSpec |
    accessor (uid) = realizer (instantiator(uid,ps) ) ) )

```

---

### **The Run-Time Resolver**

The session's *Run-TimeResolver* is the run-time version of the storage's layer *resolver* operation. It maps component specifiers into component UIDs. The *Run-TimeResolver* is needed when run-time information is used for the resolution process, i.e. when history or time aspects are taken into account in the process. The Storage Layer *resolver* would not be able to handle this specification. The *runTimeResolver* is a superset of the Storage Layer resolver.

**context** Session :: runTimeResolver (cs: ComponentSpec, ui: UserID, uop: Operation, ph: PhaseType) : UID

**post:** result =

```

if uop.opn = #CREATE
  or uop.opn = #EDIT
  or uop.opn = #DELETE
  or uop.opn = #SAVE
then self.adaptation.adaptationResolver (cs,ui, #INPUT,ph)
else if uop.opn = #TIMEOUT
  then self.adaptation.adaptationResolver
    (cs,ui, #INACTIVITY,ph)
  else self.adaptation.adaptationResolver
    (cs,ui, #BROWSING,ph)
endif
endif

```

---

### **Opening a Session**

A Session starts with an existing domain (Storage Layer) and neither instantiations nor history. The *openSession* has to fulfil the following constraint:

**context** Session :: openSession (d:Domain)

**pre:** history.operations → isEmpty

**post:** self.ocllsNew  
 and d.sessions = d.sessions@pre → including (self)  
 and history.operations.opn → asSequence → first = #OPEN

and iids  $\rightarrow$  isEmpty

---

### Opening an Instantiation

There are several operations, which can open a new instantiation: opening components, presenting a component, following a link and creating a new component.

The first operation is called *openComponents* and it opens up a set of new instantiations based on a set of existing components. The function uses a sequence of specifiers and a sequence of present specifications as input. Two sequences are defined therefore instead of a set of pairs as in OCL all collections are flat.

```

context Session :: openComponents (specs: Sequence (Specifier),
    pspecs: Sequence (PresentSpec) ) : Set (Instantiation)
pre: specs  $\rightarrow$  size > 0 and pspecs  $\rightarrow$  size = specs  $\rightarrow$  size
post: let op = Operation.allInstances
     $\rightarrow$  select ( o:Operation | o.opn = # PRESENT )
     $\rightarrow$  asSequence  $\rightarrow$  first
in op.oclsNew
    and history.operations  $\rightarrow$  asSequence =
    (history.operations@pre  $\rightarrow$  asSequence )  $\rightarrow$  append (op)

post: let specs  $\rightarrow$  iterate ( j : Integer; newinst = Set {} |
    Instantiation.allInstances  $\rightarrow$  select (ins:Instantiation
    | ins.oclsNew
        and IID.allInstances  $\rightarrow$  exists ( iid:IID |
            instants (iid) = ins
        and ComponentSpec.allInstances  $\rightarrow$  exists ( cs:
            ComponentSpec | (specs  $\rightarrow$  at (j)).compSpec = cs
        and UID.allInstances  $\rightarrow$  exists ( uid:UID |
            runTimeResolver (cs) = uid
            and instantiator (uid, pspecs  $\rightarrow$  at (j)) = ins
            and instantsUID (iid) = uid ) ) )
in iids.inst = iids.inst@pre  $\rightarrow$  union (newinst)
    and result = newinst

```

Another way of opening a component is to follow a link from a given link marker in a given instantiation and present all the components for which the associated links have specifiers with a direction that has value “TO”. There may be more than one link involved because there may be more than one link associated with a particular anchor.

```

context Session :: followLink (instID:IID, lm:LinkMarker) :
    Set (Instantiation)

```

```

post: let specs = Specifier.allInstances → select (s:Specifier |
    s.direction = #TO
    and AnchorID.allInstances → exists (aid:AnchorID |
        aid = instants (instID).linkAnchor (lm)
        and UID.allInstances → exists (uid:UID |
            LinksToAnchor (instantsUID (instID), aid)
            → includes (uid)
            and accessor(uid).oclIsTypeOf (Link)
            and accessor(uid)./anchorSpecs
            → includes (aid)
            and accessor(uid).specifiers → includes (s) ) ) )
in specs → iterate (i:Integer ; pspecs = Sequence {} |
    pspecs = pspecs → append ((specs → at (i)).presSpec )
    and result = openComponents (specs,pspecs)

```

The *newComponent* operation models the opening of a new instantiation when a new component is created.

```

context Session :: newComponent (co:Content, sp:Set (Specifier) ,
    sc:Set(Component), ps:PresentSpec,
    presentSpec:PresentSpec): Component
post: let op = Operation.allInstances
    → select ( o:Operation | o.opn = # CREATE )
    → asSequence → first
in op.oclIsNew
    and history.operations = history.operations@pre
    → append (op)
post: let new = adaptation.domain.createNewComponent (co, sp,
    sc, ps)
in let newIID = IID.allInstances → select (iid : IID |
    UID.allInstances → exists (uid : UID |
        Instantiation.allInstances → exist ( ins :
        Instantiation | iids → excludes (iid)
            and ins = instantiator
            (uid,presentSpec)
            and accessor (uid) = new
            and instants (iid) = ins
            and instantsUID (iid) = uid
            ) ) ) → asSequence → first
in iids = iids@pre → including (newIID)
    and result = new

```

---

### **Removal of an Instantiation**

The operation *unPresent* models the removal of an instantiation.

```

context Session :: unPresent (iid:IID)
pre: iids → includes (iid)
post: let op = Operation.allInstances
        → select ( o:Operation | o.opn = # UNPRESENT )
        → asSequence → first
    in op.ocllsNew
        and history.operations → asSequence =
        (history.operations@pre → asSequence ) → append (op)
post: iids = iids@pre → excluding (iid)

```

---

### Modifying an Instantiation and/or a Component

An edit operation is used to modify instantiations (*editInstantiation*). The editing of an instantiation has no effect on the component. An explicit operation to save the changes resulting from an edit is required. This operation is the *realizeEdits*.

```

context Session :: editInstantiation (ins:Instantiation, iid:IID)
pre: iids → includes (iid)
post: let op = Operation.allInstances
        → select ( o:Operation | o.opn = # EDIT )
        → asSequence → first
    in op.ocllsNew and
        history.operations → asSequence = (history.operations
        @pre → asSequence ) → append (op)
post: let old-ins = instants(iid)
        in iids.inst = iids.inst@pre → excluding (old-ins)
        → including (ins)

```

```

context Session :: realizeEdits (iid:IID)
pre: iids → includes (iid)
post: let op = Operation.allInstances
        → select ( o:Operation | o.opn = # SAVE )
        → asSequence → first
    in op.ocllsNew
        and history.operations → asSequence = (history.operations
        @pre → asSequence ) → append (op)
post: let c = adaptation.domain.components
        → select ( comp: Component |
        Instantiation.allInstances → exists ( ins:
        Instantiation |
        UID.allInstances → exists ( uid: UID |
        instants (iid) = ins
        and instantsUID (iid) = uid
        and realizer (ins) = comp ) ) )

```

```

→ asSequence → first
in adaptation.domain.components =
adaptation.domain.components@pre → including (c)

```

---

### Deleting a Component and its Instantiations

To delete a component this component has to be instantiated in a Session. Any other instantiations of the same component have also to be deleted.

```

context Session :: deleteComponent (iid:IID)
pre: iids → includes (iid)
post: let op = Operation.allInstances
        → select ( o:Operation | o.opn = # DELETE )
        → asSequence → first
in op.ocllsNew
and history.operations → asSequence = (history.operations
@pre → asSequence ) → append (op)
post: let uc = UID.allInstances → select (uid : UID |
        uid = instantsUID (iid) )
        → asSequence → first
in let iinst = Instantiations.allInstances → select ( i:IID |
        iids (i) = uc )
in iids = iids@pre - iinst
and adaptation.domain.components =
adaptation.domain.components@pre → excluding (uc)

```

---

### Closing a Session

A session ends when it is closed out, i.e. the last operation registered in the history has value CLOSE. All instantiations of components are deleted. Changes to instantiations that have not been explicitly be saved will be lost.

```

context Session :: closeSession()
pre: history → size > 1
post: let op = Operation.allInstances
        → select ( o:Operation | o.opn = # CLOSE )
        → asSequence → first
in op.ocllsNew
and history.operations → asSequence = (history.operations
@pre → asSequence ) → append (op)
post: iids → isEmpty

```

### 4.3.5 Authoring Functions

Authoring functions are the functions needed in adaptive hypermedia systems to create or modify adaptive applications. They are mainly required to update the models, i.e. to create an atom, to create a component relationship, to create a composite component, to create a rule, to add a user attribute to the user model, to delete or modify components, rules or user attributes.

---

#### Creating a New Component

The function *createNewComponent* is the function invoked by the Run-Time Layer to incorporate a new component to the domain. It calls one of the following operations: *createAtomicComponent*, *createRelationshipComponent* or *createCompositeComponent*. This is a more readable specification than writing all conditions in one post-condition constraint. The disadvantage of the modularity is that the result is not OCL conform as the constraints does not satisfy the isQuery-is-true requirement.

```

context Domain :: createNewComponent (co: Content, sp: Set
  (Specifier), sc: Set (Component), ps:PresentSpec) :
  Component
post: result = if sc → notEmpty
  then createCompositeComponent (sc, ps)
  else if sp → notEmpty
  then createRelationshipComponent (sp, ps)
  else createAtomicComponent (co, ps)
  endif
  endif

```

*createAtomicComponent* takes a content and a presentation specification to create a new atomic component.

```

context Domain :: createAtomicComponent (co: Content, ps:
  PresentSpec) : Component
post: let c = Component.allInstances
  → select ( comp: Component |
  comp.oclsNew
  and comp.content = co
  and comp.presSpec = ps ) → asSequence → first
  in components = components@pre → including (c)
  and result = c

```



*createRelationshipComponent* takes a set of specifiers and a presentation specification to create a new relationship component. Link consistency has to be proven.

```
context Domain :: createRelationshipComponent (sp:Set(Specifier),
ps:PresentSpec): Component
post: let c = Component.allInstances
      → select ( comp: Component |
                comp.ocIsNew
                and comp.specifiers = sp
                and comp.presSpec = ps ) → asSequence → first
      in components = components@pre → including (c)
      and result = c
```

*createCompositeComponent* takes a collection of components and a presentation specification to create a new composite component. It must be ensured that any sub-component of the new composite are already in the domain.

```
context Domain :: createCompositeComponent
(sc: Set (Component), ps: PresentSpec) : Component
pre: s.ocIsTypeOf (Sequence)
post: let c = Component.allInstances
      → select ( comp: Component |
                comp.ocIsNew
                and sc.allInstances → forAll ( s: Component |
                components → includes (s)
                and comp.children = comp.children@pre
                → including (s) )
                and comp.presSpec = ps ) → asSequence → first
      in components = components@pre → including (c)
      and result = c
```

---

### **Removing a Component**

The function *deleteComponent* eliminates a Component from the domain ensuring that all links whose specifiers resolve to that component are removed.

```
context Domain :: deleteComponent (uid:UID)
pre: components → includes (accessor (uid))
post: let IIDs = linksTo (uid) → including (uid)
      in IIDs → forAll ( lid:UID | IIDs → includes (lid) implies
                components = components@pre →
                excluding (lid.component) )
```

---

### Modifying a Component

Components are modified by the operation *modifyComponent* that ensures that the associated information as well as the type (atom, composite or component relationship) remains unchanged and that the resulting hypermedia remains link consistent. The resolver is not modified when modifying a component as the new component overrides the old one.

```

context Domain :: modifyComponent (uid:UID, new:Component)
pre:   components → includes (accessor (uid))
post: let  old = accessor (uid)
        in  consistency (new,old)
           and components = components@pre → excluding (old)
                                           → including (new)

```

---

### Getting Specifiers

The operation *getSpecifier* takes a UID and uses the accessor function to return specifiers of a component.

```

context Domain :: getSpecifiers (uid:UID) : Set (Specifier)
pre:   components → includes (accessor (uid))
           and accessor (uid).oclIsTypeOf (ConceptRelationship)
post: result = accessor (uid).specifiers

```

---

### Getting and Modifying Attributes

The same as in the Dexter Model the following three operations are included to allow for manipulation of attributes of components. These operations are *attributeValue*, *setAttributeValue* and *allAttributes*.

The first one takes a component UID and an attribute and returns the value of the attribute, e.g a string.

```

context Domain :: attributeValue (uid:UID, a:Attribute) : String
pre:   components → includes (accessor (uid))
           and components.attributes → includes (a)
post: let atr = Attribute.allInstances
        → select (at:Attribute | at = a)
        and components
        → exists ( comp:Component |
                  comp = accessor (uid)

```

```

        and comp.attributes -> includes (at) )
    → asSequence → first
in result = atr.value

```

The second operation is *setAttributeValue*, that given a component UID, an attribute and a value, it sets the value of the attribute.

```

context Domain :: setAttributeValue (uid:UID, a:Attribute, v:Value)
pre:   components → includes (accessor (uid) )
        and components.attributes → includes (a)
post: let atr = Attributes.allInstances → select (at:Attribute | at = a
        and components
        → exists ( comp:Component |
        comp = accessor (uid)
        and comp.attributes -> includes (at) )
        → asSequence → first
in atr.value = v

```

The third one, *allAttributes* returns the set of all component attributes.

```

context Domain :: allAttributes () : Set ( Attribute )
post:  result = comp.attributes → asSet

```

---

### Creating a New Rule

Analogously to the creation of a new component and a new user, in the authoring mode it is allowed to create a new rule and to add this rule to the set of existing rules. The function *createRule* is defined with this purpose.

```

context Adaptation :: createRule (c:Condition, a:Action, pr: Boolean,
        ph:PhaseType) : Rule
pre:   -- none
post:  let newRule = Rule.allInstances → select (rule:Rule |
        rule.oclIsNew
        and rule.condition = c
        and rule.action = a
        and rule.propagate = pr
        and rule.phase = ph ) → asSequence → first
in rules = rules@pre → including (newRule)
        and result = newRule

```

---

### Removing a Rule

The function *deleteRule* is included in this model to allow for elimination of rules defined in the Adaptation Model.

**context** Adaptation :: deleteRule (r: Rule)  
**pre:** rules → includes (r)  
**post:** rules = rules@pre → excluding (r)

---

### Modifying a Rule

The function *modifyRule* is defined to modify the condition of a rule, the action of a rule or both.

**context** Adaptation :: modifyRule (r:Rule, c:Condition, a:Action, pr: Boolean, ph:PhaseType)  
**pre:** rules → includes (r)  
**post:** let newRule = Rule.allInstances → select (rule:Rule |  
     rule.ocIsNew  
     and rule.condition = c  
     and rule.action = a  
     and rule.propagate = pr  
     and rule.phase = ph ) → asSequence → first  
 in rules = rules@pre → excluding (r)  
     → including (newRule)

---

### Creating a New User Attribute

The function *createUserAttribute* is invoked to change the User Model structure creating a new user attribute, either a *DependentAttr* or an *IndependentAttr*, depending on if it is related to a component of the domain or not.

**context** UserManager :: createUserAttribute (n:String, c:Component, v:UserAttrValue): UserAttribute  
**post:** let ua = UserAttribute.allInstances  
     → select (uat:UserAttribute |  
     uat.ocIsNew  
     and uat.attrname = n  
     and if uat.ocIsTypeOf (DependentAttr)  
         then uat.attrval.comps → includes (c)  
             and uat.attrVal = v  
     else ua.attrVal = v  
     endif ) → asSequence → first  
 in users → forAll ( u: User |  
     u.userAttrs = u.userAttrs@pre → including (ua ) )  
     and result = ua

### Removing a User Attribute

In similar way to the incorporation of a user attribute to the set of attributes included in a User Model, a user attribute can be removed from this set.

**context** UserManager :: deleteUserAttribute (ua: UserAttribute)

**pre:** users  $\rightarrow$  forAll ( u: User | u.userAttrs  $\rightarrow$  includes (ua) )

**post:** users  $\rightarrow$  forAll ( u: User |  
u.userAttrs = u.userAttrs@pre  $\rightarrow$  excluding (ua) )

## **4.4 Basis for the definition of modeling techniques**

The Munich Reference Model presented in this chapter serves as a basis for the definition of the modeling techniques used in the design of adaptive hypermedia applications (Chapter 6). The domain model requires a conceptual design of the problem domain, which will evolve into a navigation model and a presentation model. The user model and the adaptation model find their pendant in the design. The user model is used to define user attributes and their relationships to the domain model. The adaptation model is used to specify the set of acquiring and adaptation rules as well as the collaborations between these rules and elements of the domain model and user model.



*“Unless the vast majority of Web sites are improved considerably,  
we will suffer a usability meltdown of the Web...”*  
Jakob Nielsen  
Communications of the ACM  
January 1999

## **5** Comparison of Hypermedia Engineering Approaches

Adaptive hypermedia systems are on the one hand hypermedia systems and on the other hand systems that have the capability to adapt themselves to the user’s knowledge, preferences or other characteristics. Therefore, the methods for hypermedia systems development are used as starting point for the methodology developed in this work for adaptive hypermedia systems (see Chapter 6 and 7).

Hypermedia engineering – better know as Web engineering – is a new and still evolving discipline. We are at the beginning of a long process of learning how to develop large hypermedia applications. Hypermedia applications for the Web or CD-ROM are mostly developed ad hoc, usually evolving from small to large applications and quickly becoming difficult to maintain. Guidelines and tools, which assist the hypermedia developer, are beginning to appear, but are far from being mature. Current practices often fail when used to develop large-scale applications for the same reasons as such a development fails in other areas of software development. These reasons are lack of planning and inappropriate techniques, processes and methodologies.

The development of hypermedia systems differs from the development process of traditional software in several ways. People with very different skills are involved in the process, such as authors, layout designers, programmers, multimedia experts and also marketing specialists of e-commerce applications. The role of the users is greater and this makes it more difficult to capture the structure of the domain. The non-linearity of the hyperdocuments as well as the possibility of connecting easily

to other hypermedia applications increases the complexity and risk of “lost in hyperspace”. In addition security is a concern of every Web application.

Web and hypermedia development has to take into account aesthetic and cognitive aspects as well, that traditional software engineering environments do not support (Nanard & Nanard, 1995). It tends to be more fine grained, the process more incremental and iterative, and the maintenance is a significant part of the life cycle of hypermedia applications, in contrast to the role played in traditional systems.

In the last few years many development methods have been defined. They have similarities and differences. The purpose of this chapter is to compare them and find out the similarities and the differences. These methods for the development of hypermedia systems propose a different number of steps and activities. Some of them focus only on the design or on visual representation; others focus on the complete development of hypermedia applications. They prescribe different techniques and/or notations to be used in the development process. Some tools have been implemented to support the development process that the methods propose.

This chapter is structured as follows. Section 1 briefly describes eleven different methodologies. Section 2 outlines the current notations used in hypermedia design. Section 3 presents a comparison of the hypermedia development methods based on their process steps, concepts, notations, techniques and tools. Section 4 compares the Unified Process with some of these hypermedia development methods. Section 5 provides some conclusions.

## **5.1 Development of Hypermedia Systems**

To analyse and compare methods for the development of hypermedia systems, we first of all need a more precise definition of the notion of systems development method (methodology). This is often very vaguely defined, not only because of the ambiguity of the concept of *method* and *methodology*<sup>7</sup>, but also because of the difficult in providing a precise definition of *system* and *system development* (Iivari & Maansaari, 1998).

Avison and Fitzgerald (1995) define *method for system development* as “a set of phases which guide the developers in their choice of techniques that might be

---

<sup>7</sup> Method and methodology are used as synonyms in this work.



appropriate at each stage of the project”. These techniques also have to help them to plan, manage, control and evaluate information systems projects. Palvia and Nosek (1993) give instead the following definition: A methodology is a “an organised and systematic approach to a systems life cycle or its parts. It will specify the individual tasks and their sequences”. Another problem is the distinction between method and technique. Palvia and Nosek also define technique as accomplishing a task in the systems life cycle. The result of applying a technique is certain outcomes (deliverables).

One scope problem is to determine which aspects have to be covered by a methodology. Rumbaugh (1995) proposes that a method should include four components:

- a set of modeling concepts to capture semantic knowledge about the problem and its solution,
- a set of views and notations for the visualisation of the underlying modeling information,
- a step-by-step iterative process for constructing models and implementations of them, and
- a collection of hints and rules of thumb for performing development.

Henderson-Sellers and Firesmith (1997), by contrast, suggests a more extensive list of aspects that have to be covered by a methodology. There are the following nine constituents:

- a full life cycle process,
- a full set of concepts and models that are internally self-consistent,
- a collection of rules and guidelines,
- a full description of deliverables,
- a workable notation,
- a set of metrics, together with advice on quality, standards and test strategies,
- guidelines for project management,
- advice for library management and reuse, and
- identification of organisational rules.

Most of the major methodologies developed for hypermedia systems only partially cover the life cycle of hypermedia systems and focus on the design of these systems – according to Rumbaugh’s definition. See comparison presented in Figure 5-3 in Section 5.3. Only HFPM (Olsina, 1998), Conallen (1999) and the engineering

approach of Lowe and Hall (1999) cover the whole development process following Henderson-Sellers proposal.

Basically, two modeling techniques, if any, are applied in hypermedia design: entity-relationship and object-oriented techniques. HDM and RMM are based on the E-R model. In contrast, EORM, OOHDM, SOHDM and WSDM adopt object-oriented approaches. Other methods go beyond the design and implementation, like HFPM, to describe the process covering the whole life cycle of hypermedia applications. WCML (Gellersen & Gaedke, 1999) and WebML (Ceri, Fraternali & Bongio, 2000) are both, approaches that focus on a markup language for the development of Web applications.

In addition to analysis, design and implementation, some methods include project management as well as feasibility studies, deployment, maintenance and/or quality control. HFPM and the OO/Pattern Approach suggest the use of pattern for the navigation and user interface design. The hypermedia-engineering model presented by Lowe and Hall (it is not named) goes further, proposing the creation of a reference model for hypermedia development processes.

A complementary approach to these methods is the Extended World Wide Web Design Technique (eW3DT) of Arno Scharl (1999) based on the W3DT (Bichler & Nusser, 1996). eW3DT is recommended to be used as a communication tool between researchers, system analysts and the management responsible for the development of a hypermedia system. It includes a graphical notation for the visualisation of the structure of deployed systems, including a combination of design, implementation and maintenance aspects that are mainly useful for re-engineering.

A brief description of the most relevant methods (mentioned above) for this work is given in the remainder of this Section.

### 5.1.1 HDM: Hypermedia Design Method

The Hypermedia Design Model (HDM) is one of the first methods developed to define the structure and interaction in hypermedia applications (Garzotto, Paolini & Schwabe, 1993). It is based on the E-R methodology, but extends the concept of entity and introduces new primitives as units (corresponding to “nodes”) and links. HDM entities have an inner structure and have a browsing semantics associated with them, i.e. a specification of how navigation may be performed and how information is visualised. An entity is a hierarchy of components and components are made up of units. Three types of links are defined: *structural*, *perspective* and *application links* (see Table 5-2). Structural links connect components; perspective

links connect units. These links can be automatically derived from the structure of the entities. Application links are defined by the author and connect components and entities of the same type or different type.

Two different groups of entities exist in HDM: the application entities described above and so-called “*outlines*”. These allow access to the application entities offering entry points to start navigation and the possibility of locating and selecting entities. These outlines or access structures are ordered trees of components.

Garzotto, Mainetti and Paollini (1995) distinguish the following dimensions in the analysis of hypermedia applications: content, structure, presentation, dynamics and interaction. The content addresses the pieces of information, while the structure is the content’s organisation. The presentation defines how the application content and functions are shown to the users. The interaction for HDM is the dynamic functionality operated on presentation elements. In other methods interaction is considered as part of the dynamics and presentation as it is a combination of both factors. The outlines defined by HDM are *index links*, *guided tours* and *collection links*. Index links connect the collection node to each member of the collection. A guided tour link connects the collection’s nodes in a linear sequence with each member connected to the next and previous one. In circular collections the last member connects to the first. Collection links are index or guided tour links that allow for traversing of the nodes of a collection. To support the presentational design HDM defines two concepts: *slot* and *frame*. A slot is an atomic piece of information. It can be simple or complex, such as a video synchronised with sound. Slots are composed of frames. A frame is a presentation unit, i.e. what is shown to the user.

This methodology distinguishes between authoring-in-the-large and authoring-in-the-small. The former identifies the entities, components and units while latter fills these units with content. HDM specifies the structure of the hyperspace (they call it hyperbase), as authoring-in-the-small is not within its scope.

### **5.1.2 RMM: Relationship Management Methodology**

Relationship Management Methodology (RMM) addresses the design and construction of hypermedia applications defining for this purpose a process of seven steps (Isakowitz, Stohr & Balasubramanian, 1995). These steps are: *entity-relationship design*, *slice design*, *navigation design*, *user interface design*, *protocol conversion design*, *run-time behaviour*, and *construction and testing*. An application design is represented with RMDM (Relationship Management Data

Model) based on the entity-relationship model and HDM. This method is at the same time a top down and a bottom up approach.

During the E-R design step entities, attributes and relationships are identified which will become nodes and links in the resulting hypermedia application. The second step, slice design, involves grouping entity attributes for presentation. *Slices* are “presentation units” which appear as pages of hypermedia applications. Separation of contents and presentational aspects is not carried out in this step. Navigational design is the step that identifies the navigation paths. RMM specifies navigation through *access primitives: link, grouping* (menus), *index, guided tour* and *indexed guided tour*. The conversion protocol design converts components of RMM into physical objects in the target hypermedia application. The step user interface design involves the design of the screen layouts of every diagram element including access primitives, links, anchors, indexes and general navigational aids. The techniques proposed for the user interface design by Balasubramanian, Bieber and Isakowitz (1996) are construction of mock-ups and prototyping.

A Relationship Management Case Tool – RMCASE – has been designed (Diaz, Isakowitz, Maiorana & Gilabert, 1995) to support the development of hypermedia WWW applications. It supports the RMM methodological stages via development of contexts, one for each stage. Design objects are shared among different contexts. The transition between contexts is possible through navigation.

### **5.1.3 EORM: Enhanced Object Relationship Methodology**

The Enhanced Object-Relationship Model (EORM) is defined as an iterative process that focuses on the enrichment of the object-oriented modeling by the representation of relations between objects (links) as objects. According to Lange (1996), this has the following advantages: relations become semantically rich as they are extensible constructs, they can participate in other relations and they can be part of reusable libraries. This method proposes the construction of a prototype of the user interface at an early stage during design.

The method is based on three *frameworks: class, composition* and *GUI*. The class framework consists of a reusable library of class definitions. To identify classes for an application EORM follows standard object-oriented techniques. EORM distinguishes two types of object-oriented relationships: *generalisation relationships* and *user-defined relationships*. Whereas the former have predefined semantics associated to them; the latter rely completely on the user specification.

The composition framework consists of a reusable library of link class definitions that enable users to reuse already developed link classes and extend them when necessary by using inheritance. The semantics of the basic link classes is the following:

- *simpleLink*: is the root link class that provides basic interlinking capabilities, including functions for creation, deletion and traversing.
- *navigationalLink*: provides traversal mechanisms for hypermedia links, including storage of creation time and history information (backtrack). It inherits from *simpleLink*.
- *nodeToNode*: is a link that inherits from *navigationalLink* supplying an object-to-object hypermedia link functionality.
- *spanToNode*: inherits from *navigationalLink*. It links the content of an object to another object.
- *structureLink*: is a child of *simpleLink* and the root of the structural links. It is inserted after creation into the structural context.
- *setLink*: is a *structureLink* that provides access to an object in an unordered collection of objects.
- *listLink*: is a *structureLink* that supplies access to an object in an ordered collection of objects.

The last step in this method is the design of the GUI application using elements of the GUI framework. It determines the windows of the domain and which presentation has to appear in each window, obtains presentations from attributes and operations of classes and determines how functionality is assigned to window menus.

The ONTOS Studio tool was developed to support the hypermodeling process with EORM. It utilises an interactive graphical user interface that generates a C++ implementation of the hypermodel. It is based on the ONTOS database. The method and the tool was not further developed since 1996.

#### **5.1.4 OOHDM: Object-Oriented Hypermedia Design**

The Object-Oriented Design Method (Rossi, 1996, and Schwabe & Rossi, 1998) comprises the following four activities:

- *conceptual modeling*,
- *navigational design*,
- *abstract interface design*, and
- *implementation*.

The OOHDM activities are performed in a mix of incremental, iterative and prototype-based development style. Object-oriented models are built in each step improving the models designed in previous iterations.

The *conceptual model* of the application is represented with a class diagram. This method sees a hypermedia application as a view over the conceptual model. Classes of these views are called navigational classes. The concept of *navigational context* is introduced to describe the navigational structure. It is a powerful concept that allows different groupings of navigational objects for the purpose of navigating them in different contexts. The access to these navigational elements is modeled with *access structures*, such as *indexes* and *guided tours*. Different types of indexes and navigational contexts are defined in the navigational design. A special notation is used for the representation of the navigational context schema. In addition, *InContext* classes are defined to enrich navigational objects allowing them to look different, present different attributes (including anchors) as well as different behaviour (methods) depending on the context within which they are navigated.

The abstract interface model is the result of the specification of the interface objects the user will perceive. OOHDM uses *Abstract Data Views* (ADVs) to model the static aspects of the user interface (Carneiro, Cowan & Lucena, 1993) while dynamic aspects of the user interface are modeled with a technique based on Statecharts (Harel, 1987).

The OOHDM-Web environment allows for a rapid prototyping of hypermedia applications designed using OOHDM. It requires the conversion of navigation constructs defined by the user, (such as navigation classes, navigational contexts, classes for presentations (*InContext* classes) and access structures) into tables as it is not an object-oriented approach. Navigation and interface constructs of OOHDM are mapped into a library of functions in the cgi-Lua script language therefore using the Lua-Database (Schwabe & Almeida Pontes, 1998). Based on the table it generates pages dynamically. The designer builds page templates with a mix of HTML tags and special commands that are interpreted by the cgi-Lua scripting environment with a set of special functions derived from OOHDM navigation primitives.

### 5.1.5 SOHDM: Scenario-based Object-oriented Hypermedia Design Methodology

Another method was recently developed by Lee, Lee and Yoo (1998) is the Scenario-based Object-oriented Hypermedia Design Methodology. It consists of six phases: *domain analysis*, *object modeling*, *view design*, *navigation design*, *implementation design* and *construction*. This methodology has similarities with RMM, OOHDM and EORM. It differs in the use of scenarios, which are described through scenario activity charts, based on events, activities and activity flows primitives. *Scenarios* are defined in the domain analysis phase and are used for the object modeling. View design consists of determining object-oriented views generated from single object classes or from associations between object classes.

The navigation design uses scenarios to determine access structure nodes. They are defined as a menu-like mechanism that enables users to access other parts of hypermedia documents. The access structures nodes (ASN), together with the object-oriented views, are called navigation units. The ASNs are similar to the access primitives of RMM: *grouping* (menu), *index* and *guided tour*. Object-oriented views are categorised into three types: *base view*, *association view* and *collaboration view*. These views are similar to contexts defined in OOHDM. A base view is generated from a single object class. An association view is extracted from an association relationship. Similarly, a collaboration view is generated from a collaboration relationship. The identification of navigation links completes the navigational design.

During the implementation design the user interface, pages and a logical database schema are modeled. This method presents a clear sequence of steps that benefits from the scenarios obtained as results of the analysis phase. It defines its own elaborated graphical notation.

### 5.1.6 WSDM: Web Site Design Method

The Web Site Design Method (WSDM) is a user-centred approach that defines the information objects based on the information requirements of the users of a Web application. WSDM proposed by De Troyer and Leune (1997) consists of three main phases: *user modeling*, *conceptual design* and *implementation design*.

In the user modeling phase the potential users/visitors of the Web site are identified and classified according to their interests and navigation preferences, for

example<sup>8</sup>. Starting point is the description of the domain, taking into account user activities. Different perspectives are defined for the user classes; these are different ways classes of users look at the same information and navigate through the information. Conceptual design consists of two steps: object modeling and navigational design. Object modeling is then done in three steps: business object modeling, user object modeling and perspective object modeling.

The navigational model consists of a number of navigation tracks, one for each perspective, expressing how users of a particular perspective can navigate through the available information. WSDM describes it in terms of components and links. It distinguishes three types of *components*: *navigation*, *information* and *external*. Each *navigation track* consists of three layers: context, navigation and information layers. The context layer is the top level of the navigation track starting with a navigation component. The information layer is the bottom level of the navigation track. The navigation layer connects the context layer and the information layer. Intermediate components and links, such as indexes and menus are created to access the information.

This kind of navigational design achieves Web applications that have a very hierarchical structure. The implementation design step creates a consistent and efficient look and feel to the conceptual model. Few recommendations are given in this step, such as the use of index pages, information divided into right-sized chunks, use of context and information cues and use of navigational cues.

## 5.1.7 MacWeb Approach

The MacWeb Hypermedia Development Environment is an engineering environment developed by Nanard and Nanard (1995). They emphasise the importance of the creative aspects in the hypermedia development process: "An important part of hypertext design concerns aesthetic and cognitive aspects that software engineering environments do not support". Therefore, the design activity is performed in a two-dimensional space of *methods steps* and *mental processes*.

The *mental process* includes the steps: *generating material*, *organising and structuring*, *reorganising and updating*, as well as *evaluation*. The *methods steps* are similar to steps in other methods described in this chapter: concepts elicitation, navigation model, abstract interface, implementation model and testing. The

---

<sup>8</sup> Note that the meaning of user modeling in WSDM is different to the definition given in Chapter 3.



designer switches from mental process to methods steps, as there are not predefined transition rules between the activities or steps. The designer's strategy may be bottom-up and/or top-down. They assert that this chaotic process must not be impaired or the author's creativity would be reduced drastically.

MacWeb's design environment proposes using object-oriented techniques, such as generalisation and instantiation as well as the well-known technique *light prototyping* used in human-computer interaction (HCI) development. Through prototyping users are able to evaluate interface design and hypermedia structure.

MacWeb's basic hypertext model relies on typed nodes connected by bi-directional typed links. MacWeb has a built-in tool that helps users construct a structure as a semantic network. It provides a few primitive built-in types, such as node, link, script node, group node. Node represents a concept. A link denotes a relationship between two types (nodes). Firing a link to a script node will trigger the execution of its content. A node of type group comprises sets of nodes organised as sub-webs.

### **5.1.8 HFPM: Hypermedia Flexible Process Modeling**

The Hypermedia Flexible Process Modeling (HFPM) presented by Olsina (1998) is an engineering-based approach that includes analysis-oriented *descriptive* and *prescriptive* process modeling strategies. It describes existing processes, thereby giving guidelines for the planning and managing of a hypermedia project.

HFPM embraces *functional*, *methodological*, *informational*, *behavioural* and *organisational* views or perspectives of the hypermedia development process.

- The *functional view* prescribes a set of phases and activities to perform tasks (e.g. to find users, classes, uses cases, etc.).
- The *methodological view* defines a set of specific process constructors to be applied to the different tasks (e.g. use-case-driven analysis, OOHDM-based conceptual and navigational design, etc). One or more methods are selected to support the tasks of a specific process and one or more tools can support a specific method.
- The *informational view* plans to produce a set of artifacts (i.e. results such as a navigational model or physical model), which are required by the tasks.
- The *behavioural view* represents the dynamic of the process model making decisions about sequencing and synchronisation of tasks,

iterations, increments, parallelisms, termination conditions, feedback loops, etc.

- The *organisational view* defines aspects such as roles, team organisation, communication mechanisms, groups dynamic, etc.

The list of tasks and suggested subtasks prescribed by HFPM for the development of hypermedia applications is listed below. It includes the following technical, managerial, cognitive and participatory tasks:

- *software requirement modeling*, such as initial survey, use-case modeling, non-functional modeling, glossary construction;
- *project planning*, i.e. analysis and specification of the project plan;
- *conceptual modeling*, that consists of analysis and specification of the problem domain;
- *navigational modeling*, that comprises analysis of intended user's tasks, identification of navigational classes, specification of navigational schema, specification of navigational transformations;
- *abstract interface modeling*, that refers to analysis of user interface models, specification of interface perceptible objects, events and transformations;
- *design patterns employment*, i.e. use of navigational, architectural and user interface patterns;
- *multimedia data capture and editing*;
- *physical modeling/integration*, that comprise component employment, rapid-functional prototyping, evolutionary prototyping, sketching or storyboarding, and component integration;
- *validation/verification*;
- *cognitive criteria employment*, such as coherence and orientation criteria employment;
- *quality assurance*, like analysis of quality and improvement strategies, specification of quality plan;
- *project co-ordination and management*, i.e. control and management of process, artifacts and resources; and
- *documentation*.

This is a wide engineering-based approach. It covers all essential phases and activities of a hypermedia project, helps to establish milestones and metrics as well as promoting communication and human understanding. Olsina (1998) presents a conceptual model for HFPM based on a set of key-concepts, such as process itself,

task, artifact, resource, agent, role, process constructor, process description, goal, resource and operation.

### 5.1.9 OO/Pattern Approach

The OO/Pattern approach to hypermedia collection design (Thomson, Greer & Cooke, 1998) is similar to HFPM as it proposes to utilise both, an OO-design and the application of patterns for the navigational and presentational design. It differs from HFPM because it does not cover the whole life cycle of a hypermedia application, i.e. project management, testing and maintenance aspects are not included. The use of patterns has well-known advantages, such as that the process is well defined, documentation can be reused and maintenance is easy.

This method prescribes the following steps: *use cases*, *conceptual design*, *collaboration design*, *class definition*, *navigational design* and *implementation*. The innovative aspects of these methods in relation to other methodologies are:

- *Use case analysis* for the different types of users.
- *Collaboration design* based on the defined use cases and the conceptual design.
- *Navigational design* based on patterns.

They describe the layered hierarchy pattern whose purpose it is to create an intuitive navigation in a naturally hierarchical structured information.

### 5.1.10 WAE – Conallen Process

Conallen (1999) proposes the use of the Rational Unified Process (Kruchten, 1998) and defines a Web Application Extension for UML (WAE). The workflows (steps) included are: project management, requirements capture, analysis, design, implementation, test and configuration management. Three Web application architectures patterns are presented in this work: Thin Web Client, Thick Web client and Web Delivery. In the first one there is little control of the client's configuration. In the second one an amount of business logic is executed on the client. The last one describes a Web client-server system supporting distributed objects.

The WAE includes stereotypes for the modeling of Web-specific architectural elements. An icon file including stereotypes can be downloaded for use in the Rational Rose tool. These stereotypes (see Table 5-1) are used to define different types of nodes (e.g. *pages*) and user interface elements (e.g. *forms and framesets*)

as well as different types of relationships between elements (e.g. *link*, *build* and *submit*).

The Conallen process focuses on architecture and implementation, including a reverse engineering step, which achieves the update of the design models according to the code. It offers little support for a systematic construction of the navigational structure of Web application and its presentational aspects. It does not treat each of the Web specific aspects — content, navigation and presentation — separately as most of the hypermedia design methods do.

### 5.1.11 Low-Hall's Engineering Approach

Low and Hall (1999) provide a framework for the development process of hypermedia applications. The framework includes *domain analysis*, *product modelling*, *process modelling*, *project modelling*, *development* and *documentation*. Product modelling consists in choosing a model for the final product. The framework supports three different product models: a programming language-based approach, a model-based approach and an information-centred approach.

- In the *programming language-based model* the information and the information structure is embedded into the programming structure.
- In the *screen-based model*, pages are manually linked together to obtain the hypermedia application.
- In the *information-centred model*, the information is stored in a database.

Process modelling consists of selecting phases, activities and artifacts for development and establishes how these are integrated into the specific development process of an application. Incremental development and prototyping are recommended for the development of hypermedia applications.

The activities that are performed during the development process are: *system analysis*, *design*, *production*, *verification* and *testing* as well as *maintenance*. A set of development techniques are presented for some activities related to the analysis and to the design of a hypermedia application. For example, it suggests RMM as methodology for the design of the structure.

## 5.2 Notations used in Hypermedia Design

Most of the works in the hypermedia modeling field concentrate their attention on defining a new notation for the visualisation of design models, such as the methodologies already outlined above: HDM, RMM, OOHDM, SOHDM and WSDM. Some of them use standard notations only for the conceptual design. These notations are object-oriented like OMT or UML, or are based on E-R. But all of them define their own notation and graphical techniques for the other steps in the design process. RMM uses E-R notation for the E-R design. EORM chooses OMT for the class structure diagram. In early papers OOHDM proposes the use of OMT notation; however, in a latter one it uses UML, but only for the conceptual model as it uses its own notation for the navigation and abstract interface model. WSDM is not restrictive in the notation selection; E-R and OMT are both suggested for the business object modeling, user object modeling and perspective object modeling steps. For the navigation model an individual notation is chosen. SOHDM is a scenario-based methodology using its own notation for the scenario activity diagrams, the class structure diagrams and object-oriented views. The class structure diagram is a graphically representation of the information contained in Class-Responsibility-Collaboration Cards, so called CRC Cards (Wilkinson, 1995).

Other works, like WAE and the Structured Hypermedia Design Technique (SHDT) focus on an implementation near notation. WAE defines therefore a UML extension. SHDT only defines a set of design primitives that are derived from the functionality of HTML. These primitives used for the static representation are: site, diagram, page, layout, form, index, menu, link and dynamic link. Site and diagram allow for grouping of diagrams and pages respectively. Dynamic page generation is represented by templates and dynamic links. The SHDT WebDesigner is a CASE tool that provides support for the design of Web applications. It generates a prototype of the planned Web system based on the SHDT design and on HTML stylesheets performed for the pages.

Another approach is the work of Baumeister, Koch and Mandel (1999) that combines ideas of OOHDM, RMM and SOHDM. It focuses on the use of UML techniques for the graphical representation of the models and presents a UML extension for hypermedia applications. The models built are the conceptual model, navigation model and presentation model. The UML extension is based on the definition of stereotypes and the use of OCL constraints.

### **5.3 Comparing Hypermedia Development Methods**

The comparison of software systems development methods is a difficult task. The focus of the methodologies may be different, some try to address many aspects in the development process, others try to detail in depth one or two of them. The

comment in this work that a specific issue is not addressed, is then not be seen as a criticism, rather as an observation that the methodology will not offer help with this issue.

In this chapter three comparative studies are presented. The first one shown in Table 5-1 extends the comparison of Lee, Lee and Yoo (1998). It compares the steps to be performed when using a method for the development of hypermedia applications, the modeling technique used as well as the graphical representation and notation chosen for the models. Another criterion used for the comparison is the CASE tool support they offer for the development.

The steps or phases of the process are numbered. These numbers are used in the fourth and fifth column to indicate what kind of graphical representation and notation is proposed for each step. It can be observed that although the number of steps, the techniques, notations and the graphical representation are different, the sequence these steps are performed in is similar. First the domain model of an application is analysed and/or designed, second the methods focus on structure and navigation and in a third step they proceed with the user interface design.

	Process	Modeling technique	Graphical representation	Notation	Tool support
HDM	1. Authoring-in-the-large 2. Authoring-in-the-small	E-R	1.- 2. E-R diagram	1. E-R	
RMM	1. E-R design 2. Slice design 3. Navigational design 4. Conversion protocol design 5. UI screen design 6. Run-time behaviour design 7. Construction and testing	E-R	1. E-R diagram 2. Slice diagram 3. RMDM diagram	1. E-R 2.-3- own	RMCASE
EORM	1. Class framework 2. Composition framework 3. GUI framework	OO	1. Class diagram 3. GUI design	1. OMT	ONTOS Studio
OOHDM	1. Conceptual design 2. Navigational design 3. Abstract UI design 4. Implementation	OO	1. Class diagram 2. Navigational Class + Context schema 3. ADV Configuration diagram + ADV charts	1. OMT/ UML 2. own 3. ADVs	OOHDM- Web
SOHDM	1. Domain analysis 2. OO Modeling 3. View design 4. Navigational design 5. Implementation design 6. Construction	Scenarios OO Views	1. Scenarios activity diagrams 2. Class structure diagram 3. OO view 4. Navigational link schema 5. Page schema	1.- 5. own	

	Process	Modeling technique	Graphical representation	Notation	Tool support
WSDM	<ol style="list-style-type: none"> <li>1. User modeling</li> <li>2. Conceptual design                             <ol style="list-style-type: none"> <li>2.1 Object modeling</li> <li>2.2 Navigational design</li> </ol> </li> <li>3. Implementation design</li> <li>4. Implementation</li> </ol>	E-R/ OO	<ol style="list-style-type: none"> <li>1. E-R or class diagram</li> <li>2. Navigation layers</li> </ol>	<ol style="list-style-type: none"> <li>1. OMT</li> <li>2. own</li> </ol>	
MacWeb Approach	<ol style="list-style-type: none"> <li>A1. Concept elicitation</li> <li>B1. Generating material</li> <li>A2. Navigational model</li> <li>B2. Organising and structuring</li> <li>A3. Abstract Interface</li> <li>A4. Implementation model</li> <li>B3. Reorganising and restructuring</li> <li>A5. Testing</li> <li>B4. Evaluation</li> </ol>	OO	A2. Class structure	2. own	MacWeb
HFPM	<ol style="list-style-type: none"> <li>1. Requirement modeling</li> <li>2. Project planning</li> <li>3. Conceptual modeling</li> <li>4. Navigational modeling</li> <li>5. Abstract interface modeling</li> <li>6. Design patterns employment</li> <li>7. Multimedia data capturing/editing</li> <li>8. Physical modeling/integration</li> <li>9. Validation/verification</li> <li>10. Cognitive criteria employment</li> <li>11. Quality assurance</li> <li>12. Project co-ordination and management</li> <li>13. Documentation</li> </ol>	OO	OOHDM	3.-5. = OOHDM	
OO/ Patterns Approach	<ol style="list-style-type: none"> <li>1. Use case analysis</li> <li>2. Conceptual design</li> <li>3. Collaboration design</li> <li>4. Class definition</li> <li>5. Pattern-based navigational design</li> <li>6. Implementation</li> </ol>	OO	<ol style="list-style-type: none"> <li>2. Class diagram</li> <li>3. Collaboration diagram</li> </ol>		
WAE-Conallen Approach	<ol style="list-style-type: none"> <li>1. Project management</li> <li>2. Requirements capture</li> <li>3. Analysis</li> <li>4. Design</li> <li>5. Implementation</li> <li>6. Test</li> <li>7. Deployment</li> <li>8. Configuration and change management</li> </ol>	OO	2.-5 UML-diagrams	UML-extension	Rational Rose
Low-Hall	<ol style="list-style-type: none"> <li>1. Domain Analysis</li> <li>2. Product modeling</li> </ol>	RMM (sugges-	RMM	RMM	

	Process	Modeling technique	Graphical representation	Notation	Tool support
Approach	3. Process modeling 4. Project planning 5. Development 5.1 Analysis 5.2 Design 5.3 Production 5.4 Verification and testing 5.5 Delivery and maintenance 6. Documentation	ted)			

Table 5-1: Methods for Hypermedia Development: Processes, Techniques, Notations and Tools.

The second comparative study is a concept-based comparison. It is similar to the table elaborated by Jacobson (1992) to compare OOSE, OOA, OOD, HOOD, and OMT. The results are shown in Table 5-2. It relates the design concepts of some of the methods outlined to the three typical levels of hypermedia design: conceptual, structural and visible level. Most of these methods, as mentioned above, clearly separate the domain problem analysis from the specification of the navigation space structure as well as from the design of the user interface. The most important concepts used by seven different design methods are listed in the table. Similar main concepts are placed on the same row.

	HDM	RMM	EORM	OOHDM	SOHDM	WSDM	WAE
Conceptual level	entity	entity	class	class	scenarios: -event activity	object	class
	collection perspective relationship	relationship	oo- relationship - generalised - user defined	perspective oo- relationship	activity flow	perspective relationship	oo- relationship



	HDM	RMM	EORM	OOHDM	SOHDM	WSDM	WAE
Structural level	link: structural application perspective  node component  outlines: collection link index link guided tour	link: uni- directional bidirectional  Slices  acc. primitives: grouping (menu) index guided tour indexed guided tour	link: simple navigational -nodeToNode -spanToNode structural: -set -list	link  nav. class  nav. context  access structures: index guided tour	nav. link  oo-view: -base -association -collaboration  ASN (acc. structure nodes): grouping index - guided tour	link  component navigation information external  nav. track	link targeted link redirect build submit  web page client page server page
Visible level	slot frame	slices		ADV (Abstract Data View)   inContext	UI component:  choice search input text button image slide bar anchor -others		frameset form target select element input element text area elem.

Table 5-2: Concepts Used in the Methods

Abbr.: nav: navigation, acc: access, oo: object.-oriented, UI: user interface

The third comparison is given in Figure 5-3. It compares the steps or phases covered by these methods. Note that comparing phases like this, hides other important aspects. The depth with which a method describes a phase and the guidelines given for that phase can vary enormously. Some method descriptions

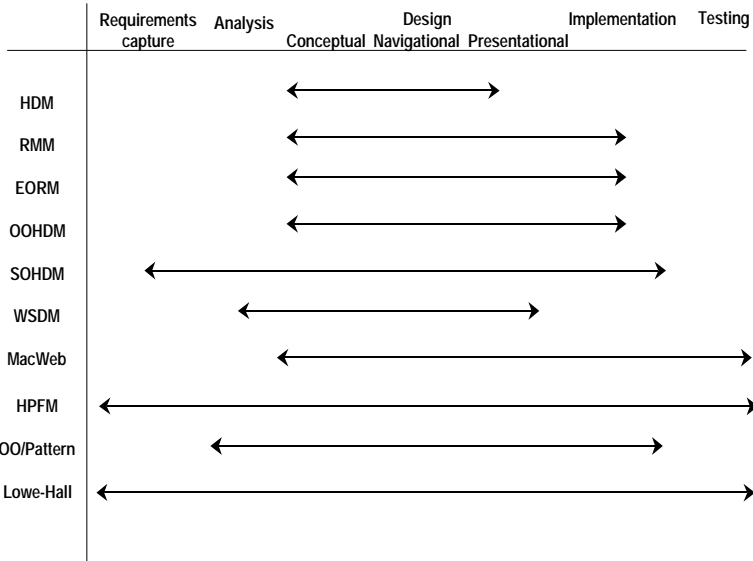


Figure 5-3: UP Workflows covered by the Hypermedia Development Methods

only propose a set of textual guidelines while others provide tools to support the phase. As a basis of comparison the core workflows proposed by the Unified Software Development Process (in short form Unified Process) are used here: requirements capture, analysis, design, implementation and testing.

## 5.4 The Unified Process and Hypermedia Development

Object-oriented development processes are becoming a de facto standard in software engineering. In particular, UML-based processes, such as the RUP (Kruchten, 1998), the Irrational Separated Process (ISP) of Hitz and Kappel (1999), the object-oriented approach of Oestereich (1999) and the Unified Process of Jacobson, Booch and Rumbaugh (1999).

The Unified Process proposes a development process for general software systems. The Unified Process is the result of the development of, and practical experience with, other methodologies, such as the Objectory Process (Jacobson, 1992) and the Rational Unified Process. The latter extends the Unified Process with the goal to cover the whole software life cycle providing workflows for project management, configuration and change management.

The methods described and compared in the previous sections are not based on the Unified Process with the exception of the method of Conallen. UML notation is used in only a few cases. A list of the most relevant characteristics of the Unified Process is detailed below. For each of these characteristics a brief analysis is done to determine which aspects are covered by methodologies for hypermedia systems outlined in the previous sections.

- **Grouping of the iterations in inception, elaboration, construction and transition phases.**

The life cycle of the development process of a software product is divided in the Unified Process in inception, elaboration, construction and transition phases. Note that maintenance is not a separate phase. It can be considered as another development project with its own inception, elaboration, construction and transition phases. The Unified Process suggests one or more iterations for each of the four phases.

OOHDM, RMM, the WAE-Conallen approach and the engineering approach of Lowe and Hall describe an iterative process but they do neither make a distinction between these iterations nor do they specify a number of iterations to be performed.

- **Covering requirements capture, analysis, design, implementation and testing**

The Unified Process focuses on the entire development process that begins with the user's requirements and ends with the software system. Most of the above described methods focuses on the design of hypermedia applications. Figure 5-3 depicts the phases each method covers (core workflows in Unified Process terminology). Requirements capture and testing are only taken into account by HFPM, WAE-Conallen and Lowe-Hall.

- **A Use-Case-driven process**

There are two different ways to capture the requirements of the system to be built. First, the traditional functional specification that finds out what the system is supposed to do. Second, the use case strategy that consists of finding out the system functionality from the user's perspective, i.e. what the system is supposed to do for each user. Use-case driven means that the development process proceeds through a

series of models derived from the use cases. Use cases are found, are specified and are the source for the analysis model and test cases. Use cases are not sufficient to define the analysis and design model, these models are also based on the architecture definition.

The hypermedia design methods started with the conceptual analysis or design with exception of SOHDM, WSDM and HPFM. SOHDM is a scenario-based approach that uses these scenarios as source for the domain analysis. WSDM describes the domain from the user perspective; it is a user-centric approach. HPFM and WAE-Conallen proposes use-case modeling for the software requirements modeling task.

- **A specification based on the architecture of the system**

The Unified Process sees the architecture as a complement to the use cases. The use cases define the functionality of the software system. The architecture specifies the organisation of a software system, the structural elements and their interfaces. The architecture is described by a set of five models: use case models, analysis model, design model, deployment model and implementation model.

All methods described above construct at least a static model of the domain; they also proposes a graphic representation of these models. These models are part of the description of the architecture of the software system.

- **Component-based implementation**

The Unified Process is a component-based methodology as the software systems being built are made up of software components interconnected via interfaces. For all the hypermedia methods to perform a component-based architecture is not a restriction posed by the methodology. It is a decision of the designer to perform a component-based design and implementation or not.

- **An incremental and iterative process**

An incremental and iterative process provides the strategy for developing software systems in small manageable steps. The iterations in the inception and elaboration phases of the Unified Process are concerned with establishing the scope of the project, removing critical risks and base lining the architecture. In the construction and transition phases the objective is to implement and integrate the components step by step, i.e. in small increments.

All hypermedia methods can be applied in successive iterations, but only a few proposes it explicitly, such as OOHDM, WAE-Conallen, Lowe-Hall and the Oestereich approach. For the Unified Process an iterative and incremental development process is one of the cornerstones of the method.

- **Initiating the development process with a feasibility study**

HPPM and the Lowe-Hall approach suggest a first step that involve a feasibility study. The purpose is to evaluate the viability of the project to convert a vision into a software system. The Unified Process just mentions it by the way. But this step is of particular importance, if an expensive adaptive software system is to be developed.

- **Elaboration of a business model**

Business modeling is used in the Unified Process to describe the business processes of an organisation in terms of business use cases and business actors corresponding to business processes and customers, respectively. The business model represents the business from the usage perspective and outlines how it provides value to the users. Although the business aspects are relevant in most hypermedia systems, only WSDM proposes to build a business object model.

- **Planning and assessment**

The Unified Process proposes the elaboration of a project plan and a project assessment and so, briefly, do HPPM and the engineering approach of Lowe-Hall. The project plan comprises of a list of milestones, at least one for each phase, a time schedule, evaluation criteria and a number of iterations for each phase. The objectives of the assessment are to examine what has been accomplished since the last evaluation, to review progress against the project plan and to determine if quality requirements are satisfied.

- **Risk management**

Risk management consists of risk identification, risk description, impact analysis, assignment of priorities, risk monitoring, determination of risk responsible, risk mitigation and alternative plans in case risk materialises. The Unified Process gives only a set of brief explanations, hints and rules for risk management. Only Lowe and Hall and WAE-Conallen mention the importance of risk management. Although general risk management concepts and techniques can be used, the hypermedia development process has its own risk factors.

- **A model-based development**

Models are built, updated and consolidated in the Unified Process in all core workflows and in all iterations of the phases: inception, elaboration, construction and transition. These models are built based on the techniques and notations defined by the UML. All other methods mentioned in this chapter also construct one or more static and/or dynamic models. The difference is that they do not use the Unified Modeling Language with the exception of OOHDM (only for the conceptual model) and WAE-Conallen.

## **5.5      Lessons Learned from the Comparative Study**

The comparative study of the most relevant engineering approaches for hypermedia and the comparison between the Unified Process and these development methods presented in this chapter are used as basis for the methodology developed during this work. Chapter 6 and 7 present the UWE (UML-based Web Engineering) approach. UWE comprises a methodology, a set of techniques and a notation. It covers the whole life cycle of hypermedia applications, from requirements capture to maintenance. Special attention is paid to quality management, i.e. validation of requirements, verification of the design and testing. First results for non-adaptive systems are included in Koch (2000a).

The evolution of hypermedia development methods can be compared to the steps taken by traditional software development methodologies. As Lowe and Hall (1999) stress, development processes for hypermedia will continue to evolve and improve. State of the art reports and comparative studies like this, allow for a better understanding of what is missing and failing in the current methods. It can be helpful in showing in which direction research has to continue. On the other hand, this chapter describes the strength of each method, providing ideas on how to combine these strengths in an unified and improved approach

*" Object orientation as a competitive advantage"*

*Ivar Jacobson,*

*American Programmer,*

*October 1992.*

## **6** Modeling Techniques for the Design of Adaptive Hypermedia

The development of hypermedia and adaptive systems differs from the development process of general software in several dimensions. These differences – outlined in the introduction of the previous chapter – are mainly related to the navigation facilities of the hypermedia structure, the role of the user, the people involved in the development and the maintenance process. Non-functional requirements have to be addressed by a development process for adaptive systems, such as security and privacy that is a concern of adaptive hypermedia applications. Adaptive hypermedia poses ethical and legal questions, as it allows for individual identification of the user and her preferences.

If we limit ourselves to the design steps, the main differences observed between the design of adaptive hypermedia solutions and other software applications are the central role of the user, the heterogeneity of the designer group, the hypertext structure composed of nodes and links, the need for navigation assistance, the multimedia content, the observation of user behaviour as well as the dynamic adaptation of contents, navigation and presentation. Thus, the design is centred around three main aspects of the hypermedia paradigm and two additional aspects of adaptive systems. The first three are the content, navigation structure and presentation with the goal to construct a conceptual model of the domain, and a navigation and presentation model of the application. The latter two are the user modeling and the adaptation. Treating these five aspects separately during design will pay off in the maintenance phase. It can be observed, that the domain, user and

adaptation models of the Munich Reference Model described in Chapter 4 are covered here by the design.

The need of authoring support for the development of Web applications is outlined by Nielsen (1999), who recommended the separation of content and presentation and emphasised the importance of a better navigation design. Authoring support for adaptive hypermedia applications is proposed by Wu, Houben and De Bra (1999). This authoring support consists of three main implemented classes, which are used to define the data model and include operations for authoring support. In that work a domain model, a user model and a teaching model are defined, but neither a methodology nor visual modeling is supported.

In this chapter the focus is on the analysis and design of the UWE (UML-based Web engineering ) approach. The techniques developed are then embedded in the development process that is described in detail in Chapter 7. The objective is to fill the methodical assistance gap in the development of adaptive hypermedia applications.

The analysis and design techniques presented in this chapter are based on a UML extension for hypermedia (Koch & Mandel, 1999; Baumeister, Koch & Mandel, 1999; and Hennicker & Koch, 2000a) and on a first approach to a method for adaptive hypermedia (Koch, 1998). These techniques are also known as UML-based Hypermedia Design Method (UHDM).

The methodology — part of the UML-based Web Engineering approach — consists of a notation and a method. The notation is a “lightweight” UML profile for hypermedia in general and for adaptive hypermedia in particular. It is defined as a set of stereotypes for (adaptive) hypermedia. These stereotypes are built using the UML extension mechanism (UML, 1999). They are used to indicate the descriptive and restrictive properties that the modeling elements have in comparison to standard UML elements. The advantage of using UML is that it is standard. The advantage of using a UML profile is that any practitioner with a general UML background is able to understand a model based on these specialisations (Selic, 1999). Selic, the author of the UML profile for the real-time domain, stresses that the resulting language remains compact, because the refinements fully respect the general semantics of their parents concepts and retain its “universal” quality.

The method consists of the constructing of six analysis and design models. Figure 6-1 shows the models represented as UML packages related by trace dependencies. The construction is performed in an iterative and incremental design process. The modeling activities are the requirements analysis, conceptual, user model, navigation, presentation and adaptation design.



The main artifacts produced by the design method of UWE are the following:

- a use case model that captures the system's requirements,
- a conceptual model for the content (domain model),
- a user model,
- a navigation model that comprises a navigation space model and a navigation structure model,
- a presentation model that comprises static and dynamic models (presentation structure model, presentation flow model, abstract user interface model and object lifecycle model), and
- an adaptation model.

Note that if the specific steps related to adaptation are suppressed, the methodology is applicable to the development of hypermedia applications – including Web applications – in general. The focus of the methodology is on the structure of the navigation and presentation and less on the architecture of the adaptive hypermedia application. The objective is to describe a systematic construction of the models, identifying as many steps as possible, which can be performed in an automatic way. This thus provides the basis for a systematic mechanism for adaptive hypermedia applications.

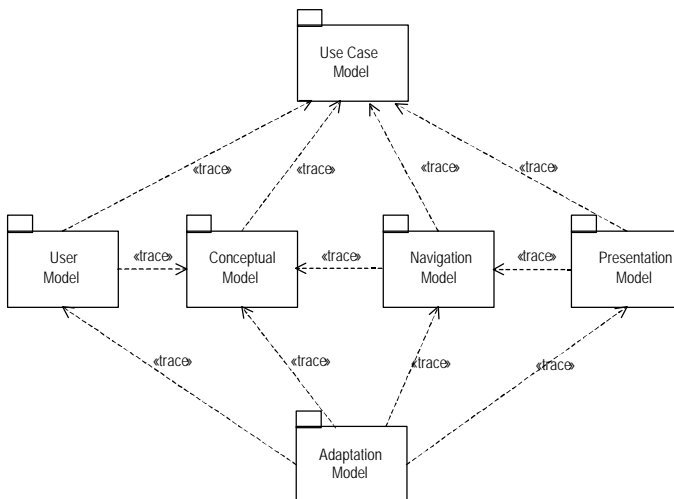


Figure 6-1: Models built during the Design for Adaptive Hypermedia Applications

Sections 1 to 6 of this chapter present a description of these models. Subsections are included for each model detailing the modeling elements, giving an example and enumerating a set of rules for the systematic construction of the model.

## 6.1 Use Case Model

Following the Unified Software Development Process of Jacobson, Booch and Rumbaugh (1999) use cases for capturing the system's requirements is proposed. It is a user-centred technique that forces to define who are the users (actors) of the application and offers an intuitive way to represent the functionality an application has to fulfil for each actor. For non-functional requirements see Chapter 7.

### Modeling Elements

The main modeling elements used for use case modeling are: *actors* and *use* cases. They can be related by *inheritance*, *include* or *extend* relationships. All these modeling elements as well as the package and view mechanisms are used with the semantics defined in the UML (1999) and graphically represented with the UML notation.

### Example

As a running example to illustrate the techniques presented through the whole chapter, the Web site of an adaptive online library (personalised online library) is used. This *Online Library* offers information about publications to registered and anonymous users. The publication information comprises journals, books and proceedings. These are described by a title, a number, a publisher, a publishing date, a set of articles and authors for each article. Books consists of exactly one article whose title is the same as the book title. In addition, for each article a set of keywords is stored.

The user is modeled by tracking her interest in articles and registering the articles she visits. The user can also mark articles (bookmarks) as being of special interest. A list of personal keywords for each user is administrated by the system. The list is initialised by the user and is updated either by the user or the system. The system performs the updating in accordance with the observations on the user's behaviour (in this case limited to the articles she marks or visits frequently). The list can include positive as well as negative keywords. Negative keywords are used to hide irrelevant publications and articles from the user.

To summarise, the *Online Library* offers users the following functions:

- access to the publication information as an anonymous or a registered user,
- dynamic up-dating of a user model,
- different search possibilities for publications, articles and authors,
- search mechanisms for articles already visited,
- dynamic adaptation of the search results to the current state of the user model,
- customisation of the presentation of the articles,
- notification of articles recently published by e-mail or included in a news page.

Figure 6-2 depicts the use case model for the *Online Library*.

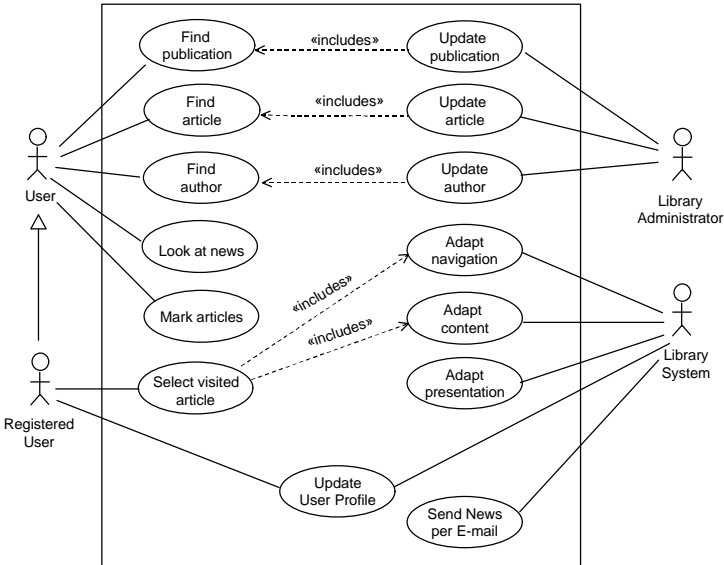


Figure 6-2: Use Case Model for the Online Library Application

---

## **Method**

To build the use case model of a Web application the following steps are suggested (Kruchten 1999, Schneider & Winters, 1998, etc.):

1. Find the actors.
2. For each actor search the text for activities the actor will perform.
3. Group activities to use cases.
4. Establish relationships between actors and use cases.
5. Establish “include” and “extends” relationships between use cases.
6. Simplify the use case model by defining inheritance relationships between actors and/or between use cases.

For each use case a detailed description can be provided in terms of (primary and secondary) scenarios, for instance following the guidelines of Schneider and Winters (1998). The activities flow of tasks related to a use case can be represented by a UML activity diagram.

## **6.2**

---

## **Conceptual Model**

The design of adaptive hypermedia applications builds on the requirements specification in the same way as the design of software applications in general does. UWE as a UML-based approach proposes use cases for capturing the requirements. UWE provides a user-centred technique that forces developers to define who are the users (actors) of the application, how a user model is built and offer an intuitive way to represent the functionality that an adaptive application has to fulfil for each actor.

The conceptual design is based on these use cases and includes the objects involved in the typical activities users will perform with the application. The conceptual design aims to build a conceptual model, which attempts to ignore as many of the navigation paths, presentation and interaction aspects as possible. These aspects are postponed to the navigational and presentational steps of the design.

---

## **Modeling Elements**

The main modeling elements used in the conceptual model are: *class* and *association*. These are represented graphically by the UML notation (1999). If the

conceptual model consists of many classes, it is recommended that they be grouped using the UML *package* modeling element.

- *class*

A class is described by a name, attributes, operations and variants. The optional compartment named *variants* is added to some classes of the conceptual model (see Figure 6-3). It contains additional information used for the *adaptive content* functionality, i.e. to present different or additional content to the user according to the current state of her user model. Content adaptation mechanisms are described in Chapter 2.

- *associations and packages*

Associations and packages are used as in standard UML class diagrams.

- *tagged values*

Tagged values are attached to modeling elements to extend the properties of the modeling element.

Classes defined in this step are used during navigational design to derive nodes of the hypermedia structure. Associations will be used to derive links.

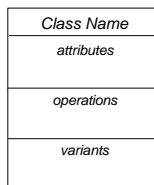


Figure 6-3: Class with additional Compartment Variants

---

### **Example**

The conceptual model of the *Online Library* is shown in Figure 6-4. The example is limited to the core data and functionality, although many other aspects should be included in the *Online Library* in an incremental and iterative process. These aspects could be additional classes and more advanced search functions, such as editors of publications, classification of authors, search by author name and article title. In addition to the search engines, authoring functions must be incorporated to allow for a visible and modifiable user model. Neither this user model authoring nor the administration of the publication database is within the scope of this chapter.

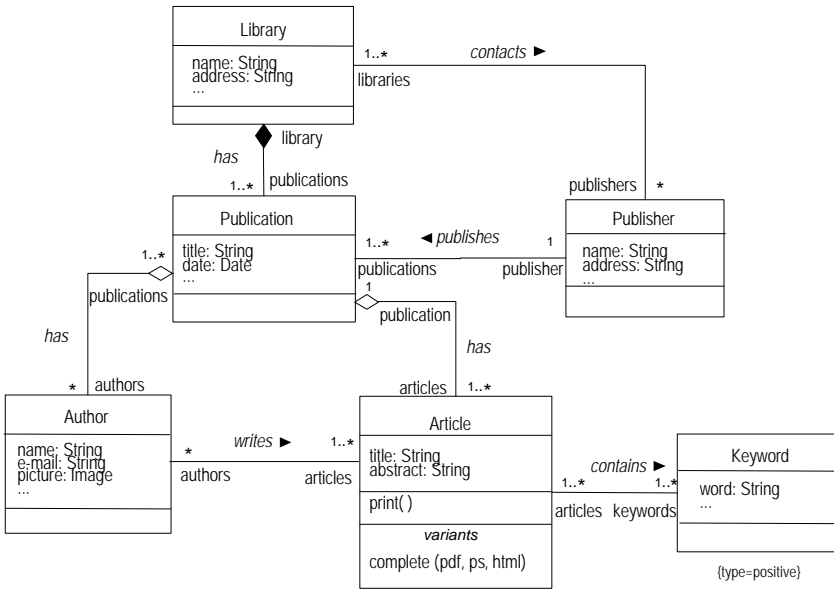


Figure 6-4: Conceptual Model of the Online Library Application

## Method

The developer can follow well-known object-oriented modeling techniques to construct the conceptual model, such as:

1. Find classes.
2. Specify the most relevant attributes and operations.
3. Determine associations between classes.
4. Define inheritance hierarchies.
5. Find dependencies.
6. Use tagged values.
7. Define constraints.

The detailed use case descriptions serve as an input for modeling the content of the application. The result of these activities is the conceptual model represented as a UML class model. An example is depicted in Figure 6-4.

## 6.3 User Model

The user model is designed with the purpose of establishing which user attributes will be chosen to elaborate a user profile, to determine how these attributes are related to each other and how they are related to the elements of the domain. A class diagram is used for the representation of a static user model. State diagrams can be used to represent object life cycles of user attributes and to show dependencies between state transitions. Most of these state transitions are defined by the rules of the adaptation model.

The user model represents knowledge, goals and/or individual features, such as preferences, interests and tasks of the users. The user model is the view the system has of the user, i.e. the system's belief about the user's knowledge (defined in Chapter 3 as  $B_S K_{up}$ ). Its main purpose is to influence user interface presentation and/or generation. User models also are used to administrate user roles for users and user groups. A user role express the users' rights within the application.

### Modeling Elements

The modeling elements used in the user model are: *class* and *association*. If the user model consists of many classes, it is recommended to group them using the UML *package* modeling element. Based on the user properties classification presented in Chapter 3, the following sub-models can be created: domain-dependent knowledge, background knowledge and cognitive model. Each sub-model may then be represented as a package.

### Example

The following characteristics of the users are included in the user model of the *Online Library* application: articles the user visits, articles that are marked by the user, positive and negative keywords, preferences the user chooses to be informed about new articles and the type of file she selects for the download of the articles. Some of the values of these user attributes are updated dynamically by the system (e.g. visited articles); others are set by the user and can only be changed by her (e.g. file type). The user model for the *Online Library* is shown in Figure 6-5.

Class Article of the domain model is appended to the diagram to show how the user model is related to the domain model. Class Visited registers how often the user visits an article. The class Marked models the articles that are marked by the user (bookmarks). They are part of the domain dependent knowledge, i.e. one instance of these classes is required for each instance of the class Article and of the class user. Class UserKeyword models themes of interest. It is considered as background knowledge and its instances are not related to specific instances of domain classes. Classes FileType and News model preferences of the user.

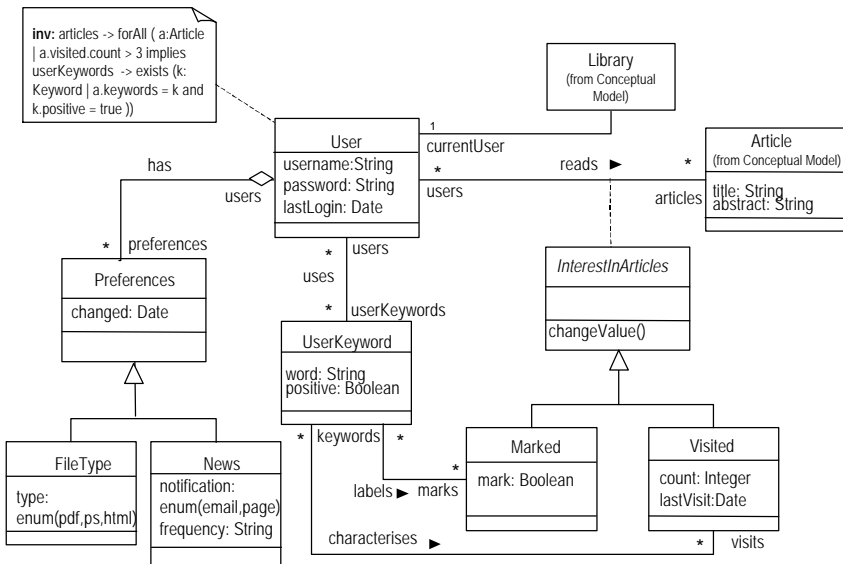


Figure 6-5: User Model of the Online Library Application

User attributes can be grouped in packages. With the only purpose to illustrate the package building, the user attributes defined in the *Online Library* are grouped in the above mentioned packages (see Figure 6-6).



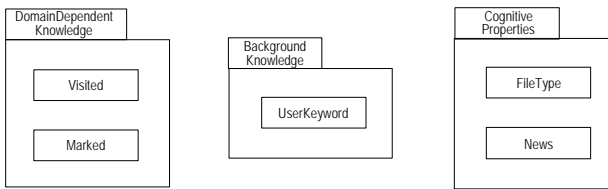


Figure 6-6: User Model Packages

---

## Method

This step requires the same abilities of the developer as the design of the domain model does. The requirements captured as use cases can be helpful to identify classes for the user model. The structure of the user model of an application is based on the user model of the reference model presented in Chapter 4, i.e. it contains a class user and a set of user attributes.

1. Decide which user attributes that describe characteristics of the users are relevant for the application.
2. Define a class for each identified user attribute.
3. Determine ranges of values for the attributes of these classes.
4. Establish which user attributes are domain-dependent, and which represent background knowledge or model cognitive properties of the user.
5. Identify associations between domain dependent user attributes and classes of the domain model.
6. Define constraints.
7. Group user attributes in packages of domain-dependent knowledge, background knowledge and cognitive properties.

The results of these activities are the user model represented as a UML class model. An example is depicted in Figure 6-5.

## 6.4

## Navigation Model

---

Navigation design is a critical step in the design of a hypermedia application. Even simple applications with a non-deep hierarchical structure will very quickly become complex as a result of adding new links. On the one hand, additional links

improve navigability; on the other hand, however, they increase the risk of loss of orientation. Building a navigation model is not only helpful for the documentation of the application structure, it also allows for a more structured increase in navigability. Maintaining coherence as the user moves across the application is not simply a matter of providing links. It depends on the structure of the overall system, which is related to its underlying communicative performance.

The navigation model comprise the navigation space model and the navigation structure model. The former specifies *which* objects can be visited by navigation through the hypermedia application. *How* these objects are reached is defined by the navigation structure model. These models are described in subsections 6.4.1 and 6.4.2, respectively.

### 6.4.1 Navigation Space Model

In the process of building the navigation space model the developer takes crucial design decisions, such as which view of the domain model is needed for the application and what navigation paths are required to ensure the application's functionality. The designer's decisions are based on the domain model, use case model and the navigation requirements that the application must satisfy.

A set of guidelines is proposed for modeling the navigation space. A detailed specification of associations, their multiplicity and role names, establish the base for a semi-automatic generation of the navigation structure model.

#### Modeling Elements

Two modeling elements are used for the construction of the navigation space model: navigation classes, external nodes and navigation associations, which express direct navigability. They are the pendant to page (node) and link in the Web terminology.

- *navigation class*

A navigation class models a class whose instances are visited by the user during navigation. Navigation classes will be given the same name as the corresponding domain classes. For their representation the UML stereotype «navigation class» is used (see Figure 6-7). Navigation classes may contain derived attributes. These attributes are derived from domain classes that are not included in the navigation model. The formula to compute the derived attribute can be given by an OCL expression. A derived attribute is denoted in UML by a slash (/) before its name.

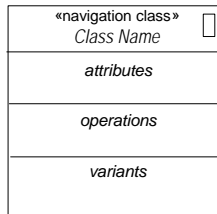


Figure 6-7: Navigation Class

- *external node*

An external node models a navigation target belonging to another hypermedia application, i.e. the node is not part of the application that is being modeled, but can be reached from a source node of the application. For their representation the UML stereotype «external node» is used (see Figure 6-8).



Figure 6-8: Stereotype for External Node

- *direct navigability*

Associations in the navigation space model are interpreted as representing direct navigability from the source navigation class to the target navigation class. However, their semantics are different from the associations used in the conceptual model. To determine the directions of the navigation the associations of this model are directed (possibly bi-directed). This is shown by an arrow that is attached to one or both ends of the association. Moreover, each directed end of an association is named with a role name and equipped with an explicit multiplicity. If no explicit role name is given, the following convention is used: if the multiplicity is less than or equal to one, the target class name is used as the role name; if the multiplicity is greater than one, the plural form of the target class name is used. In the diagram shown in Figure 6-9 all associations are implicitly assumed to be stereotyped by «direct navigability».

**Example**

Figure 6-9 shows the navigation space model for the Web Site of the *Online Library* application. The diagram includes one invariant that defines the set of new articles which is navigable for the user. The articles belonging to this set depend on the last login date of the user. The constraint is attached as a note to the correspondent class. Alternatively, constraints can be specified separately as the invariant listed below the diagram.

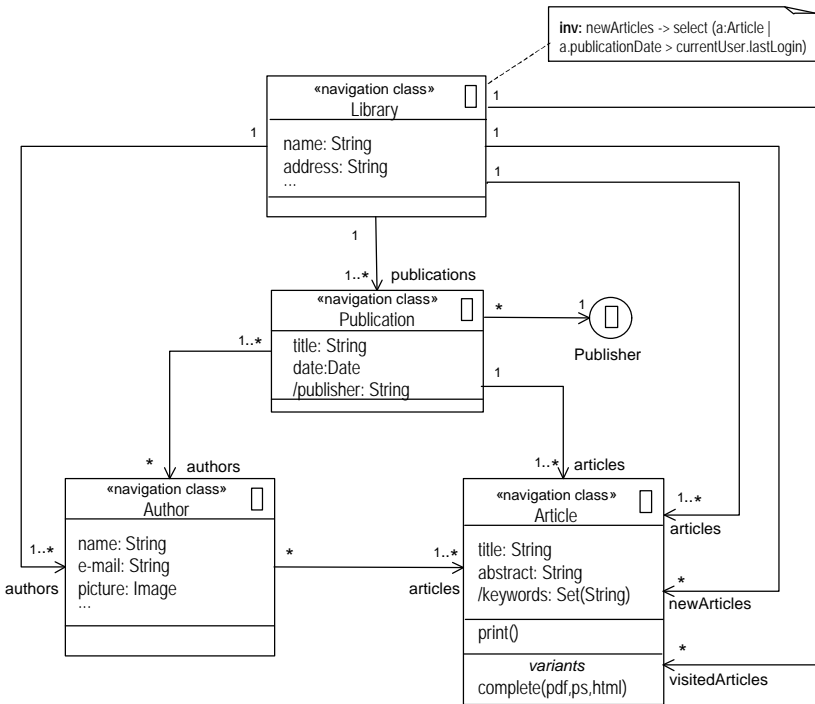


Figure 6-9: Navigation Space Model of the Online Library Application

**context** Library

**inv:** visitedArticles → select ( a:Article | a.visited.count > 1 and lastVisit.year = currentYear )

---

**Method**

The navigation space model that is built with the navigation classes and navigability associations as shown in Figure 6-9 is graphically represented by a UML class diagram.

Although there is obviously no way to automate the construction of the navigation space model, there are several guidelines that can be followed by the developer:

1. Include classes of the conceptual model that are relevant for the navigation as navigational classes in the navigation space model (i.e. navigation classes can be mapped to conceptual classes). If a conceptual class is not a visiting target in the use case model, it is irrelevant in the navigation process and therefore omitted in the navigation space model (such as the classes Publisher and Keyword in our example).
2. Keep information of the omitted classes (if required) as attributes of other classes in the navigation space model (e.g. the newly introduced attribute publisher of the navigation class Publication and attribute keywords of the navigation class Article). All other attributes of navigation classes map directly to attributes of the corresponding conceptual class. Conversely, exclude attributes of the conceptual classes that are considered to be irrelevant for the presentation in the navigation space model.
3. Associations of the conceptual model are kept in the navigation model. Often additional associations are added for direct navigation to avoid navigation paths of length greater than one. Examples are the newly introduced navigation associations between Library and Author and between Library and Article (all articles).
4. Add additional associations based on the requirements description or the scenarios described by the use case model. Hence, an association for visited articles and for new articles is added.
5. Add constraints to specify restrictions in the navigation space as shown in the UML note of Figure 6-9 or as the invariant that defines the set of visited articles below Figure 6-9.

**6.4.2**

---

**Navigation Structure Model**

The navigation structure model describes how the navigation can be performed using access elements such as indexes, guided tours, queries and menus. Technically, the navigation paths together with the access elements are presented by a class model which can be systematically constructed from the navigation space

model in three steps: Firstly, the navigation space model is enhanced by indexes, guided tours and queries. Secondly, menus are derived directly from the enhanced model. Menus represent possible choices for navigation. Thirdly, properties for access primitives are introduced to model adaptive navigation.

The resulting navigation structure model defines the structure of nodes and links in the adaptive hypermedia application showing how navigation is supported by the access primitives as well as depicting where the adaptive navigation is applied.

The following subsections describe how the navigation structure model is built step by step.

### 6.4.2.1 Including Access Primitives

Access primitives are additional navigation nodes required to access navigation objects. The following access primitives are defined: index, guided tour, query and menu. In this section the first three are described and used to refine the navigation space model. Menu is treated separately in the next subsection.

---

### Modeling Elements

The following modeling elements are used for describing indexes, guided tours and queries. Their stereotypes and associated icons are defined in (Koch & Mandel, 1999); some of the icons stem from Isakowitz, Stohr and Balasubramanian (1995).

- *index*

An index allows direct access to instances of a navigation class. This is modeled by a composite object, which contains an arbitrary number of index items. Each index item is in turn an object, which has a name that identifies the instance and owns a link to an instance of a navigation class. Any index is a member of some index class, which is stereotyped by «index» with a corresponding icon. An index class must be built conform to the composition structure of classes shown in Figure 6-10. Hence the stereotype «index» is a restrictive stereotype in the sense of Berner, Glinz and Joos (1999). In practice, the shorthand notation shown in Figure 6-11 is always used.

Note that in the short form the association between Index and NavigationClass is derived from the index composition and the association between IndexItem and NavigationClass.

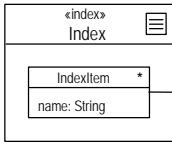


Figure 6-10: Index Class

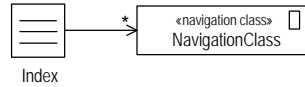


Figure 6-11: Shorthand Notation for Index Class

- *guided tour*

A guided tour provides sequential access to instances of a navigation class. For classes, which contain guided tour objects the stereotype «guidedTour» is used and its corresponding icon depicted in Figure 6-12. Any guided tour class

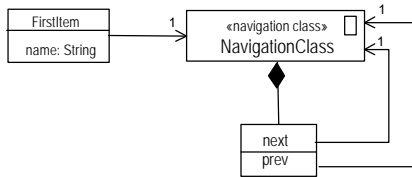


Figure 6-12: Guided Tour Class

must be built conform to the composition structure of classes shown in Figure 6-12. Each NextItem must be connected to a navigation class. Guided tours may be controlled by the user or by the system. Figure 6-13 shows the shorthand notation for a guided tour class.

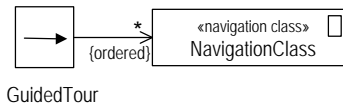


Figure 6-13:: Shorthand Notation for Guided Tour Class

- *query*

A query is modeled by a class which has a query string as an attribute. This string may be given, for instance, by an OCL select operation. For query classes the stereotype «query» is used and the icon depicted in Figure 6-14. As shown in Figure 6-14, any query class is the source of two directed

associations related by the constraint {xor}. In this way a query with several result objects is modeled to lead first to an index supporting the selection of a particular instance of a navigation class. The query results can alternatively be used as input for a guided tour.

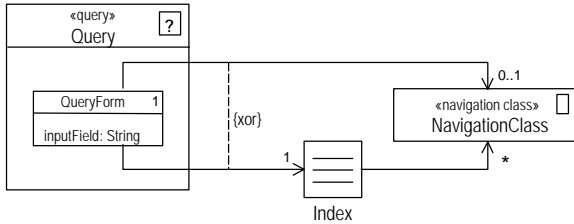


Figure 6-14: Query Class

Figure 6-15 shows the shorthand notation for a query class in combination with an index class or with a guided tour.

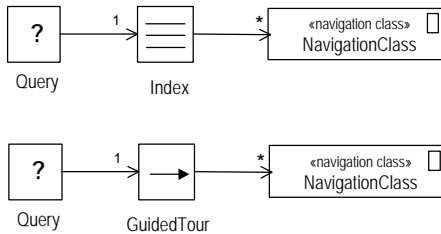


Figure 6-15: Shorthand Notation for Query

---

### Example

Figure 6-16 shows how the navigation space model of the Web Site of the *Online Library* can be enhanced by indexes, guided tours and queries.

Note that the access to all authors or all articles of the Library is restricted in this example to search mechanism via queries. News is reached directly via an index and the use of a guided tour is illustrated by a guided tour through the articles of a publication



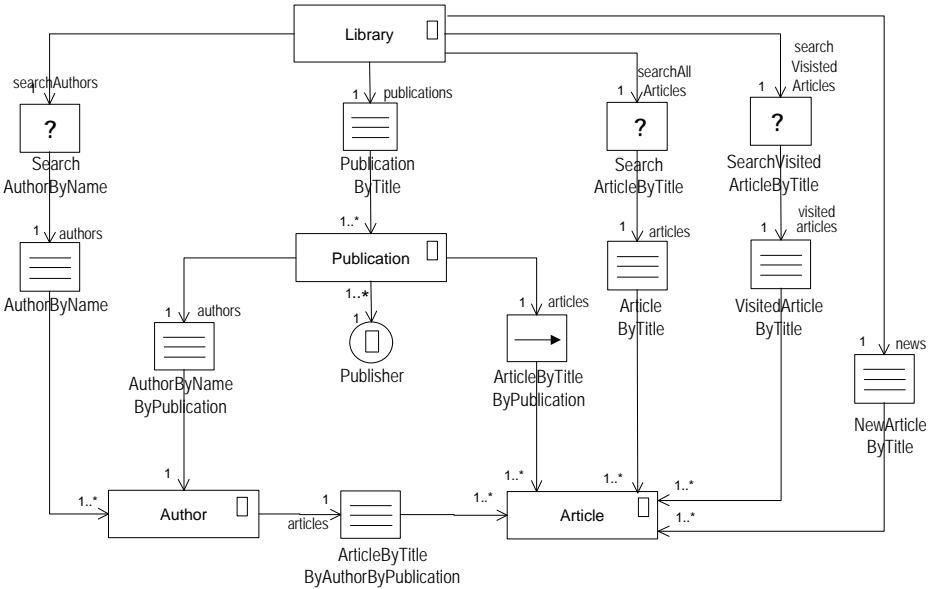


Figure 6-16: Navigation Structure Model (First Step) of the Online Library Application

## Method

The enhancement of a navigation space model by access elements of type index, guided tour and query follows certain rules, which can be summarised as follows:

1. Replace all bi-directional associations, which have multiplicity greater than one at both associations ends by two corresponding unidirectional associations.
2. Replace all bi-directional associations, which have multiplicity greater than one at one association end with one unidirectional association with an directed association end at the end with multiplicity greater than one. The navigation in the other direction is guaranteed by the use of navigation trees introduced later in the design.
3. Consider only those associations of the navigation space model, which have multiplicity greater than one at the directed association end.

4. For each association of this kind, choose one or more access elements to realise the navigation.
5. Enhance the navigation space model correspondingly. Role names of navigation in the navigation space model are now moved to the access elements (compare Figure 6-9 and Figure 6-16).
6. Add constraints to model invariants and conditions. They are deduced from the detailed description of the use case model.

In item 4 the task of the designer is to choose appropriate access elements. However, note that it is also possible to fully automate this step by making the choice of an index a default design according to an attribute with the property {key} of the target navigation class.

#### 6.4.2.2 AddingMenus

In this step, access primitives of type menu are added to the navigation structure model.

#### Modeling Elements

The modeling element *menu* is an additional access primitive that can be added to the list presented in the previous step. Its stereotype is defined by Koch and Mandel (1999).

- *menu*

A *menu* is an index of a set of heterogeneous elements, such as an index, a guided tour, a query, an instance of a navigation class or another menu. This is modeled by a composite object which contains a fixed number of menu items. Each menu item has a constant name and owns a link either to an instance of a navigation class or to an access element. Any menu is an instance of some menu class which is stereotyped by «menu» with a corresponding icon. A menu class must be built in accordance with the composition structure of classes shown in Figure 6-17. Hence the stereotype «menu» is again a restrictive stereotype according to the classification of stereotypes given by Berner, Glinz and Joos (1999). The property {frozen} is attached to each name attribute in a menu item class to show that menu items have fixed names. Nevertheless, the same menu item class may have different instances since there may be menu items with the same name but linked to different objects.

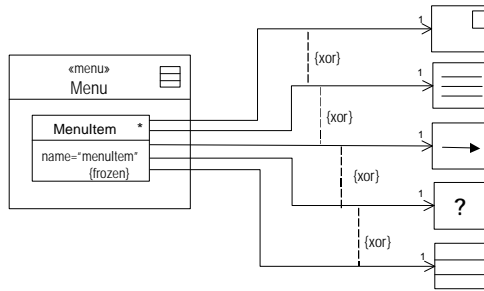


Figure 6-17: Menu Class

For a convenient notation of menu classes in navigation structure models the shorthand notation shown in Figure 6-18 is used. This is a somewhat more flexible extension than the extension mechanisms of UML allows, since it includes a variable number of compartments with the names of the menu items.

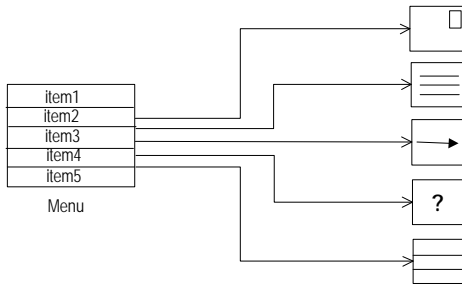


Figure 6-18: Shorthand for Menu Class

### Example

Figure 6-19 shows how menus enriched the navigation structure model of the previous subsection where each menu class is associated with a composition association to a navigation class. Note that the role names occurring in the previous model (Figure 6-16) are now names of corresponding menu items.

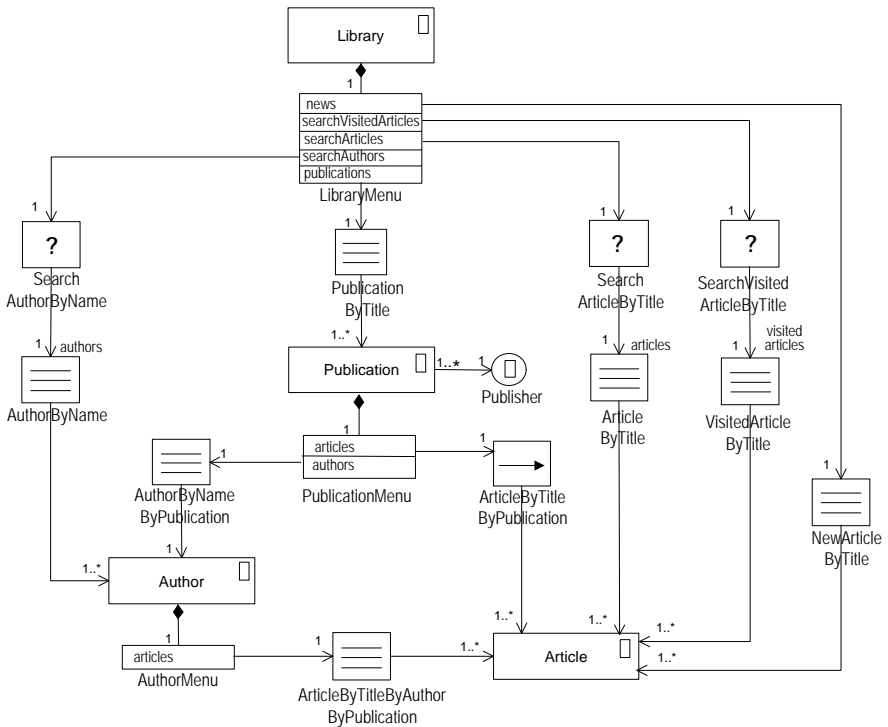


Figure 6-19: Navigation Structure Model (Second Step) of the Online Library Application

## Method

The navigation space model is enhanced by access elements of type menu following certain rules which can be summarised as follows:

1. Consider those associations, which have as their source a navigation class.
2. Associate to each navigation class, which has (in the previous model) at least one outgoing association, a corresponding menu class. The association between a navigation class and its corresponding menu class is a composition.
3. Reorganise a menu in a menu with sub menus.

4. Introduce for each role, which occurs in the previous model at the end of a directed association a corresponding menu item. By default, the role name is used as the constant name of the menu item.
5. Any association of the previous model, which has as its source a navigation class now becomes an association of the corresponding menu item introduced in step 4.
6. Add constraints to add precision to the model.

Note that all the steps in the above method can be performed in a fully automatic way. As a result a comprehensive navigation structure model of the application is obtained. The method guarantees that this model conforms to the pattern shown in Figure 6-20.

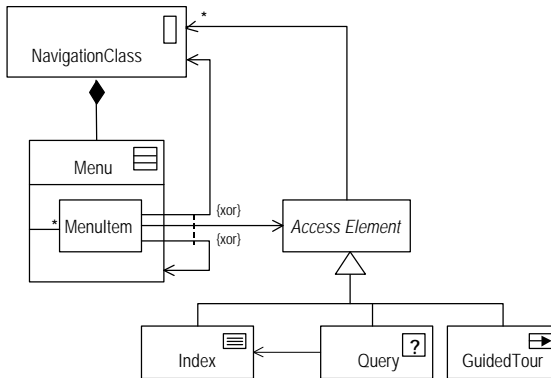


Figure 6-20: Pattern for Access Structures

### 6.4.2.3. Modeling Adaptive Navigation

An adaptive hypermedia system adapts navigation to the preferences, knowledge or tasks of the user. To model these adaptive navigation features, associations showing navigability and access primitives, such as indexes, guided tours and menus have to be annotated with properties to specify their adaptive behaviour.

Adaptive navigation can be performed by direct guidance, sorted links, annotated links, removed links and passive navigation (Brusilovsky, 1996b, and Wu, Houben & De Bra, 1999). A detailed description of these adaptive techniques is given in Chapter 2.

---

## **Modeling Elements**

The following properties for modeling elements are defined in order to specify adaptive navigation.

- *direct guidance*

Direct guidance is a property assigned to a guided tour or to a link. It specifies that the system decides, which is the “best” target node based on the current state of the user model.

- *sorted*

The list of index items of an index is given the property {sorted} to denote that these items are ordered to indicate the relevance they have for the user.

- *annotated*

An index or a menu is given the property {annotated} to indicate that the index or menu items are annotated to indicate the relevance they have for the user. How annotation is presented (colours, icons or symbols) is decided in the presentation design.

- *removed*

Index items and menu items can be removed if the system believes that they are not relevant for the user. To show that certain items of an index or a menu eventually are removed, the property {removed} is specified.

- *passive navigation*

{passive navigation} is a property added to an association to indicate that navigation is performed by the system. This is used to allow a reaction of the system to a passive behaviour of the user. A timeout is used to trigger passive navigation.

Note that hidden links can be modeled by annotation choosing the appropriate font, icon or colour. Also note that annotation and passive navigation can be used as a general mechanism for user orientation and guidance, i.e. without being based on a specific user model.

---

## **Example**

Figure 6-21 shows how the navigation structure model of the previous subsection is enhanced in this third step by properties for the access primitives and associations indicating passive navigation. Properties using UML notation are placed under the modeling elements.

An example of passive navigation is incorporated in the *Online Library*: in the case of user inactivity while visiting any article, the system navigates to the news, to call the user's attention.

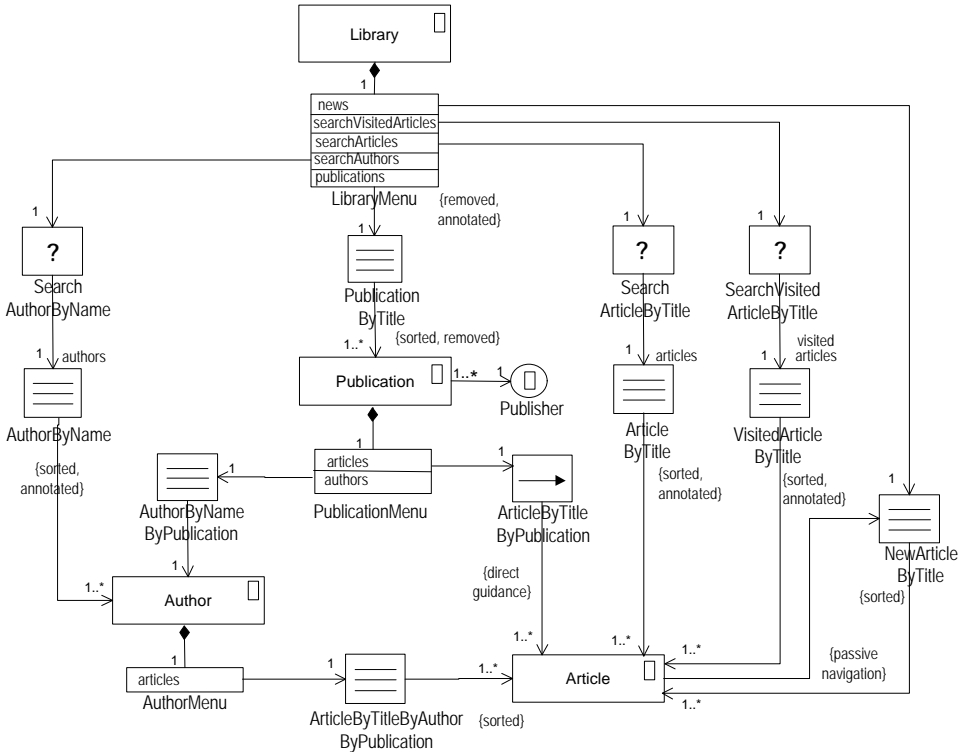


Figure 6-21: Navigation Structure Model (Third Step) of the Online Library Application

The following is an example of a constraint for the LibraryMenu. It describes the property {removed} specifying which menu items are removed if the user is anonymous.

**context** LibraryMenu  
**inv** removal of menu items by anonymous user:  
 library.currentUser → isEmpty

implies ( searchVisitedArticlesItem → isEmpty  
and newsItem → isEmpty )

In analogy, invariants can be defined for the other properties specified in the class diagram.

---

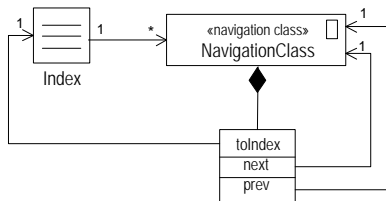
### **Method**

The navigation structure model can be enhanced by properties and additional associations to model adaptive navigation following certain guidelines which can be summarised as follows:

1. Specify {direct guidance} for a guided tour if the sequence of instances of the guided tour is generated based on the actual current state of the user model.
2. Choose the property {sorted}, {annotated} and/or {removed} for access primitives, such as indexes and menus, if links are sorted, annotated or removed according to the adaptation rules which take into account the current values of the user attributes in the user model.
3. Add associations with the property {passive navigation} to indicate that navigation is performed by the system when a timeout occurs. It can be performed in a user model dependent or independent mode.
4. Specify{annotated} for associations if the system offers general orientation or guidance to all type of users.
5. Describe the properties by OCL invariants.

#### **6.4.2.4 Other Modeling Elements**

This subsection shows how modeling elements of other methods can be defined as stereotyped classes in UML as well. The following examples are selected: the indexed guided tour of RMM and the navigation contexts of OOHDM.



*Figure 6-22: Indexed Guided Tour*



An indexed guided tour provides sequential access to instances of a navigation class as well as through an index. The first method that used this modeling element was RMM (Isakowitz, Stohr & Balasubramanian, 1995). Figure 6-22 depicts the structure of an indexed guided tour.

Navigation contexts allow the organisation of the navigation space in sequences that can be traversed following a predefined order. A navigation context consists of a defined set of instances of a navigation class, a predefined order and a mechanism that makes it possible to find the next and/or the previous instances of the sequence as well as to return to the starting point (index or menu). This is the basis for different presentations of a navigation class instance, depending on the context it is navigated, so called “InContext” classes of OOADM. Navigation contexts were introduced by Rossi (1996).

The use of navigation contexts may replace navigation maps or trees (see presentation model in the next section) or they can be used to include additional links. Navigation contexts have the disadvantage that they add unnecessary complexity to the navigation model. The combination of both techniques – navigation maps and navigation contexts – leads to an enriched navigation structure but with higher risk of “lost in the hyperspace”.

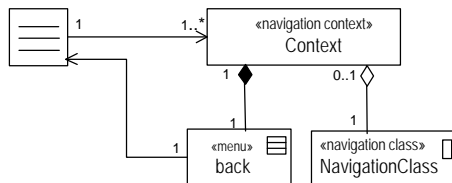


Figure 6-23: Navigation Context triggered by an Index

A simple context is provided by a sequence of navigation objects; each of them is connected to the previous and to the next object. This is modeled by a composite of a sequence of instances of a navigation class, for which an order is given. Attributes of the class are usually used for the definition of the ordering. The navigation context class includes a menu with menu items *previous*, *next and/or back* (to the index or to the previous menu). The menu items *previous* and *next* are responsible for the access to the previous and next objects in the sequence, respectively. A context class is built in conformity with the structure of classes shown in Figure 6-23.

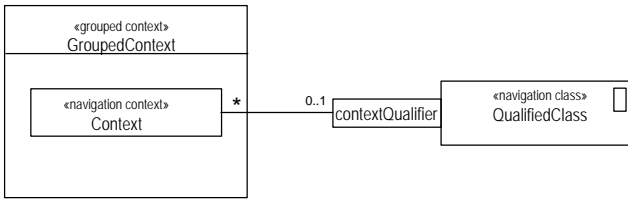


Figure 6-24: Grouped Context

A grouped context is a sequence of sequences of instances of a navigation class, i.e. a sequence of simple navigation contexts. It represents a partition of a sequence of navigation objects into sub-sequences determined by a common value of an attribute or a common object related by an association. Both cases can be reduced to one, as an attribute can also be modeled as a class with the appropriate aggregation. A grouped context is modeled by a composition of navigation contexts, which are obtained as a qualified partition of all instances of the navigation class (see Figure 6-24). The qualifier values are given by the names of the objects of the qualified class.

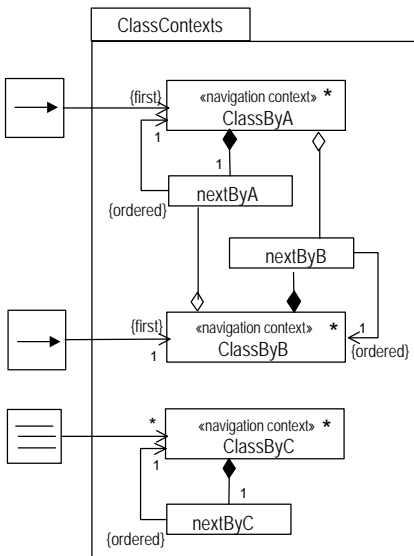


Figure 6-25: Context Package and Context Change

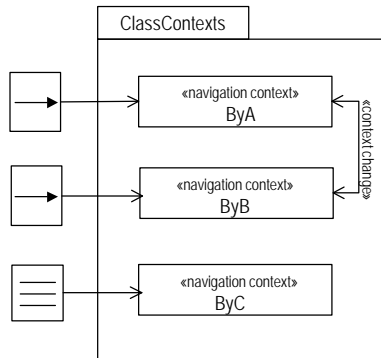


Figure 6-26: Shorthand for Context Package

Navigation contexts for a same navigation class can be grouped in a UML package. They are related by a stereotyped association called «context change» that makes it possible to continue navigation in another context and to return then to the original context. The condition for a context change is that the instances of a navigation class are part of both contexts. Within the package the default semantic is that all context changes are permitted when they are not explicitly drawn. Figure 6-25 shows three navigational contexts and the context changes between these navigational contexts. Figure 6-26 depicts a shorthand for a context package.

## 6.5 Presentation Model

The presentation model is the description of *where* and *how* navigation objects and access primitives will be presented to the user. Presentation design supports the transformation of the navigation structure model in a set of models that show the static location of the objects visible to the user, a schematic representation of these objects (pages in the Web design) and the dynamic behaviour of them. The schematic representation is similar to the sketching technique used by some user interface designer, but without having a precise notation for it as described by Sano (1996). Here a UML notation is chosen for the graphical representation.

Presentation design focuses on the structural organisation of the presentation, such as texts, images, forms and menus, and *not* on the physical appearance in terms of special formats, colours, etc. Such decisions are taken during the development of a user interface prototype or in the implementation phase. The layout of modeling elements in the presentation model, however, may provide hints, for example, about the position and the size of these elements relative to each other.

The following subsections show how the presentation model is derived from the navigation structure model. Presentation design uses additional information collected during requirements analysis. The static aspects are modeled by a presentation structure model and an abstract user interface model. The presentation structure model shows how windows and frames are filled with content and which content can be shown simultaneously. The abstract interface model sketches the content of each node.

A presentation model may yield different implementations depending on the restrictions of the target platform and the technology used. These include static and dynamic pages, client and server pages, one or multiple windows, etc. Important concepts of presentation modeling are windows, framesets and frames. The use of frames allows amongst other things, of the visualisation of the navigation space, usually presented as a navigation tree (also known as a navigation map).

In addition, dynamic aspects of the presentation are modeled using a presentation flow model and optionally object lifecycle models. UML sequence diagrams and state diagrams respectively are used for these models. Sauer and Engels (1999) also propose the use of this type of UML diagrams for the modeling of multimedia applications. Both models describe the behaviour of the presentation objects, i.e. the changes on the user interface when the user interacts with it or when the system reacts to internal events such as timeouts. The construction of a presentation flow model is normally proposed when a multiple-window technique or frame style is chosen. This specifies when windows are open, closed and when frames change their content. An object lifecycle model describes the behaviour of critical objects and how state transitions influence the status of other objects.

### **6.5.1 Abstract User Interface Model**

The objective of the abstract user interface model is to provide a technique and notation for the sketching of the user interface, i.e. how the content of a node (Web page) is presented to the user. The abstract user interface design, as mentioned above, mainly models the structural organisation of the presentation, such as texts, images, forms and menus, and *not* the layout characteristics, in terms of fonts, colours, special formats, etc. Such decisions are taken during the development of a user interface prototype during an early analysis stage or in the implementation phase. The abstract user interface model may, however, provide some hints, for example, on the position and the size of the user interface objects relative to each other. In order to construct a presentation model, one has to decide, on the one hand, which presentation elements will be used for the presentation of the instances of navigation classes and, on the other hand, which will be used for the presentation of the access elements.

The abstract user interface design may be considered an optional step as the design decisions related to the user interface can also be taken during the realisation of the user interface. However, the production of sketches of this kind is often helpful in early discussions with the customer.

---

#### **Modeling elements**

Instances of a presentation class are containers, which comprise modeling elements like texts, images, forms, buttons, video sequences, audio sequences, anchors, collections (i.e. lists of texts, images, etc.) or anchored collections (i.e. lists of anchors). A presentation class follows the composite rules depicted in Figure 6-27.

The following modeling elements (ten stereotyped classes and two properties) are proposed to describe the abstract user interface of an adaptive hypermedia application. Stereotypes for text, form, button, image, audio, video, anchor, collection and anchored collection as depicted in Figure 6-27 are presented in Baumeister, Koch and Mandel (1999).

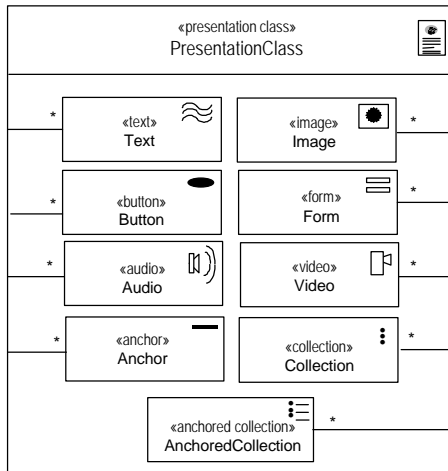


Figure 6-27: Presentation Class as Container other Presentation Modeling Elements

- *presentation class*

A presentation class models the presentation of a navigation class, an access primitive, (index, a guided tour, query or menu) or a composite of presentation classes. A presentation class is also a container for a set of other classes that model the presentation of the attributes of the presentation class. This is stereotyped by «presentation class» with a corresponding icon as shown in Figure 6-27.

- *text*

A text is a sequence of characters together with formatting information.

- *anchor*

An anchor is a clickable area, which is the starting point of a navigation and thus has associated a link to another node. Anchors are generally presented in the literature as part of links, seldom as independent objects. An anchor consists of a presentation together with a link. The presentation may be either a

text (even a single character), an image, a video, a group of mixed media, any interactive object or a whole document.

- *button*

A button is a clickable area, which has an associated action. Example of actions are `playVideo`, `displayImage`, `stopAudio` and `runApplet`. Note that anchors may be implemented as buttons, i.e. buttons can also be navigation triggers.

- *images, audio and video*

Images, audio and video are multimedia objects. Images can be displayed; audio and video can be started, stopped, rewound and forwarded. To provide this functionality user interface objects that allow for interaction, such as buttons or anchors may be associated with these multimedia objects.

- *form*

Forms are used to request information from the user. They supply information in one or more input fields or select options from a browser or checkbox. The semantics of this model element includes the display of the content, waiting for the user activity, the evaluation of the input and the trigger of the defined event.

- *collections and anchored collections*

Collections and anchored collections are model elements introduced to provide a convenient representation of frequently used composites. They avoid textual description by comprehension or by extension. A collection consists of a set of text elements. An anchored collection comprises a set of anchors. Whether the collection will be laid out horizontally or vertically is not specified; objects may still be arranged as a table.

- *adapted language*

Adapted language is a property that can be added to presentation objects. It indicates that the language used in the user interface to which the property is applied, is the language specified in the current state of the user model.

- *layout variant*

Layout variant is a property that can be added to presentation objects. It indicates that the modeling elements to which the property is applied, have a layout that is adapted to the current state of the user model.

---

### **Example**

Figures 6-28 to Figure 6-38 show some presentation classes that are part of the abstract user interface model of the *Online Library*.

Figure 6-28 shows the main menu of the application and Figure 6-29 depicts a variant of LibraryMenu that includes the AuthorMenu. These menus and other variants of the main menu are visible while navigating in the whole application.

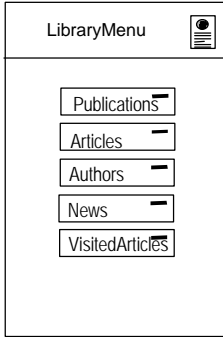


Figure 6-28: Presentation Class of Library Main Menu

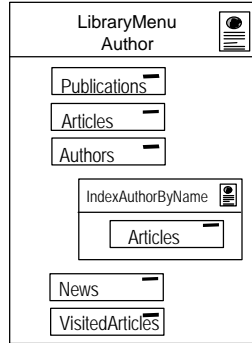


Figure 6-29: Presentation Class of Composite Library and AuthorMenu

Anchors of the menu may be annotated, i.e. underlined or change their colour, to show that they are selected. Another inheritance hierarchy for the main menu may be used to represent this kind of annotations.

Figure 6-30 and Figure 6-31 depict how the modeling elements described above are used to construct a template for the presentation of author and article, respectively.

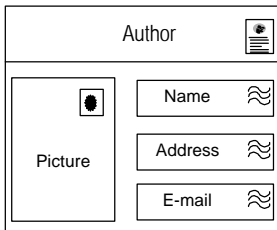


Figure 6-30: Presentation Class Author

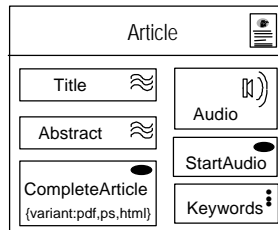


Figure 6-31: Presentation Class Article

Figure 6-32 shows the presentation that the user sees after she selects the author option in the LibraryMenu. It is a form to be filled by the user. As response to the send of this form, a list of authors matching the search input fields is shown as it is depicted in Figure 6-33, where she can select one author (an index item). The list of authors is sorted and annotated according to the values of the current user model.



Figure 6-32: Presentation Class  
*SearchAuthorByName*

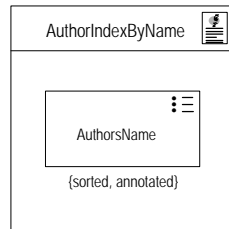


Figure 6-33: Presentation Class  
*AuthorIndexByName*

---

## **Method**

The abstract user interface model is constructed as a set of class diagrams, represented mainly as compositions. Each composition models one presentation class using a template to represent the content. A template for a Web page is then given by one presentation class composition or by the presentation classes that are presented in each frame of the Web page.

The following rules can be used as guidelines for the construction of the abstract user interface model:

1. Construct a presentation class for each navigation class occurring in the navigation structure model. A template must be provided to present the instances of the class, taking into account the given attributes. Stereotyped classes, such as «text», «image», «audio», «video» are used for attributes of primitives types and «collections» are used for lists, etc. Figure 6-30 shows the presentation class for an author and Figure 6-31 shows the presentation class for an article.
2. Construct a presentation class for each index and menu occurring in the navigation structure model. The presentation of an index or menu class consists usually of a list of anchors. Use stereotypes «anchored collection»



- and «anchor», respectively. Examples for menus and indexes are the LibraryMenu (Figure 6-28) and the AuthorIndexByName (Figure 6-33).
3. Construct a presentation class for each query and guided tour. Use a «form» stereotype for representing queries and for guided tours introduce an additional menu item (“next”, and “prev”) which makes it possible to navigate to the next and to the previous object within the guided tour.
  4. Add anchors to the presentation classes to allow creation and destruction of objects, or the execution of operations on objects.
  5. Specify properties, such as {direct guidance}, {sorted}, {annotated} and/or {removed} for buttons, anchors or anchored collections, if these properties are specified for their related guided tours, indexes or menus in the navigation structure model (see Figure 6-33).
  6. Specify properties {adapted language} and/or {layout variant} for each class that requires adapted presentation based on the user model state (these techniques for adaptive presentation are defined in Chapter 2).
  7. Add OCL constraints to specify these restrictions.

It must be ensured that there is only a finite set of navigation paths from the root class to each navigation or index class. For this purpose it is assumed that the given navigation structure model has no cycles, i.e. forms a directed acyclic graph. This is not a proper restriction since in any case, a presentation of the navigation tree is provided which, specifically, allows us to move backwards.

Concerning the presentation of a navigation tree it is obvious that, in practice, the depth of the tree must be limited. For a convenient representation of such trees one may also use several frames, for instance a top frame and a left frame.

## **6.5.2 Presentation Structure Model**

The presentation structure model is a static description of *where* the navigation objects and access primitives will be presented to the user. Thus, the focus of the step is to specify whether a single or multiple-window technique is used, how many frames framesets are divided into (if a frame style is chosen) and in which frames or windows the content is displayed.

### **Modeling Elements**

The following modeling elements are used to describe the presentation structure of a hypermedia application. Windows, framesets and frames are used to describe location of presentation while presentation classes (defined in the previous section)

are used to describe the content of nodes. Hennicker and Koch (2000b) presented the stereotyped presentation and frameset classes, but with a different semantic.

- *window*

A window is the area of the user interface where presentation objects are displayed. A window can be moved, maximised/minimised, resized, reduced to an icon and/or closed. It therefore includes at least five buttons, one to transform the window into an icon, one to close the window, one to maximise/minimise the window, one to resize the window and one to close the window. In addition, windows include horizontal and vertical scrollbars that allow for visualisation of the whole content of the window. Any window is an instance of a stereotyped class «window» that is built in conformity with the structure shown in Figure 6-34. Windows can be organised as a hierarchy.



Figure 6-34: Window Class

- *frameset*

A frameset is a modeling element used to define different visualisation areas within a window. A frameset is always contained in a window; it is divided into lower level location elements – so called frames – and may also contain an arbitrary number of nested framesets. A frameset is an instance of a frameset class stereotyped by «frameset» with a corresponding icon (see Figure 6-35).

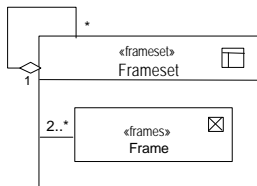


Figure 6-35: Frameset Class and Frame Class

- *frame*

A frameset is divided into a series of frames. A frame is an instance of a frame class stereotyped by «frame» with a corresponding icon. Framesets and frames must be built in accordance with the structure shown in Figure 6-35.

- *presents*

Specifies that the target object of the association is displayed in the location indicated by the source object.

A presentation model of a hypermedia applications is built with stereotyped classes window, frameset, frame and presentation classes. It also includes associations of type «presents» in accordance with the pattern shown in Figure 6-36. This is a design pattern for the case that only one frameset is defined for a window.

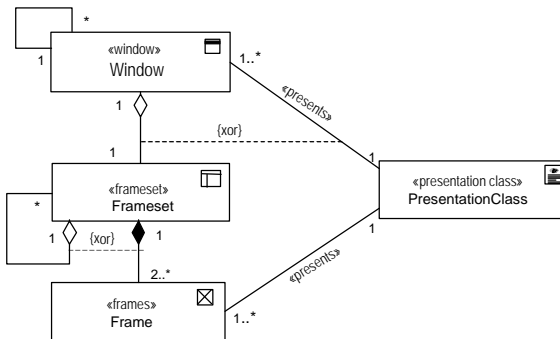


Figure 6-36: Pattern for Main Presentation Modeling Elements

---

### Example

Figure 6-37 shows a partial view of the presentation structure model of the *Online Library* application. This is the view following the link corresponding to the menu item *searchAuthors* of the main menu (see Figure 6-21). Similar views may be constructed for the other menu items of the LibraryMenu, i.e. *publications*, *searchArticles*, *searchVisitedArticles* and *news*. Note that for the LibraryMenu a subclass is included that extends the menu to include the AuthorMenu. This model is obtained from the navigation structure model following the guidelines described in the next subsection.

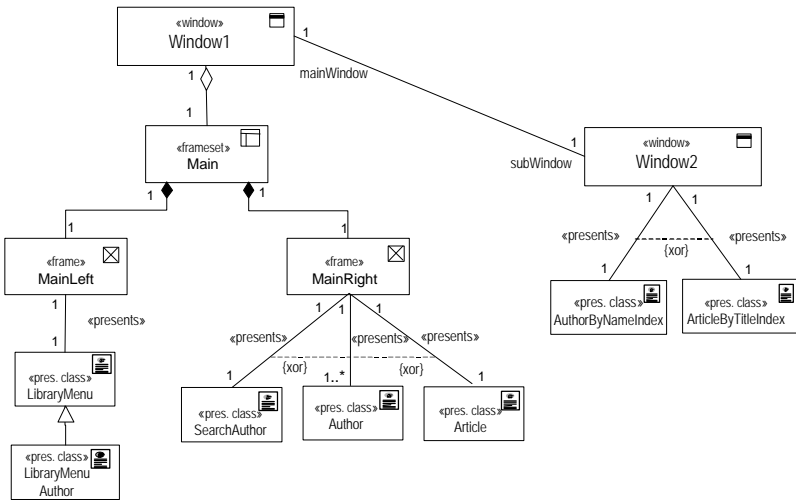


Figure 6-37: Presentation Structure Model of the Online Library Application (Partial View)

Figure 6-38 and Figure 6-39 show two different representations of a frameset, their frame and the associated presentation classes, using the UML composition modeling element. It is an example of a complete Web page (presentation node). This is a frameset which has two frames. The left frame presents the presentation class of the main menu of the *Online Library* application and the right frame

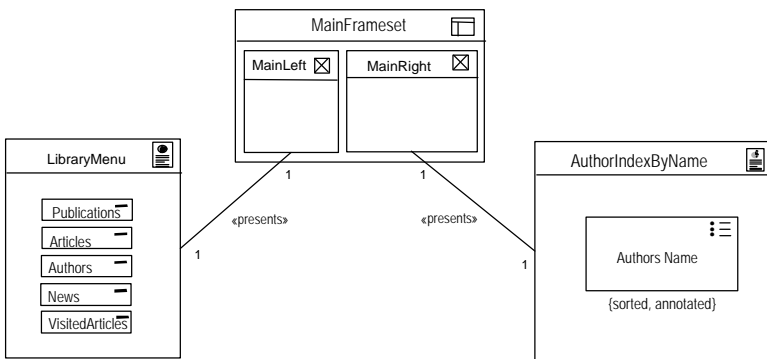


Figure 6-38: Page Presentation

presents the selected content. This kind of representation is optional, it gives an idea of how pages of the application will look, i.e. a sketch of the user interface. Note that the LibraryMenu includes an additional item that allows navigation back to the starting point of the application. Figure 6-39 presents also the constraint which expresses that the menu items News and VisitedArticles not are shown for anonymous users.

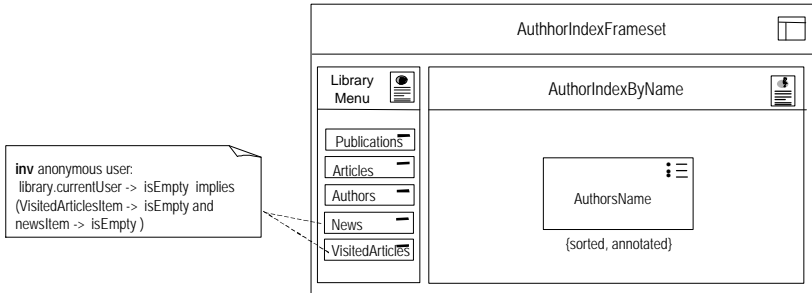


Figure 6-39: Alternative Representation for Page

## The Method

The presentation structure model requires that the designer take some decisions, such as number of windows to be used and number of frames each frameset is divided into. Hence the construction of the presentation structure model cannot be fully automated, but there are certain guidelines that the designer can follow:

1. Select between a single or multiple-window technique. In case of a multiple-window technique plan how many windows will be used.
2. Choose the frame style, i.e. with or without frames. In the first case specify how many frames each frameset has.
3. Use the presentation classes constructed for each access primitive and for each navigation class during the sketching process. Composite presentation classes may be modeled, if their content is presented in one frame or window, such as the LibraryMenuAuthor in Figure 6-29.
4. Decide in which frame of a frameset or a window (no frames style) each presentation class is to be presented to the user.
5. Use a «presents» association for the relationships created in step 4 between window or frame, and presentation class.

6. Depict class diagrams (optional) using composition to show how Web pages will look like. Such a diagram is shown in Figure 6-39.

If many windows and/or frames are used, it is advisable to then construct partial views of the presentation structure model to avoid an overloaded presentation structure model.

Note that different types of presentations can be modeled using this method depending on how composites of presentation classes are defined and how window technique and/or frame style are used. For example a *map-based presentation* consists of a collection of presentation classes with the peculiarity that some of the presentation classes are based on a composition of menus of the navigation structure model. They show a navigation tree (total or partial) that is permanently visible. This allows for navigation at the same level (instances of the same navigation class or context), or at any higher level of the navigation map, i.e. the navigation map is displayed with differing depth depending on the frameset.

### **6.5.3 Presentation Flow Model**

The presentation flow model describes in which window, frameset or frame (visual location element) presentation objects are presented and how the control flows from one location element to another. It is a combined representation of navigation and presentation aspects of the hypermedia application.

At a particular moment a presentation object is active if it is included in the window or frame that is active. A window is active if the mouse focuses on that window (or frame). Only one window or frame can be active at any moment. The presentation object is perceptible if it is included in any window at that point of time. Perceptible means audible in the case of audio and visible in case of all other user interface objects.

If one-window technique is used, only the frameset included in that window is active and the presentation object(s) included in that frameset (or frames of the frameset) are perceptible (visible). The transition to another presentation object thus implies removing the current one and displaying the new one.

If a multiple-window technique is used the following situations are possible:

- A presentation object is displayed in the active window replacing the current one, i.e. the same window remains the active one.

- A presentation object is displayed in a new window opened for this purpose, the content of other windows remain unchanged, but the new window is now the active one.
- A window is closed without altering the content of the other windows. One of these windows has to be assigned as active.
- If dependencies between windows are established, then to close one window may imply closing other windows. Dependencies are usually based on the window hierarchy.
- Mouse focus can change from one window to another, i.e. another window becomes active and therefore another presentation object is active.

Control flow between windows or frames can be represented with interaction diagrams (UML sequence or collaboration diagram) showing which windows are open, which active and which presentation objects are displayed in each window at a certain moment.

---

### **Modeling Elements**

The modeling elements used in the presentation flow model are the user (actor) and objects of the modeling elements defined in the previous subsection, such as windows, framesets and frames.

---

### **Example**

Figure 6-40 shows a presentation flow model for the “search author” scenario in the sample application *Online Library*. It consists of the representation of the message flow between user, windows and frame objects. The presentation flow model shows an abstract representation since the implementation will include more objects, in particular some control objects that collaborate in this interactive process (Conallen, 1999).

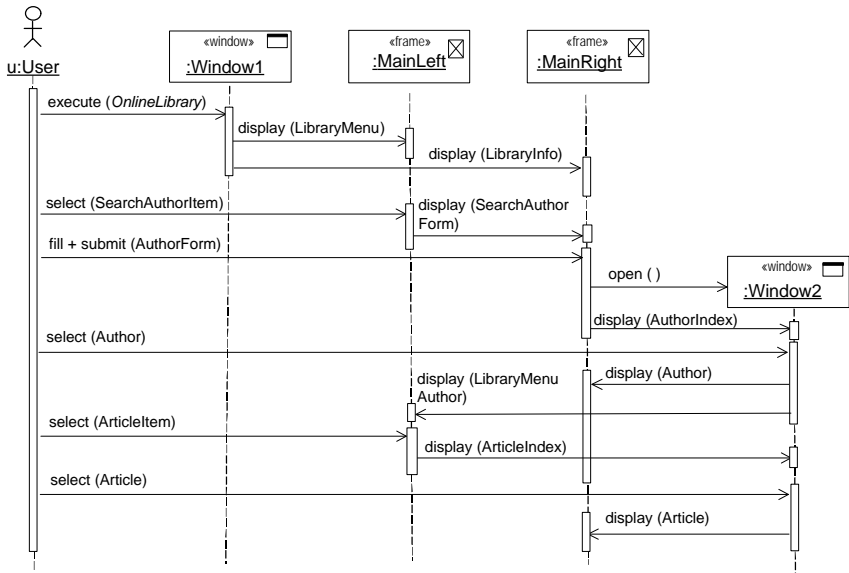


Figure 6-40: Presentation Flow Model of the “Search Author” Scenario of the Online Library Application

## Method

The following is a guideline to assist the developer in the modeling of the presentation flow model, based on the navigation and presentation structure models.

1. Set the scenario for the interaction model, i.e. define which navigation path of the navigation structure diagram will be modeled. A navigation path is always related to a use case.
2. Represent the user, the windows and/or frame objects in the horizontal dimension.
3. Specify a *display* message for each presentation object that should be presented to the user (in a window or frame). The parameter of the display message is the corresponding presentation object (described in Subsection 6.5.1).



4. Include a *select* message for each user action, which selects an anchor or a button. The anchor or button names are the parameters of the message.
5. Specify a *fill* and a *submit* message for each user action, which consist of supplying data in a query form. This form is the parameter of the message.
6. Include a message for each *open* and each *close* of a window.
7. Use “balking” to specify the period of time that a window or frame is active.

UML sequence diagrams are used to represent the presentation control flow (see Figure 6-40). Note that the representation does not include additional classes needed in the implementation to allow the communication.

## **6.5.4 Object Lifecycle Model**

The objective is to model the lifecycle of reactive presentation objects and the influence they have on the status of other presentation objects. The lifecycle of an object is defined by a set of states and transitions between states. A state is specified by a name, entry and exit actions, internal transitions, and/or sub-states. A transition is triggered by an event and may have an action associated with it. In the case of user interfaces most of the events are generated by the user, such as mouse focus, mouse clicks, or keyboard inputs. Complex behaviours can be modeled easily in UML with sub-states (Sauer & Engels, 1999). Two different types of sub-states are possible: sequential and concurrent (Booch, Rumbaugh & Jacobson, 1999).

UML state diagrams have been chosen to depict the object lifecycle model. The design of these UML state diagrams is time consuming and usually so many details are not necessary for presentation classes with well known behaviour. They are therefore used only for complex composite presentation classes.

---

### **Modeling Elements**

The modeling elements used for the object lifecycle model are states and state transitions as defined in the UML to build state diagrams.

---

### **Example**

In the *Online Library* the presentation of an article consists of the title, author's list, abstract of the article, list of keywords and a button that makes it possible to access the complete article.

The design pattern “information on demand” (Rossi, Schwabe & Garrido, 1996) is applied to display the complete article in a separate window. The type of file is chosen in conformity with the preferences of the current user model. Figure 6-41 shows a UML state diagram of the classes involved in the presentation of the complete article, i.e. a button to request the complete article, a window and the content.

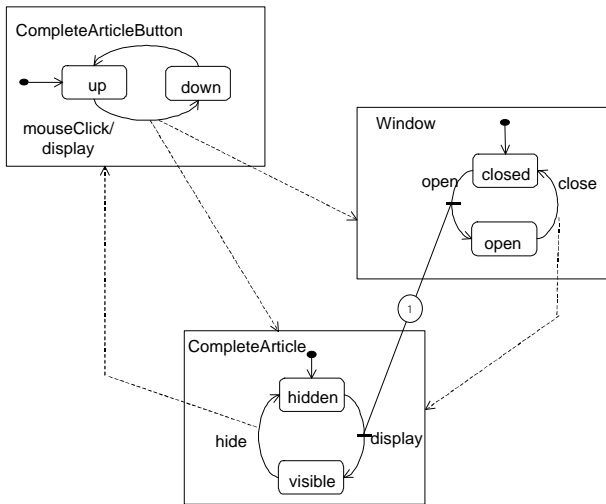


Figure 6-41: Object Lifecycle Model

---

## Method

The object lifecycle model is constructed as a set of state diagrams. The following rules can be used as a guide:

1. Identify the different states of an instance of a class during the presentation (object life cycle). Represent them as UML states. For example, states hidden and visible of the instances of the CompleteArticle class.

2. Specify transitions between states determining the events that trigger these state transitions. For the class `CompleteArticleButton`, the event that triggers the display transition is the mouse click on the button.
3. Define at least one initial state and, possible, one or more final states for each state diagram.
4. Establish if any dependencies exist between state transitions of different objects. Dependencies are represented using UML notation, i.e. dashed lines as shown in Figure 6-41.
5. Determine synchronisation of transitions. In the sample application synchronisation between the open transition of the `Window` and the display transition of the `CompleteArticle` is needed, as the article cannot be displayed before the window is opened.

## 6.6

## Adaptation Model

Finally, the adaptation rules are defined in the adaptation model specifying the conditions under which the content, navigation and presentation are adapted, which actions are performed for the adaptation and how the user model is updated according to the observations of the user behaviour (De Bra & Calvi, 1998). The representation of the rules in a model shows how they collaborate with user behaviour, navigation, presentation and user model elements. UML interaction diagrams (sequence and collaboration) are appropriate for the dynamic representation of the message flow between the adaptation model and the other models as they allow for depicting anonymous and named objects, i.e. showing how global and how local rules collaborate with all or specific objects of a domain class.

A detailed description of the adaptive functionality is given in Chapter 2 and in the Munich Reference Model for adaptive hypermedia systems presented in Chapter 4.

---

### Modeling Elements

The following modeling elements are used to describe the adaptive functionality of an adaptive hypermedia system:

- *rule*

A rule specifies how concepts are found, pages are built and/or presented to the user as well as how the user model is updated. Rules can thus be classified as follows:

- *construction rules* (finding the appropriate concepts)

- *acquisition rules* (acquiring information about the user to update the user model)
- *adaptation rules* (adapting content, link, and/or presentation)

These three types of rule are included in the Munich Reference Model described in Chapter 4. Acquisition techniques are presented in Chapter 3 and adaptation techniques are detailed in Chapter 2. In many adaptive hypermedia systems there is no need for construction rules, as for each concept a page is presented to the user.

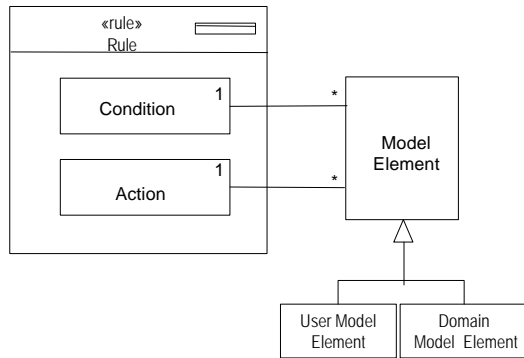


Figure 6-42: Rule Class

A rule is modeled by a condition, an action and an executor function. A rule has at least two attributes. These indicate whether the rule triggers other rules and whether the rule has to be applied before or after presentation. A rule is stereotyped by «rule» with a corresponding icon and follows the composite principle depicted in Figure 6-42.

- *user behaviour*

User behaviour is a class that models the behaviour of the user as observed by the system. Three types are observable in hypermedia applications:



Figure 6-43: User Behaviour Class

- *browsing*, i.e. the visit to a hypermedia node as a result of following a link,
- *input* of data in the input fields of a form together with the sending of the form, and
- *user inactivity* registered by a time-out mechanism.

User behaviour is stereotyped by «user behaviour» with a corresponding icon shown in Figure 6-43

Figure 6-44 shows the analysis pattern, after which an adaptation model is built using rules and user behaviours. This analysis pattern is based on the adaptation model of the reference model presented in Chapter 4.

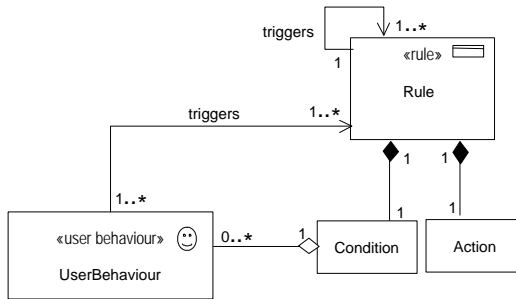


Figure 6-44: Pattern for Adaptation

### Example

The following rules are part of the *Online Library* application. A textual description of each rule is given here. In a further iteration a formal language can be used for the description of these rules.

- The system observes the following user behaviour:
  - visiting articles (browsing),
  - marking articles (input), and
  - inactivity, i.e. no browsing activity during  $n$  minutes ( $n$  can be predetermined).
- Rule 1: If the user visits an article, this visit is registered in the instance of the associated class Visited. The attribute count indicating how many

times the article is visited, is incremented by one and the current date of the visit is recorded.

- Rule 2: If an article is marked, the instance of the associated class `Marked` is set to true.
- Rule 3: If an article is marked, i.e. indicating that the user is interested in that article, then the keywords of the article are registered in the keyword list of the user. If the keywords are already included in the list, the `lastObservationDate` is changed and the `observationTimes` is incremented by one (see Section 6.3). If the keyword is not part of the set, then a new instance of the class `UserKeyword` is created and initialised.
- Rule 4: If the user remains inactive during a period of time, the system presents the `NewsFrameset` to the user, adapted to the current values of the user model.
- Rule 5: For the presentation of the complete article, the type of the file is chosen, in conformity with the `FileType` selected by the user.
- Rule 6: In the `LibraryMenu` the link to news is removed, if the user has chosen to receive the list of news via e-mail.
- Rule 7: An article included in any `ArticleIndex` or `ArticleGuidedTour` is removed if the negative keyword list includes two or more keywords of the article.
- Rule 8: The articles in the `ArticleIndexByTitle`, `VisitedArticleIndexByTitle`, `NewArticleByTitle` and `ArticlesGuidedTour` are sorted based on the current values of positive keywords in the user model.
- Rule 9: The authors in the `AuthorIndexByName` and the publications in the `PublicationIndexbyTitle` are also sorted based on the positive keywords.
- Rule 10: Annotation is performed in the article indexes as follows:
  - red bullets for articles not visited and not marked,
  - white bullets for visited but not marked,
  - blue bullets for visited and marked.

The first three rules are acquisition rules, that update the user model; the other seven rules are adaptation rules.

Note that the initialisation and modification of the user model, in particular the setting of the preferences, is not modeled in the sample application.

Figure 6-45 shows part of an adaptation model for the sample application *Online Library*. It shows the adaptation process that begins when a SearchArticleByTitle form is filled in by the user. The list of articles is provided by the ArticleByTitle context, which is adapted through elimination of links and through addition of links given by positive keywords. The model is represented as a collaboration diagram. The graphical visualisation permits the recognition of loops in the flow of rules triggered by other rules.

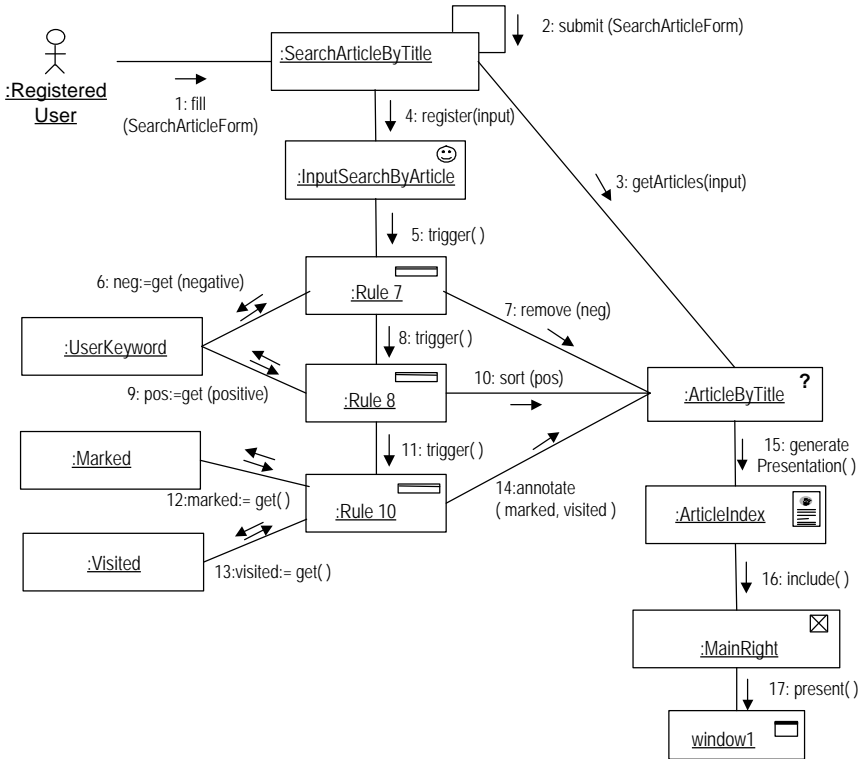


Figure 6-45: Part of the Adaptation Model of the Online Library Application

**Method**

The adaptation model consists of a set of rules described textually or with a formal language, e.g. OCL, and a set of UML collaboration diagrams. Although there is

obviously no way to automate the construction of the adaptation model, there are some guidelines that the developer can follow:

1. Define the user behaviour the system has to observe, such as browsing, input and inactivity.
2. Define a rule for each variant of a navigation class, specifying under which conditions the content variant has to be presented to the user.
3. Define at least one rule for each class of the navigation structure model that is adorned with one or more properties. The condition of the rule establishes under which circumstances the rule has to be applied. The action of the rule specifies a selection in accordance with the current values of the user model or an algorithm based on these values to be applied to obtain a certain order of items for example.
4. Define rules to adapt presentation, e.g. in accordance with the preferences chosen by the user.
5. Represent the more complex adaptation situations as collaboration diagrams. They show how rules, user behaviour, user attributes and presentation objects collaborate to build a presentation.
6. Verify that each user behaviour that is observed is used in the condition of one rule at least.
7. Verify that each attribute of the user model is updated by at least one rule.
8. Verify that the recursive application of rules has a guaranteed termination in all possible cases.



*"If you wait for a complete and perfect concept  
to germinate in your mind,  
you are likely to wait forever"*  
Tom De Marco,  
*Structured Analysis and System Specification,*  
Prentice Hall, 1979.

## **7 The Software Development Process**

Most of the existing adaptive hypermedia applications are built as prototypes. Their implementation is usually performed ad hoc and is improved in successive steps. Adaptive hypermedia systems are complex software systems and they therefore require, an appropriate software engineering process. As far as we know, there is currently no systematic engineering process which describes how adaptive hypermedia systems should be developed.

Even the production of non-adaptive hypermedia systems demands, as far as certain aspects are concerned, a development process that differs significantly from developing other software applications (Olsina, 1998; Lowe & Hall, 1999; Conallen, 1999). Adaptive hypermedia and Web applications are characterised by a rich multimedia material, conversion of content to hypertext format, a linked structure, and a greater emphasis on graphic design. Other minor differences as compared to classical software engineering can be found in the activities of project management and quality management. The Interactive Media Process Assessment, Characterisation and Tracking schema (IMPACT) of Lowe and Webby (1998) for example, presents a metamodel of the information layer and specifies activities, artifacts and workers for each separate aspect of the Web information systems, i.e. content, structure, presentation and functionality.

The development of adaptive hypermedia systems requires additional effort. Different users require different content, different forms of presentation and different guidance through the material. Adaptability is based on user modeling and adaptive features, i.e. on the building, maintaining and exploiting of a user model and an adaptive engine. Paternó and Mancini (1999) propose a method for

developing configurable hypermedia systems on the basis of task modeling and the usage of a set of heuristics. It does not support user modeling, as only three static user profiles are considered for adaptation.

It is difficult to decide how formal or how formless software development process should be. Very formal methods have the advantage to allow correctness proofs, but they add many formalisms that tend to abort creativity. Formless development is chaotic and seldom conducts to a successful project. Booch (1994) mentions five different schools that can be followed during the development of a software system:

- *Anarchists* ignore all kind of methodical procedure; they rely only on their intuition and creativity.
- *Behaviourists* concentrate on roles and responsibilities.
- *Storyboarder* see the world as a set of business processes.
- *Information modeler* observe primarily the data; for them the behaviour stays in second place.
- *Architects* focus on frameworks and patterns.

For Sommerville (1982) the best known process models are the waterfall model, the spiral model and the explorative process model. During the last years object-oriented development replaced functional and structured oriented approaches. New process models that focus on object-oriented development have been introduced, such as the Objectory Software Development Process (Jacobson, 1992), the OPEN Process (Graham, Henderson-Sellers & Younessi, 1997) and the Unified Software Development Process (Jacobson, Booch & Rumbaugh, 1999). They also allow for an iterative approach. These process models are only applicable to the development of adaptive hypermedia applications with certain restrictions.

The UML-based Web Engineering approach (UWE), presented here, covers all aspects that the development of adaptive hypermedia applications require. In this chapter the development process of UWE is described, which chooses a mixture of the different directions proposed by Booch (1994). UWE is based on the Unified Software Development Process – also known as Unified Process – (Jacobson, Booch & Rumbaugh, 1999) and the Rational Unified Process – RUP (2000).

The main differences of UWE with respect to the Unified Process (UP) and the RUP are:

- It specialises the UP for the development of adaptive hypermedia applications describing which “experts” (workers) are required, which activities they perform and which specific artifacts they produce.

- It extends the coverage of the UP development cycle including a maintenance phase.
- It adds to the UP development process supporting workflows for project management (included in RUP) and quality management.
- It changes the idea of quality control management, defined in the UP and RUP only through testing, incorporating workflows for requirements validation and design verification.
- It proposes a stereotype-based extension of UML (UML Profile) for adaptive hypermedia applications.
- It includes a systematic method for the analysis of adaptive hypermedia applications based on the separation of user modeling, conceptual, navigation, presentation and adaptation aspects (already presented in Chapter 6).

In addition, UWE, same as the RUP, is an object-oriented approach that covers the entire life cycle of adaptive hypermedia systems, it moves through a series of iterations and increments and uses UML notation and diagrams. UML activity diagrams are used to visualise the activities of the workflows and the artifacts produced by the workers.

Summarising, the UWE is a systematic, prescriptive, user-centric, UML-based, iterative and incremental methodology for adaptive hypermedia systems.

This chapter is structured as follows: Section 1 outlines the most important aspects of the development process. Section 2 presents the phases and milestones of the process. Section 3, 4 and 5 describe the workflows of the development process, project management and quality management, respectively.

## **7.1 Adaptive Hypermedia Systems**

The aim of this chapter is to describe UWE, a systematic hypermedia engineering approach for adaptive hypermedia systems. The methodology and the techniques described here, however can also be used as well for non-adaptive hypermedia applications. Web applications are a subset of hypermedia applications; UWE is therefore also applicable to Web applications.

The term *adaptive hypermedia development* is used in this work, to refer to the development and the maintenance of adaptive hypermedia applications, in the same way as Lowe and Hall (1999) define it for hypermedia applications. UWE covers the life cycle of this type of applications from the creation to cessation.

*Adaptive hypermedia engineering* refers to the application of an engineering approach to the development process. The development process is supported by project management and quality management activities.

*Adaptive hypermedia design* covers only a part of the life cycle, i.e. the application of a method to generate a schema for the structure and functionality of the domain as well as to implement a user model. It does not cover other activities such as iteration planning, requirements capture, implementation and testing.

*Adaptive hypermedia authoring* is an even more reduced set of activities in the life cycle of adaptive hypermedia. It is limited to the creation and structuring of content, usually supported by special tools.

### **7.1.1 Covering the Life Cycle**

UWE covers – as already said – the whole *life cycle* of adaptive hypermedia applications, starting when the idea for an adaptive hypermedia system is conceived and ending when the product is no longer available for use. A clear distinction should be made between UWE and the *software development cycle* that begins with the decision to build an adaptive hypermedia application and ends when the adaptive software product is delivered. A software development cycle that comprise analysis, design, implementation and usage is analysed by Scharl (1999).

UWE describes an iterative and incremental process. The iterative approach reduces the risks of the waterfall model, in which the development consists of a single sequence of phases with little feedback from each phase to the previous ones. Development based on the waterfall model produces results very late in the process, making it difficult to introduce changes to the initial decisions. UWE supports a more incremental development process than the spiral model of Boehm (1988). In the spiral model there are four distinct cycles of development: concept, requirements, design and implementation. During each cycle the same four activities are performed. These activities are: determination of objectives and constraints, evaluation of alternatives and resolution of possible risks, development and verification, and planning for the next cycle. The spiral model does not address maintenance activities.

UWE is based on the UP, which allows for an incremental development process through the inception, elaboration, construction and transition phases. In each phase a little of requirements capture, a little of planning, a little of design, implementation and little of testing is done. It uses UP terminology, whenever

possible. The workflows of the UP have been adapted or extended. The methodology specifies the activities to be performed at each phase as well as the resulting artifacts and the workers responsible for these activities.

UWE establishes a priori milestones between the phases. The goal of an iterative process with milestones is to allow major control during the whole process, thus mitigating the risks inherent in the development process. Steps for planning, designing, implementing, integrating and testing are performed at each iteration. In between steps, the project manager can get feedback and adjust the goals of the next step.

UWE is specially tailored for the development of adaptive hypermedia applications as user modeling, adaptation specification and user behaviour capture are included as separate design steps. If one skips the special steps for user modeling and adaptive aspects, this methodology can also be used for the development of hypermedia systems.

UWE is an object-oriented approach, which uses UML techniques for analysis and design of the adaptive hypermedia applications to be developed. The UML notation used corresponds to the UML Profile presented in Chapter 6. This UML Profile is defined according to the extension mechanisms supported by UML, i.e. by the definition of stereotypes and the specification of OCL constraints.

## 7.1.2 Iterative Development

UWE is an iterative process that a priori does not establish how many iterations are to be performed. At each iteration a set of process workflows are performed; this set is called the *iteration workflow*. UWE consists of *development process workflows* and two supporting workflows: *project management* and *quality management*.

A UML activity diagram is used in this work to represent the iteration workflow. This activity diagram is shown in Figure 7-1. It depicts the flow of control (continued lines) and dependencies (dashed lines) between workflows. Swimlanes are used to indicate the different workflow groups: project management, development process and quality management.

The core workflows belonging to the development process are: *requirements capture*, *analysis and design*, and *implementation*. The supporting workflows of the project management are: *risk management*, *iteration planning* and *iteration evaluation*. Quality management comprises *validation* of the requirements,

*verification* of the artifacts resulting from the analysis and design results, and *testing* of the implemented system.

Further differences between UWE and the UP or between UWE and the RUP should be noted here: First, UWE includes development process workflows and supporting workflows in the same way the RUP does. Second, whereas the RUP defines the following management supporting workflows: configuration and change management, project management and environment, in UWE six supporting workflows are included. They are grouped into project management and quality management. Third, as in the RUP, analysis and design are treated in the same workflow. Finally, the more general concept quality management is used instead of “testing”.

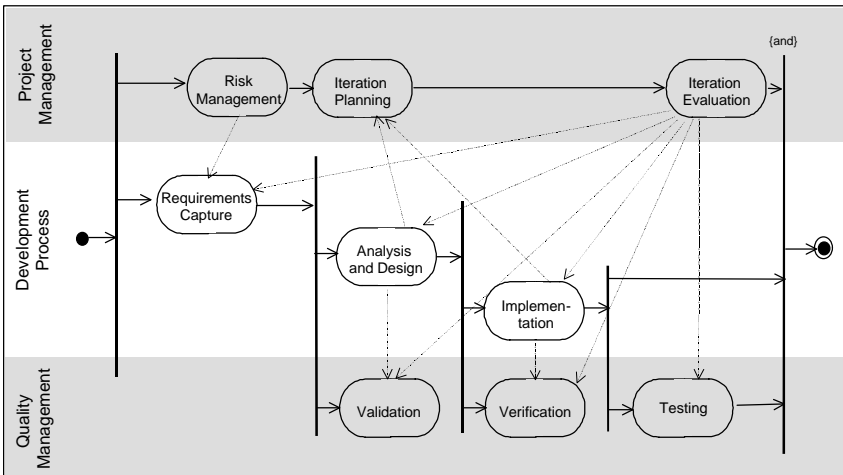


Figure 7-1: Iteration Workflow

The life of an application is divided by UWE into cycles, each cycle concludes with a release. The first cycle begins with the conception of the idea and the last one ends with the cessation of the use of the application. UWE does not establish the number of iterations required in a life cycle of an adaptive hypermedia application, but distinguishes a number of phases. Each iteration belongs then to one of following phases: inception, elaboration, construction, transition or maintenance. These phases take place over a period of time and each phase terminates in a milestone. A milestone is defined by the delivery of artifacts, such as models, code

or documents and by the decisions that workers will take at this phase in the process before the work of the next phase can proceed.

The number of iterations planned for each phase varies, essentially, with the complexity of the project. If the project is simple, one iteration per phase will be enough. The number of iterations of the elaboration and construction phases usually increases with the complexity of a project.

Before a detailed description of the phases of UWE as well as of the processes and workflows are given, the terms artifact, activity, worker, stakeholder and model are defined.

- An *artifact* is a tangible piece of information that is created, changed and used by workers when performing activities. It can be a model, a model element or a document.
- An *activity* is a tangible unit of work performed by a worker in a workflow. The activity implies a clearly-defined responsibility of the worker, a clearly-defined result (artifacts) and a well-defined input.
- A *worker* is a person with certain capabilities to perform one or more activities during a process. The term “worker” is not synonymous with person. One person can play the role of one or more workers during a project, even simultaneously.
- A *stakeholder* is any person interested in the outcome of the project, such as a user, a designer, an engineer, a customer, a contractor or a project manager.
- A *model* is a simplification of reality, i.e. a semantically close abstraction of a system, the objective being to better understand the system being created.

Each workflow is defined by a set of activities that are performed by a set of workers with the goal to produce some artifacts, which are the measurable results of the workflow. The artifacts, workers and activities of each workflow are briefly described in subsections 3 to 5 of each section. The objective is to give only a general overview of the Unified Process and to detail the aspects specific to the hypermedia and adaptive systems. Examples are shown in the activities subsections.

The software development process described by UWE consists of five phases, each of which can be performed in one or more iterations. In an orthogonal dimension to these phases a set of workflows are defined (see above Subsection 7.1.2). In each phase activities of almost all those workflows are performed.

The first four phases are the same as those in the Unified Process: *inception*, *elaboration*, *construction* and *transition*. The process is extended to include the *maintenance* phase as maintenance activities are an important part of the life cycle of hypermedia applications and often begin immediately after transition has been completed. Maintenance implies not only changes in the content, but also in the layout and changes in the hypermedia structure. These latter changes constitute an important difference vis a vis traditional software. The activity diagram shown in Figure 7-2 presents the iterative software engineering process including these five phases for each release. In each phase of the process some activities of the iteration workflow are performed, i.e. activities of the workflows requirement capture, risk management, iteration planning, analysis and design, validation, etc.

The *inception phase* starts with no more than an idea for a new system or the need of an extension of an existing system. The final state of the inception phase is a vision of the end system and its business case. The objectives of the systems development are defined during this phase as well as a first approach to the architecture of the system, an estimation of costs and a schedule plan.

During the *elaboration phase* the architecture of the system and a set of design models are defined. The project manager will elaborate a plan of activities and an estimation of the resources needed to complete the project. A stable architecture as well as control over the risks are prerequisites for the next phase.

The *construction phase* focuses on the development of the system, although additional requirements elicitation and minor changes in the architecture design are performed during this step. The phase is finalised when all the uses cases are implemented.

The *transition phase* covers the period during which the system is tested as a complete version (normally called beta release) by a reduced group of users. Training, help-assistance and correction of defects are the main activities of the transition phase.

The *maintenance phase* begins when the first version is delivered and extends itself until the system is no longer used. During this period of time the system requires different types of adjustments: content updates, layout improvement, structure modifications and adaptation to new technologies or new software versions.



Five main milestones are part of the process marking the end of each of these phases. These milestones are: life cycle objectives, life cycle architecture, initial operational capability, product release and product cessation. The following

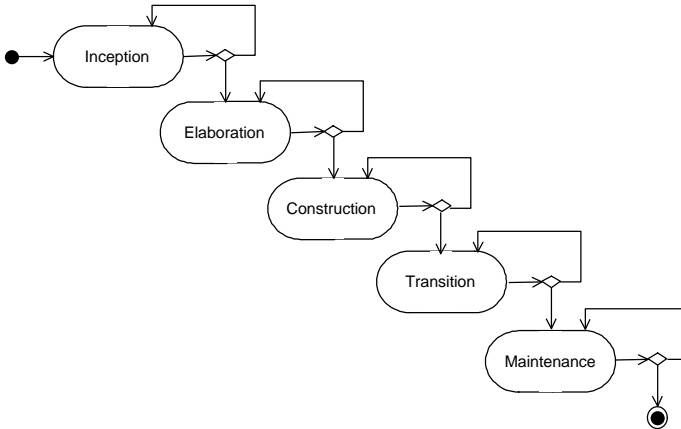


Figure 7-2: UML-based Web Engineering Process for one Release

sections outline the characteristics of the iteration workflow in each of these phases.

## 7.2.1 Inception

The principal objective of the inception phase is to establish the feasibility of the project, i.e. to define the business case for the system and delimit the scope of the project. This business case includes success criteria, risks assessment, budget and resource estimation as well as a phase plan with a schedule and delivery plan for the major milestones. Sometimes an executable prototype is developed during this phase.

At the beginning of the inception phase there is only an idea. The process starts with the conception. The goal is to develop and evaluate the idea for an adaptive hypermedia system, i.e. the need for an adaptive capability and benefits in developing an application based on it. The main aim of the feasibility study is to define the high level functional requirements for the adaptive hypermedia system, to outline a first budget, produce a draft schedule plan and establish the main non-functional requirements of the system. The costs of the project must be estimated during this phase, as this is a crucial element in determining the feasibility of the

project. It may be advisable to develop a prototype, but one has to consider the need to minimise effort just in case the idea fails to live up to expectations. Depending on the results of this study a decision can be taken as to whether it is worthwhile to develop an adaptive hypermedia application which dynamically adapts to the current state of the user model.

Techniques which support feasibility study vary from informal *textual description*, through *checklists* and *spread sheets* to implementation-oriented techniques such as paper or computer based *storyboarding* as proposed by Boyle (1997). *Prototyping* is a meaningful but expensive alternative.

The inception phase focuses on requirements capture, risk management, project planning and validation of the requirements. Activities, such as design, implementation and testing are of minor importance during inception. Decisions are taken about Web specific techniques, such mouse-overs, animations, multimedia and about the browsers under which the application will run.

The most important factors that influence the activities of the workflows in the inception phase are:

- *current information, information sources and information structure*

In the development of adaptive hypermedia applications, it is important to have an understanding of the types of information available and of the relationships between the information as it will be reflected in the hypermedia structure. Other considerations include: how often the information changes as well as security and legal aspects related to information sources and to user models.

- *current applications*

In many cases adaptive hypermedia applications are developed to replace or enhance existing non-adaptive hypermedia-based or non web-based applications. An understanding of these applications, and any problems involved in them is helpful in understanding the scope of the new application to be developed. The navigation of the final application has to support the same or an improved behaviour and functionality of the legacy system.

- *stakeholders*

Any existing system has numerous categories of people who have a stake in the system, such as clients, users, designers and providers. Adaptive hypermedia systems involve a greater variety of stakeholders than other software development projects. They are a heterogeneous group of people including a complex and heterogeneous group of users, multimedia experts,

graphic designers, hypermedia architects, public relation experts, marketing people, legal experts, e-commerce experts, etc.

- *resources*

This factor includes personnel, budget, time, information, expertise, hardware and tools.

- *technological limitations*

Technological limitations are constraints that condition possible and appropriate solutions. Limitations include processing power, bandwidths, equipment costs, the equipment's reliability and security. The development of adaptive hypermedia applications requires detailed analysis and testing of new and existing technologies, such as JSP's, ASP's, java applets and servlets.

- *constraints*

Constraints may be inherent to the domain or imposed by the client. They are mainly related to the architecture that may be predefined or are re-used from other hypermedia applications projects. Examples of Web applications frameworks are WebSphere<sup>9</sup> and WinDNA<sup>10</sup>. Adaptive hypermedia systems require a careful evaluation of the need and benefits of user modeling and adaptation features to be included in the system. Different alternatives for the construction and exploitation of the user model can be evaluated during this phase.

At the end of the inception phase the decision is taken whether to proceed with the development or not. The first milestone, i.e. *life cycle objectives*, marks the end of the inception phase. The deliverables for the inception phase are:

- a first version of a problem domain model,
- a first version of the use case model,
- a first draft of the architecture description,
- a prototype to prove the concepts or a new technology (optional),
- a risk study,
- a plan for the whole project,
- a business case, including success criteria, risk analysis and budget estimation, and
- an architecture validation and a requirements review report.

---

<sup>9</sup> IBM's WebSphere

<sup>10</sup> Microsoft's WinDNA

## 7.2.2

## Elaboration

The principal objective of the elaboration phase is to capture the remaining requirements, to establish a sound architectural baseline, to elaborate a guide for construction based on models, to identify additional risks, review already known risks and to detail the project plan.

At the beginning of the elaboration phase, after the first milestone of the process the principal inputs that are available are the draft architecture, draft use cases and problem domain model, a project plan, a list of risks and a business case.

The elaboration phase focuses on analysis and design as well as on iteration planning and verification of the design. There are many factors which influence the activities of workflows in the elaboration phase. These factors can be classified into:

- *design creativity*

The creativity of the design very often requires the expertise of multimedia, graphic or marketing experts. Hypermedia design needs to take into account cultural and perceptual aspects. These will influence the success of an adaptive hypermedia application over and above the functionality it offers to the users. It is a factor that most of the traditional software engineering environments do not support (Nanard & Nanard, 1995).

- *technical issues*

The limitations of current technology impose restrictions related to the hardware, software, databases and authoring tools that can be used. For example bandwidth puts a limit on the information that can be accessed in an appropriate time interval during online access. In the same way the type of video or audio to be used depends on the processing power of the user's computers. In addition, technologies for dynamic page generation or adaptation of content and links have to be compared.

- *cognitive issues*

Cognitive issues play an important role in the development of hypermedia applications. The designer should know how to avoid the problems of cognitive overload and becoming lost in the hyperspace. He has to be aware of techniques to organise and navigate information. Content structuring parallels the traditional educational concern of curriculum development. The structuring of interactivity, in a similar way, corresponds in learning environments to the concern of pedagogy (Boyle, 1997).

- *non-technical issues*

Non-technical issues include aspects, such as laws and security regulations in the different countries that will access the application. Privacy of data stored about a user in the user model of an adaptive hypermedia application is one of the questions to be resolved during this phase.

At the end of the elaboration phase the system is ready to be produced. The second milestone, called *life cycle architecture*, marks the end of the elaboration phase. The deliverables for the elaboration phase are:

- a complete business or problem domain model,
- a new version of all models: use cases, analysis, design, deployment, implementation and testing,
- an architectural baseline and detailed description,
- an updated risk list,
- a project plan for construction and transition,
- a complete business case, and
- an architecture verification and model review reports.

### 7.2.3

### **Construction**

The principal objective of the construction phase is to produce a software product ready for initial operational release, the so called “beta release”.

At the beginning of the construction phase the complete set of use cases have been described, all the models are developed, the main risks analysed in detail, the architecture defined and almost all requirements captured (in practice about 70% to 80%).

The construction phase focuses on implementation as well as on testing of the system. During the construction phase various components required for the application are produced, obtained or modified according to the design specifications. These components are then integrated to create a prototype or the final application.

The most important factors that influence the activities of the different workflows in the construction phase are:

- *media components availability*

The production or adaptation of existing multimedia components contributes to the complexity of a project. It requires the participation of more experts than is the case for traditional software development, such as audio, video and animation designers.

- *dynamic page generation*

If pages of a hypermedia application are generated dynamically, a database has to be administrated, templates need to be defined and a page generator must be implemented or generated. Performance may be a critical risk in this case.

- *user behaviour observation*

Adaptive hypermedia systems require user modeling, i.e. the system stores a set of beliefs about the user and changes these beliefs dynamically according to observations made about the user's behaviour. A mechanism to register the user's behaviour has to be implemented.

- *adaptation engine*

The adaptive engine or functionality is responsible for the adaptation of the content, presentation and navigation to the current state of the user model.

- *usability*

The usability of the application for different platforms is a typical factor that affects construction; it requires additional effort. The objective is to obtain a Web application that has the same quality independently of the browser used.

At the end of the construction phase the first version of the adaptive hypermedia system is ready to be used. The third milestone is the *initial operation capability* and marks the end of the construction phase. The deliverables for the construction phase are:

- the executable software itself – the initial operational capability release (last version built during construction),
- all the artifacts and models developed during the project,
- the architecture description updated to reflect all the changes introduced during the construction,
- a draft version of the user manual to guide beta users, and
- a project plan for the transition and maintenance phases.

## 7.2.4

## Transition

---

The principal objective of the transition phase is to integrate the product in the user's environment and correct the operational version until customers provide positive acceptance tests. Jacobson and Thomas (1995) stress that the use of object-oriented techniques introduce additional complexities in system integration and testing.

At the beginning of the transition phase the system has reached initial operational capability. In addition the results of the testing workflow are available to remove the last bugs and inconsistencies.

The transition phase focuses on the establishment of the final product in the operational environment and on the evaluation of the project. There are different ways to perform this transition depending on the type of product that is developed. If it is a Web application for the Internet for example, a beta version is tested by a group of acceptance testers before going online. This group has to be as heterogeneous as possible and must attempt to simulate real Internet users. If the resulting product is an Intranet application within a large organisation, it could, for example, be tested by one department or section of the organisation.

The most important factors that influence the activities of the different workflows in the transition phase are:

- *insufficient or inadequate tests*

The absence of sufficient or adequate tests in the construction phase, means that the bugs, misunderstandings and errors appear during this phase, thus augmenting the amount of re-working to be done. Adaptive hypermedia systems are even more difficult to test as these systems require a variety of users to test different user's behaviour in order to generate different user models.

- *schedule pressure*

Schedule pressure is often caused by insufficient time planned for the transition activities, such as installations, test, corrections and reworks.

- *budget pressure*

If no budget is left, it is often difficult to make corrections or improvements even if there is enough time to perform them.

- *insufficient time for reworking*

This happens when software products are supposed to be final versions at the end of the construction phase.

- *insufficient knowledge transfer*

This happens if there is not sufficient time planned for documentation.

At the end of the transition phase the system is ready to be used by the customer. The fourth milestone, i.e. *the product release*, marks the end of the transition phase. The deliverables for the transition phase are similar to the deliverables in the construction phase, but are now correct and complete:

- the executable software itself, including the installation software,
- legal documents, such as contracts, licenses, etc.,
- final version of all models and artifacts produced during the project,
- completed and corrected architecture description,
- final version of user manual,
- training material,
- references and addresses where to find additional information.

## 7.2.5

## Maintenance

The principal objective of the maintenance phase is to adapt an adaptive hypermedia application to a changing environment, conditions or new resources. Maintenance plays an essential role, even more for certain types of adaptive hypermedia systems such as Web applications. Changes in the content and layout improvements may require modifications in the structure as well as to the adaptive components. Maintenance is used with the meaning of “extensions”, a concept introduced by Jacobson and Thomas (1995), as distinct from a defect repair or update to the existing system. Maintenance of adaptive hypermedia applications is thus more complicated and more time-consuming than maintenance of traditional software systems, where the maintenance process is restricted to data up-dating.

At the beginning of the maintenance phase a full operating adaptive hypermedia system is provided. During the whole maintenance process the state of this system has to be preserved.

The maintenance phase focuses on the implementation of changes, corrections or improvements.

There are many factors, which influence the activities of the workflows in the maintenance phase. These factors can be classified into:

- *coupling and cohesion of the hypermedia application*



In traditional software the rule is to minimise coupling. In comparison a typical characteristic of hypermedia applications is associative coupling through linking. Highly coupled pages are difficult to maintain and easily lead to dangling or incorrect links. Cohesion means that for hypermedia applications each node, if possible, is based on a single concept. An appropriate chunking of the information may improve maintenance.

- *maintenance of the analysis and design documentation*

The maintenance of the documentation related to analysis and design is known as a critical factor in the maintainability of software.

- *code documentation*

This factor is as important for the maintenance of hypermedia applications than for the maintenance of traditional software.

The fifth milestone, i.e. *products life end*, marks the end of the maintenance phase. At the end of the maintenance phase the system is replaced by another product or is merely taken out of service.

There are no deliverables at the end of the maintenance phase as the product life ends at that moment. At each iteration within the maintenance phase the deliverables are the same as in the transition phase.

## 7.3 Development Process

The development process of UWE consists of the workflows: requirement capture, analysis and design, and implementation. They are called the core workflows. Note how this differs to the UP, which includes five core workflows: requirements capture, analysis, design, implementation and testing.

UWE does not include design as a separate workflow. Design is considered as a refinement process of the analysis. The first iteration of the analysis and design corresponds to a rough analysis, followed by successive refinements until a detailed and implementation-oriented design is reached during the last iteration. Thus, in the same way as in the RUP, the analysis model is defined as an abstraction of the design model, a generalisation or a less detailed design model. Performing analysis and design in two separate workflows increases the documentation efforts as analysis and design models both have to be updated. Testing is the other core workflow of the Unified Process not included here. In UWE it is part of one of the supporting workflows: quality management process. This process includes validation, verification and testing workflows.

*Example: Online Library*

In the remaining part of this chapter the example presented in Chapter 6 is used to illustrate the activities and artifacts of the UWE development process. Starting point is the idea of an online library that gives a personal support to the user. The *Online Library* that is being developed step by step will offer information about publications to registered and anonymous users. The publication information comprises journals, books and proceedings.

### 7.3.1 Requirements Capture

The requirements capture is the process of determining or, some times under more difficult circumstances, the process of discovering what application is to be built. A *requirement* is a condition or capability to which an application must conform.

The requirements capture is not an easy task even for the development of traditional software. There are even more difficulties in the case of adaptive hypermedia applications. Some reasons for a detailed requirements specification are:

- The development of adaptive hypermedia applications may have different starting points, such as a vision, an existing application or a concrete description.
- Stakeholders usually have only partial knowledge of the process to be supported by the adaptive hypermedia application.
- Target groups, technologies, and resources may change during development.
- There is little experience in the systematic development of adaptive hypermedia applications.
- Greater risks are involved than in the development of a non-adaptive application.
- Every project is unique, as it is developed for different organisations, users, goals, technologies, etc.

Traditional approaches are based on the abilities of a system analyst, who elicits a list of requirements from the users. This approach has been improved by the specification of use cases. The analyst uses them to ensure the completeness and correctness of the requirements. Use cases are a good basis for discussions between customers, users and hypermedia analysts. Note that requirements must be described in the *customer language* without, where possible, formal and technical

specifications. Use cases have the advantage of being less unambiguous than textual descriptions and they are understandable to customers and hypermedia analysts.

Two categories of requirements can be distinguished: functional and non-functional.

- *Functional requirements* of adaptive hypermedia systems are the actions the system will be able to perform, i.e. they are used to describe the systems behaviour given certain input conditions. In this work the following types of functional requirements are distinguished:
  - *requirements related to content* (e.g. images to illustrate certain pages, evaluation of exercise results)
  - *requirements related to structure* (e.g. navigation to the homepage is allowed from every page)
  - *requirements related to presentation* (e.g. layout restrictions, such as no more than 10 items in a list)
  - *requirements related to adaptation* (e.g. anchors annotated differently depending on the current state of the user model)
  - *requirements related to the user* (e.g. user preferences or goals to be taken into account)
- *Non-functional requirements* specify systems properties, such as environment and implementation constraints, performance, reliability, extensibility, etc.

Due to the variety and complexity of requirements to be identified and analysed, the requirements capture workflow contains several activities performed by four different workers and the results are a use case model as well as documents describing potential users, adaptation rules, scenarios, content, use cases and the user interface.

The requirements capture workflow is shown in Figure 7-3. A UML activity diagram is used for the schematic representation of the workflow. The visual ordering of the activities should not to be seen as very tight; some overlapping of these activities is also possible. Swimlanes are used to specify the locus of responsibilities for the activities. A worker is responsible for all the use cases comprised between two swimlanes, but more workers can collaborate in the development of these activities. These are usually the other workers of the workflow.

An object flow has been added to the same diagram; it is denoted in dashed lines (Booch, Rumbaugh & Jacobson, 1999 and Oestereich, 1999). Hence, workers, activities and artifacts of a workflow are shown all in the same diagram. A detailed description of the activities, artifacts and workers follows. Artifacts that may be used as input to the requirements capture activities, such as description of a previous version of the system or a business model are included in the diagram.

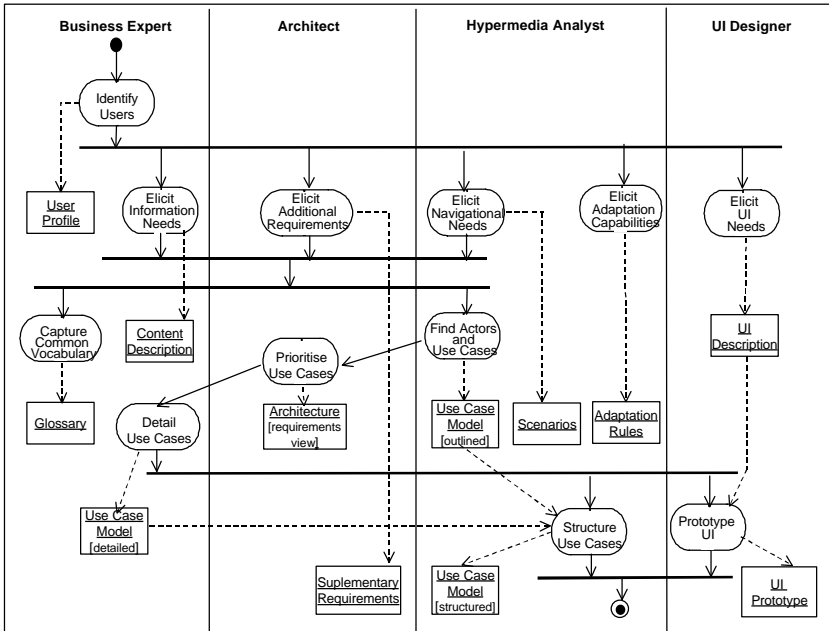


Figure 7-3: Requirements Capture Workflow



## Artifacts

The main artifact produced in the requirements capture workflow is the use case model composed by actors and use cases. The result of the elicitation process is a set of artifacts; there are the user profile, content description, scenarios, adaptation rules, supplementary requirements and user interface prototype. They are used as input for the activities that produce the use case model. In addition, an architecture is elaborated and a glossary has to be prepared from the beginning.

---

### **User Profile**

The user profile is a description of users or user groups. This description includes user goals, tasks, preferences and/or background knowledge related to the problem domain.

---

### **Content Description**

The content description consists of a detailed list of information sources that will be included in the application. Sometimes adaptive hypermedia systems require several variants for the same concept or component.

---

### **Scenarios**

Scenarios are a textual or graphical description of the typical sequences of activities performed by actors of the system. Scenarios for business process concentrate on one usage aspect at a time and express relationships between the system and business process. This type of scenarios help the developers to manage the complexity of the application domain (Weidenhaupt, Pohl, Jarke & Haumer, 1999). In hypermedia systems there are the typical navigation paths users follow through navigation or the steps that authors follow during the authoring process. Scenarios are used for work distribution within and among development teams, and sometimes for the derivation of test cases.

---

### **Adaptation Rules**

The adaptation rules describe the adaptive behaviour of the hypermedia application. They specify how the system dynamically adapts content, presentation and links according to the current state of the user model.

---

### **Use Case Model**

The use case model is a model of the system's intended functions and its environment, and serves as a contract between the customers, business experts, designers and architects. Different versions of the use case model are built during the requirements capture workflow in increasing detail. They are called the outlined, detailed and structured use case models respectively.

---

### **Architecture (View of the Use Case Model)**

The architecture contains an architectural view of the use case model, i.e. focusing on the use cases that are important for the architecture of the system.

---

### **User Interface Description and Prototype**

The UI description may consist, for example, of paper sketches, bitmaps from a drawing tool or an interactive executable prototype.

---

### **Supplementary Requirements**

The supplementary requirements are artifacts in form of documents describing non functional requirements that can not be captured in the use case model.

---

### **Glossary**

The glossary defines the terms used in the project. It is useful to reach a consensus among customers, project managers, designers and engineers regarding the definition of concepts used in the project in order to reduce the risk of misunderstandings.



---

### **Workers**

The requirements captured during the development of an adaptive or non-adaptive hypermedia application is performed by the following four workers: business expert, architect, hypermedia analyst and user interface designer.

---

### **Business Expert**

The business expert is responsible for the business use case modeling, by outlining and delimiting the organisation being modeled. He identifies the potential users groups, their goals and preferences, i.e. he establishes what business actors and business use cases exist and how they interact. In addition, he defines the glossary, working together with the hypermedia analyst.

The business expert does not need knowledge in hypermedia engineering but experience in the use of similar hypermedia applications is helpful.

The profile of the business expert is epitomised by the following skill: knowledge of the business domain.

*Example: Online Library*

The business expert must be a librarian specialised in the categorisation of publications related to the main themes included in the online library.

---

### **Architect**

The architect is the person in charge of the software development process. He must be a domain expert as well as have knowledge of software development. He is responsible for the leadership and co-ordination of all technical activities and the delivery of all technical artifacts throughout the project. He is the technical driving force of the project.

The architect does not have to have project manager responsibilities. He begins his work in the inception phase and will remain active until the transition phase is concluded.

During the requirements capture the architect describes the architectural view of the use case model. He elicits the additional requirements of adaptive hypermedia applications, such as networking, distribution, and browser restrictions. He prioritises the use cases. This prioritisation is an important input in the planning of the iterations. The architect is usually assisted in his work by other workers, such as hypermedia designers, hypermedia engineers or the project manager.

The artifacts that the architect produces during the requirements capture workflow are the *requirements view of the architecture* and the *supplementary requirements*.

The profile of the architect comprises the following skills:

- expertise in the hypermedia technology domain, user modeling, adaptive techniques and some knowledge of the business domain,
- ability to communicate the architecture to the designers and engineers,
- knowledge of security and access methods,
- leadership – co-ordinating the activities of the different teams and translating decisions into activities,
- goal-oriented focusing on the resulting adaptive hypermedia application.

---

### **Hypermedia Analyst**

The hypermedia analyst is responsible for the requirements elicitation and use case modeling. He outlines the adaptive hypermedia application's functionality by finding out the access and navigation needs of the users as well as the adaptive techniques that can be applied according to the author's vision and the potential

user groups. This adaptation functionality is described as rules in textual or formal form. The hypermedia analyst delimits the systems functionality and ensures the completeness and consistency of the use case model.

The artifacts that are produced by the hypermedia analyst during the requirements workflow are the use case model, the adaptation rules and the documents describing non-functional requirements.

The profile of the hypermedia analyst is based on the following skills:

- general knowledge of the business domain and hypermedia technology domain,
- general knowledge of user modeling and adaptive systems,
- good communication skills,
- UML expertise for the use case modeling.

---

### **User Interface Designer**

The user interface designer is responsible for the visual modeling of the user interface. He thus has to capture the requirements of the user interface during the elicitation process (Preece et. al, 1994). These requirements include usability requirements, which involve stakeholders, especially end-users. He should provide a user interface description, a user interface model or a prototype, but should not implement the user interface. The focus is on the visual shaping of the user interface. The actual implementation is carried out by other workers during the implementation workflows.

The profile of the user interface designer is determined by the following skills:

- the ability to translate stakeholders' ideas into hypermedia windows or Web pages,
- design skills necessary for the creation of adaptive presentation and navigation, and
- good communication skills necessary to capture usability requirements.



---

### **Activities**

The activities of the workflow describe the dynamics in the requirements capture workflow. The goal is to represent these requirements as use cases (see Figure 7-3).

The activities needed to capture the requirements of an adaptive hypermedia application focussed initially on the user and on the domain. Such activities



includes identifying users, eliciting information needs or capturing common vocabulary. Next, activities are carried out to find the kind of application to be developed. These are eliciting navigation needs, eliciting user interface needs, eliciting adaptation capabilities, eliciting additional requirements and prototyping the user interface. The final group of activities focuses on modeling the requirements captured as use cases. These activities are: finding actors and use cases, detailing, prioritising and structuring use cases.

---

### **Identify Users**

Users are the principal actors of adaptive hypermedia systems. This task involves finding users, interviewing them (if possible), characterising them and describing them. The goal is to delimit the adaptive hypermedia system from the environment and obtain information needed to build the user model as well as to define the adaptation rules. The objective is to identify user's characteristics, such as their tasks, preferences, interests and knowledge of the domain topics. An interview technique can be used for this elicitation process (Cordingley, 1989; König, 1976; Koch & Turk, 1997). It consists of the following activities:

- identification of relevant information in the predefined checklists,
- preparation of questionnaires based on the checklists,
- execution of the interview, and
- documentation of the results.

The following questions are part of the questionnaires prepared to identify users of an adaptive hypermedia system:

- Who will interact with the adaptive hypermedia application?
- What tasks do the users have?
- What functionality do they expect from the application?
- What (computer, language, cultural) background do they have?
- What background knowledge do they have?
- What kind of tools do they frequently use?
- What experience do they have in using similar applications?
- Can different groups clearly be distinguished?
- Which attributes define these users or user groups? For example, what age range is the user population expected to belong to?

The activity *identify users* is performed by the business expert. The result of this activity is a description of the user's goals, task, preferences and knowledge about

certain themes. The construction of a business model based on these results can be helpful as starting point of the analysis stage.

*Example: Online Library*

The *Online Library* will be used by different groups of users. According to personal attributes, such as age, background, knowledge, task and occupation it is possible to isolate the following groups: students, researchers, teachers, project managers and professionals. A common characteristic of these users is that they have a clear interest in specific topics, although these preferences may change over time. Usually they can also clearly describe which topics do not interest them at all.

---

**Elicit Information Needs**

The aim of this activity is to find out what information must be included in the hypermedia application. It is useful to establish both the depth and the breadth of the content. If a business model is available, it can be used to obtain an initial approximation of the information needed. Interviewing customers and potential users is another technique that can be applied. In many cases the scope of information is partially limited by existing content or the effort that may be put into adapting it to the application's requirements.

The following questions may be helpful eliciting the information users need:

- What information are the users interested in?
- What will they search for?
- What content is already available and in what form?
- How can the information be organised in small "chunks" that deal with one topic, theme or idea?
- What content needs to be developed?
- Who can provide the contents?
- How long are users prepared to wait before information is updated?
- How can the risk of information overload be avoided?

The task of *eliciting information needs* is performed by the business expert. The result of this activity is the content description.

*Example: Online Library*

The users of an *Online Library* are interested in publications, articles and authors. An existing library database provides information about books, journals and

proceedings. Information about authors can be found on their homepages, but the quality of the data differs from one homepage to another. A publication record includes a title, a number, a publisher, a publishing date, a set of articles and authors for each article. Books consist of one single article, the title of which is the same as the book title. An article has a title, one or more authors, an abstract, a set of keywords and a document with the complete article which may be provided in different file versions, such as PDF, PostScript, HTML or ASCII format. For each author at least, the name, postal and e-mail address will be recorded.

---

### **Elicit Navigation Needs**

The aim of this activity is to find out how the information is accessed in the hypermedia application. Interviewing potential users is a technique that may also be applied in this case.

The following questions may be helpful eliciting the navigation facilities users need:

- What information do users want to see at first glance?
- What are the typical searches they will perform?
- What are the most frequent searches they will perform?
- How can the length of navigation paths be optimised?
- How can the system adapt itself to assist users in their navigation activities?

The task of *eliciting information needs* is performed by the hypermedia analyst. The result of this activity is a set of scenarios. These scenarios consist of a description of the typical navigational behaviour of the users.

#### *Example: Online Library*

The users of an *Online Library* are primarily interested in finding articles related to certain topics as well as finding all information related to a given author of a publication. They perform typical searches, such as by publication title, article title or author name. Some users will utilise the *Online Library* frequently to locate again published material they have visited before. Another service the library should provide to the users is the information about new publications related to topics that are of user interest. They should be informed about such publications news in an application's news page or by e-mail.

---

### **Elicit Adaptation Capabilities**

The aim of this activity is to find out which adaptive capabilities are required to improve the system. Observing user behaviour when interacting with a similar, but non-adaptive hypermedia application can supply useful information.

The following questions may be helpful to elicit the adaptation capabilities:

- How is the user behaviour captured?
- What content should be adapted to the user's interests, goals, or knowledge?
- Which links should be annotated, sorted or hidden, based on the current state of the user model?
- Should the system become active, when the user is inactive?

The task of *eliciting adaptation needs* is performed by the hypermedia analyst. The result of this activity is a set of adaptation rules. These are usually presented as a textual, non-formal description in the first iteration. The specification of these rules can then be made more formal in successive iterations.

#### *Example: Online Library*

Users can choose between a number of initial options, such as type of file for the document containing the complete article and how they want to be informed about news, i.e. by e-mail or through the news page. The content of the application is then adapted to these settings.

In addition, user browsing behaviour is observed and registered to find out the user's preferred topics, authors and publications. The user model is built and updated with the information obtained from observation of user behaviour. Publication's items, articles and author's indexes are sorted, annotated or removed, based on the current state of the user model.

The system models the user's interest in articles by registering the articles she visits. Articles can be marked by the user as being of special interest (bookmarks). A list of personal keywords for each user is administrated by the system. This list is initialised by the user and is updated either by the user or by the system. The system performs the updating in line of observations on user behaviour (in this case limited to the articles she marks or visits frequently). The list can include positive as well as negative keywords. Negative keywords are used to hide irrelevant publications and articles from the user. The user must inform the system of *negative* keywords, e.g. keywords related to topics she is not at all interested in.

The user model of the *Online Library* is visible to the user. She can modify the current values of the user model at any time.

---

### **Elicit Additional Requirements**

Supplementary requirements are primarily non-functional requirements. They are not related to the content, navigation, interface or adaptive functionality of the hypermedia system. These additional requirements cannot be included in the use case model; they are thus presented as a document that consists of a list of requirements. Additional requirements can be elicited based on the following generic list. This list is not exhaustive, i.e. it can be extended.

- Budget constraints
- Time constraints
- Hardware constraints
- Software constraints
- Design constraints
- User modeling constraints
- Implementation constraints
- Performance
- Security
- Availability
- Ergonomics
- Usability

The task of *eliciting additional requirements* is performed by the architect. The result of this activity is a list of supplementary requirements.

#### *Example: Online Library*

The *Online Library* application for the Web must be optimised for the most frequently used browsers, with the objective of nearly identical presentation. The usual security procedure for user registration and manipulation of e-mail addresses is required. A user model is built for registered users and used for customisation and adaptation.

---

### **Elicit User Interface Needs**

The aim of this activity is to find out how the information and the navigation assistance are to be presented to the user in the hypermedia application. Interviewing the customer as well as potential users is a technique that can be also applied here.

The following questions may be helpful to elicit the user interface needs:

- Must the presentation to be designed from scratch?
- Does the customer's organisation have a style guide for their hypermedia applications?
- What layout constraints does the customer specify?
- How can cognitive overload be avoided?

The task of *eliciting of user interface needs* is performed by the user interface designer. The result of this activity is a user interface description and/or a user interface prototype.

*Example: Online Library*

The presentation has to be designed from scratch. No style guide is available. No commercial banners or animations are included.

---

The remaining activities of the requirements capture workflow are only explained briefly. The objectives are the same and the tasks are performed in the same way as the corresponding activities of the UP. Examples are added, if they are required to make further sections easier to understand.

---

### **Find Actors and Use Cases**

For this activity requirements are modeled as use cases, i.e. all the requirements captured during the above-described activities are used to define the actors of the system and find the use cases that describe the functionality of the system (Schneider & Winters, 1998).

The users are the main actors of the adaptive hypermedia application. The following questions will help to find candidates for other actors:

- Who will perform the authoring work?
- Who will support and maintain the adaptive hypermedia system?
- What are the system's external resources?
- What other systems will interact with this application?

A brief description of each actor should include information about what or whom the actor represents, why the actor is needed and what interests the actor has in the system.

The best way to find use cases is to consider what each actor requires of the system. The set of functional requirements, i.e. informational, navigation and adaptation capabilities, provide answers to these questions. Workshops or interviews can also be used to understand which use cases are needed. Each use case is briefly described.

The task of *finding actors and use cases* is performed by the hypermedia analyst. The result of this activity is an outlined use case model.

#### *Example: Online Library*

In summary, the *Online Library* offers to the users the following functionality:

- access to regularly updated publication information as an anonymous or a registered user,
- dynamic updating of a user model,
- different search possibilities for publications, articles and authors,
- search mechanisms for articles, which have already been visited, and
- notification of recently published articles compiled according to the user model.

The actors in the requirements description are the User, Registered User, Library Administrator and Library System.

- User

A user represents a person browsing in the *Online Library* and navigating the hyperspace defined by this application.

- Registered User

A registered user is a person who has identified herself to the *Online Library*. The system builds a user model for this user. When the user browses in the *Online Library*, she obtains personalised information and links.

- Library Administrator

A library administrator is a person who is in charge of the updating and maintenance of the *Online Library* content.

- Library System

The library system is responsible for sending e-mails with news, for updating the user model and adapting content, navigation and presentation.

The most relevant use cases are: Find Publication, Find Articles, Find Author, Look at News, Select Visited Articles, Mark Articles, Modify User Profile, Update Publication, Update Article, Update Author, Send News per E-mail, Adapt Content, Adapt Navigation, and Adapt Presentation.

The description of one use case is included here by way of example.

Use Case: Select Visited Articles

The actor in this use case is the RegisteredUser as articles she has already visited are recorded in the user model. Note that this mark is different to bookmarking, which indicates that the article is of interest to her. The registered user can select an article from an index of visited articles that are sorted according to positive keywords. In addition to the sorted articles, items on the list are annotated to distinguish articles that have just been visited from those that are marked as being of special interest.

---

### **Detail Use Cases**

The objective in detailing each use case is to describe the flow of events in detail, including how the use case starts, ends and interacts with actors. The business expert performs this activity adding his domain knowledge to the use case model specification. The result is a detailed description of the use cases in text and diagrams (Schneider & Winters, 1998).

The detailed description of a use case includes a use case name, the list of actors communicating with the use case, its priority, the status of development of the use case, pre- and post-conditions that must be true at start and finish of the use case, a list of use cases that “extend” this use case, a list of use cases that are “included”, a flow of events describing primary scenarios. Optionally, secondary scenarios (alternatives and exceptions not shown in the primary flow of events), activity diagrams, user interfaces, sequence diagrams, views of participating classes and



other artifacts as well as other requirements and open questions can be added to description of a use case.

*Example: Online Library*

The following template is proposed for a detailed description of a use case (Figure 7-4).

<p><b>Use case name:</b> “Find Author”</p> <p><b>Actors:</b> User, Registered User, Library Administrator</p> <p><b>Priority:</b> 1</p> <p><b>Status:</b> Requirements capture</p> <p><b>Pre-condition:</b> The search form must be visible</p> <p><b>Post-condition:</b> Author’s page is shown with information about the author’s person, e.g. as name, picture, postal and e-mail address. A link to the articles published by the author is also displayed.</p> <p><b>Flow of events:</b></p> <ol style="list-style-type: none"> <li>1. The use case starts when the user selects the “author” option.</li> <li>2. The user enters the name or a keyword related to the author.</li> <li>3. The user starts the search mechanism</li> <li>4. If the result is more than one author then             <ol style="list-style-type: none"> <li>a) the system displays an index of authors matching the user’s input</li> <li>b) the user selects one</li> </ol>             end if           </li> <li>5. The system displays the information about the author</li> <li>6. The system offers a link to the articles of the author.</li> <li>7. The use case ends</li> </ol> <p><b>Secondary scenario:</b></p> <ol style="list-style-type: none"> <li>2. If the system does not find a matching author, the user is asked to re-enter the keywords or author’s name.</li> <li>6. If the user is registered, the author index is sorted and annotated in accordance with the current state of the user model.</li> </ol>
---

*Figure 7-4: Template for Use Case Description*

### Prioritise Use Cases

The purpose of this activity is to determine which priority will have each use case for the development, i.e. design, validation and implementation. The activity is performed by the architect and the result is visible in the architectural view of the use case model, which includes only the critical actors and use cases.

*Example: Online Library*

The “search” use cases, such as Find Publication, Find Article, Select Visited Article, Find Author and the use case Mark Article have highest priority. The implementation of these use cases will give the customer the “look and feel” of the system. Priority two is assigned to the use cases defining updating of the database through special forms. At a third stage the “news” features will be designed and implemented. Finally, the remaining functionality is added, such as modification of the user model through a form.

### Structure Use Cases

During this activity a complete use case model is built, i.e. the relationships between actors and use cases are analysed in detailed. Relationships of type «includes» and «extends» are established between use cases as well as generalisations between actors. A detailed flow of events with a pre-condition and a

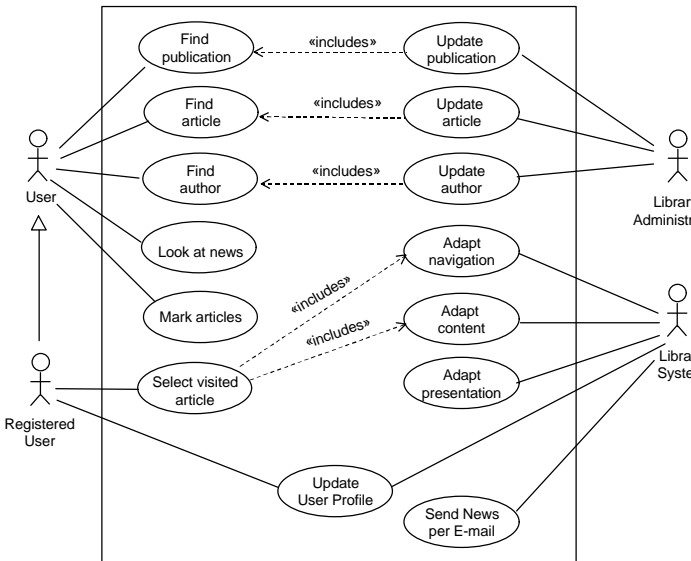


Figure 7-5: Use Case Model for the Online Library Application

post-condition can be defined in textual form as well as represented with a UML activity diagram. The result is a detailed use case model.

*Example: Online Library*

A rough granulated Use Case Model for the *Online Library* is shown in Figure 7-5: Note that additional dependencies of type «includes» relate the use cases Find Publication, Find Article and Find AuthOr with the use cases Adapt Content, Adapt Navigation and Adapt Presentation if the RegisteredUser is who navigates and searches in the application.

---

### **Prototype User Interface**

During this activity a first approach to the visual aspects and distribution of the user interface elements is prepared. The activity *prototype user interface* is performed by the user interface designer, who produces the user interface prototype.

---

### **Capture a Common Vocabulary**

The objective of this activity is to define a common vocabulary that can be used in all textual descriptions of the system, especially in use case descriptions. This common vocabulary is the basis for communication between all stakeholders. The result is a glossary produced by the business expert.

*Example: Online Library*

The following are some entries in the *Online Library* glossary:

<p>....</p> <p><b>active help:</b> short description for each navigation button, may be suppressed by the user. This change has to be registered in the user model for registered users.</p> <p><b>news:</b> are data and documents related to articles that have recently be published (period of time may be customised).</p> <p><b>publication:</b> includes books, proceedings and journals. Books consists of just one article bearing the same name as the publication title.</p> <p>.....</p>
--

*Figure 7-6: Part of the Glossary*

### 7.3.2 Analysis and Design

The purpose of the analysis and design workflow is to translate the requirements description (obtained in the previous workflow) into a specification that describes how to implement the adaptive hypermedia application. Analysis focuses on the application's functional requirements, ignoring non-functional requirements and implementation constraints. During design the analysis results are adapted to the conditions imposed by the non-functional requirements. The design is seen as a refinement process of the analysis. In this work, both analysis and design are described together. When the term "design" is used, it means analysis as well.

The design workflow of UWE consists of a model-based and user-centred approach for building adaptive hypermedia applications. It is model-based because for almost all activities a UML-model is build. It is user-centred because it takes into account user properties for the construction of these models. The design comprises the following activities:

- conceptual design,
- user model design,
- navigation design,
- presentation design,
- adaptation design,
- architecture design,
- detail design classes, and
- definition of subsystems and interfaces.

The artifacts produced as results of these activities are the design view of the architecture, the conceptual model, the user model, the navigation model, the presentation model, the adaptation model, design classes, subsystems and interfaces. These models are constructed based on the UML Profile presented in Chapter 6. The UML Profile defines several stereotypes for the navigation, presentation and adaptation modeling according to the extensions mechanisms of UML. The main workers that perform these activities producing the mentioned results are the architect, the hypermedia designer and the hypermedia engineer. Figure 7-7 depicts the analysis and design workflow showing workers, activities and artifacts.

The *conceptual*, *navigation* and *presentation design* activities provide a clear separation of the information the user can access (domain semantics), how this information is structured and how it is presented to the user. Based on the domain model the activities navigation and presentation design take into account the

special characteristics of the hypermedia paradigm, i.e. the navigation functionality and the multimedia user interface. Schwabe and Rossi (1998) stress, that as a result of this separation a more modular and reusable design is obtained. They propose a framework to reason about the design process, encapsulating the specific design experience to each step.

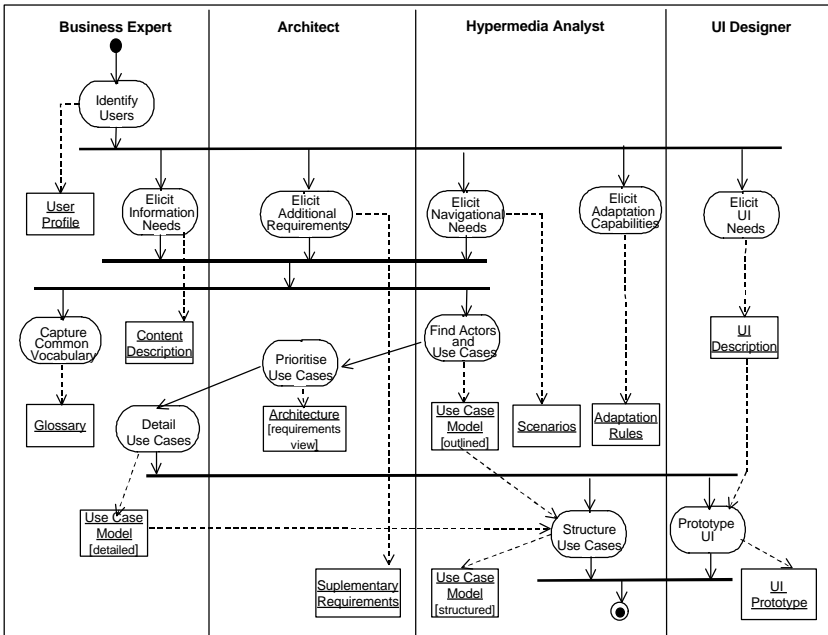


Figure 7-7: Analysis and Design Workflow

The *user design* together with *adaptation design* cover the special aspects related to adaptability. They cover user modeling and adaptation of content, navigation and presentation. User modeling consists of the construction, update and utilisation of a user model to help users in accordance their preferences, knowledge, interests, or tasks through adaptive content, navigation and presentation. Alternative content for the same concept is required to show each user an appropriate version of the concept. The same information can have different layouts for different users. Different navigation paths result from adaptive navigation. Consequently, some information or some nodes may be visible for some users but not for others. Rules are defined to specify the adaptive behaviour of the hypermedia system.

The purpose of the *architecture design* is to outline the design and deployment models and the system's architecture. This requires the identification of architecturally significant design classes, subsystems and their interfaces, nodes and their network configurations as well as special requirements as regards persistency, distribution and performance. The last two activities on the list, i.e. *detail design classes* and *define subsystems and interfaces* are performed by the hypermedia engineer in the late iterations of the design. The goal is to detail the design classes and group them into subsystems preparing them for the next workflow where the focus is on implementation. The design of hypermedia applications is an incremental, iterative and sometimes a prototype-based process.




---

## **Artifacts**

The artifacts produced in the analysis and design workflow are a design view of the architecture, a set of models, design classes, subsystems and interfaces. The design models are the – already mentioned – conceptual, user, navigation, presentation and adaptation models. A set of stereotyped modeling elements are defined in Chapter 6 as well as the method, which steps can be followed in the construction of the models. These steps are mainly helpful for an automated development of hypermedia applications supported by a case tool.

---

## **Conceptual Model**

The conceptual model is a model of the problem domain, the aim being to leave out navigation, presentation and adaptation aspects that characterise an adaptive hypermedia application. The conceptual design is thus similar to business or problem domain modeling for traditional software development. An adaptive hypermedia application requires identification of the concepts units and which must be available in the different versions. The adaptation model then establishes at run-time, criteria that are used to decide which of these versions is appropriate for the user. The decision is based on the current state of the user model. A conceptual model is represented as a UML class model (see Figure 7.8).

---

## **User Model**

A user model contains information that represents the view the system has of the knowledge, goals and/or individual features, such as preferences, interests and tasks of the users. The main purpose of including a user model is to support an application that dynamically adjusts itself to the user. Information contained in the user model will influence the layout of the user interface, navigation through the application and adaptation of the content of the nodes the user accesses. The role

the user model plays in an adaptive hypermedia system can be seen in the Munich Reference Model presented in Chapter 4.

A user model is described by a set of classes describing the user attributes that are modeled and their relationships, if any, to classes of the conceptual model. Classes and associations are represented with a UML class diagram (see Figure 7-9).

---

### **Navigation Model**

The navigation model is built in two stages. During the first stage a *navigation space model* is constructed based on the conceptual model. The result of the second stage is a *navigation structure model* that is build on the navigation space model.

*Navigation space model* defines a view of the conceptual model showing which classes of the conceptual model can be visited through navigation in the application. It is represented as a UML class diagram (Figure 7-9) built with a set of navigation classes and associations between these navigation classes. Classes and associations are mainly obtained from the conceptual model. The designer decides if additional associations representing direct navigability are required.

In the navigation space model navigability is specified for associations, i.e. direction of the navigation along the association is shown through the arrow attached to the end of the association's line. For each link a navigation source object and a navigation target object are distinguished.

The *navigation structure model* defines the navigation of the application, i.e. how navigation objects are visited. It is based on the navigation space model, but it also includes additional model elements – access elements – that are required to perform the navigation between navigation objects. These access elements are *menus, indexes, guided tours and queries*. The navigation class diagram is represented with a UML class diagram (see Figure 7-11). The UML Profile defined in Chapter 6 includes stereotyped classes for navigation and access elements.

---

### **Presentation Model**

The presentation model is the representation of an abstract user interface, showing how the navigation structure is presented to the user. The same navigation structure may yield different presentations depending on the restrictions of the target platform and the technology used.

Most of the methods for hypermedia design only suggest the development of prototypical pages for this activity. In this work, it is proposed instead to define a

presentation model as a composition of user interface objects. UML modeling elements and UML diagrams are chosen as a technique in the same way as for the conceptual and navigation model. The presentation model is a rough design of the user interface; decisions about details such as size, colour or font of user interface elements are taken when developing the prototype or in the implementation phase.

The presentation model consists of a static presentation model and a dynamic presentation model.

The static presentation consists of a *presentation structure model* and an *abstract interface model* represented by UML class diagrams and UML composition diagrams, respectively. The *presentation structure model* describes where the navigation objects and access primitives are presented, i.e. in which frame or window the user will see them displayed, as shown in Figure 7-12. The following stereotyped classes are defined for the presentation: *window*, *frameset*, *frame* and *presentation class*. Two alternative for presentation models are presented: a menu-based and a map-based presentation.

The *abstract user interface model* provides sketches of the user interface. It consists of a collection of user interface objects that shows the composition of presentation objects by other presentation objects and relationships between these objects (Koch, 1998; Koch & Mandel, 1999). For the most frequently used user interface objects special modeling elements (stereotyped classes) are defined. These are: *anchor*, *text*, *image*, *audio*, *video*, *form*, *button*, *collection* and *anchored collection* (see Chapter 6 for more details). For each navigation object at least one presentation object has to be defined. If the presentation depends on the navigation context within which the navigation object is visited, one presentation object for each context has to be specified. Hints are provided on how the navigation objects are presented to the user. For example, a first approach to position and size of the user interface elements relative to each other is given. Examples of presentation classes are shown in Figures 7-13 to 7-17.

The dynamic presentation consists of a *presentation flow model* and an *object life cycle model* represented by UML sequence diagrams and UML state diagrams, respectively. Sequence diagrams are used to describe the flow of control between presentation elements when a multiple-window technique is used (Henicker & Koch, 2000b). State diagrams are used to visualise the object life cycle of interactive presentation objects. This lifecycle is defined through states and through transitions between states. A state is specified by a name, entry and exit actions, internal transitions, and/or sub-states. A transition has an action associated with it, i.e. it is triggered by an event. In the case of user interfaces most of the events are generated by the user, such as mouse focus, mouse clicks, or keyboard inputs. Complex behaviours can be modeled easily in UML with sequential and concurrent



substates (UML, 1999). The design of these UML state diagrams is expensive and usually so many details are not necessary for user interface objects with well-known behaviour. They are therefore only used for complex composite user interface objects.

---

### **Adaptation Model**

The adaptation model presents the objects that participate in the adaptive functionality and describes how this adaptation is performed. The adaptation model consists then of a set of rules described textually (or with a formal language) and a set of UML collaboration diagrams (see Figure 7-19).

The adaptation rules are defined in the adaptation model specifying the conditions under which the content, navigation and presentation are adapted, which actions are performed for the adaptation and how the user model is updated. The representation of the rules in a model show how these rules collaborate with objects of other models, such as the navigation, presentation and user model. A detailed description of the adaptive functionality is given in Chapter 2 and it is modeled in the Munich Reference Model for adaptive hypermedia systems presented in Chapter 4.

---

### **Architecture (design view)**

The design view of the architecture depicts the architecturally important artifacts. They are:

- the subsystems, which the system is divided into,
- the interfaces of these subsystems,
- key design classes that represent some generic design mechanisms and have many relationships to other classes, but which are not necessarily detailed in the architectural view, and
- design classes that are related to the realisation of key use cases.

---

### **Design Class**

During the first iterations in the analysis and design workflow classes are outlined, i.e. more relevant attributes and operations are described. In successive iterations details are added to finally produce a design class. An adaptive hypermedia system requires the definition and the detailed description of design classes for the user, navigation, presentation and adaptation model.

For each design class:

- all attributes and operations are defined,
- visibility of attributes and operations is often specified,
- relationships in which the design class is involved sometimes implies the addition of attributes to the design class,
- the methods, i.e. the realisation of the operations, is detailed in natural language or in pseudo-code, and
- active classes are identified, i.e. classes, which objects maintain their own threads.

---

### **Subsystem**

Subsystems are defined to group artifacts of the design models in more manageable pieces or to separate design concerns. A subsystem consists of design classes, interfaces, use cases or other subsystems. A subsystem can be used to represent legacy systems or part of them or to represent reuse software components. Another purpose for which subsystems are used is to encapsulate the content, showing behaviour only through the interfaces of the subsystem. A natural group of subsystems is provided by the models of the layers defined by the reference model. For example, for the storage layer there are the domain, user and adaptation subsystem. A more fine-grained grouping of classes can be performed, which results in a greater number of subsystems.

---

### **Interface**

An interface is used to specify the operations provided by design classes and subsystems. A design class that provides an interface must also provide methods that carry out the operations of the interface. A subsystem that provides an interface has to contain a design class that provides this interface.



---

### **Workers**

The analysis and design in the development of an adaptive or non-adaptive hypermedia application is performed by the following workers at least: an architect, a hypermedia designer and one or more hypermedia engineers. The role of the hypermedia designer can be performed by several hypermedia experts with different profiles, such as navigation designers, multimedia experts and graphic designers.

---

### **Architect**

The architect is responsible in the design for the integrity of the *user model*, the *conceptual model* and the *design view of the architecture*.

The profile of the architect is defined in the requirements capture workflow.

---

### **Hypermedia Designer**

The hypermedia designer is responsible for the use case realisation. The functionality described in the use cases has to be reflected in the navigation structure, dynamic page generation and the adaptive user interface of the hypermedia application. The hypermedia designer develops the navigation model, the presentation model and the adaptation model.

The profile of the hypermedia designer is provided by the following skills:

- knowledge of the functional and non-functional requirements of the system,
- general knowledge of the business domain and hypermedia technology domain,
- experience in the design of the structure of a hypermedia system,
- general knowledge of user modeling and adaptive systems, and
- UML knowledge for the modeling activities.

---

### **Hypermedia Engineer**

The hypermedia engineer details and maintains the attributes, operations, methods, relationships and implementation requirements of one or more *design classes* as well as the integrity of one or more *subsystems* and *interfaces*. It is often appropriate to let the same hypermedia engineer be responsible for a subsystem and the modeling elements contained in the subsystem. The implementation is then done by the same hypermedia engineer who takes advantage of knowledge of these modeling elements.

The profile of the hypermedia engineer is provided by the following skills:

- UML knowledge for modeling activities,
- experience in the implementation language that has been chosen,

- know-how related to logging and event monitoring procedures, security and access methods and site “policies”,
- knowledge of the technologies with which the system will be implemented, and
- experience e.g. in HTML, JavaScript, JSP, ASP, database definition and generation, multimedia design and/or integration, etc.



---

## **Activities**

Throughout the analysis and design workflow, the designers will perform a set of activities to create the user model, content, structure and interface of the adaptive hypermedia system as well as to define the adaptation mechanisms. These models consists of classes and relationships that can be grouped into subsystems.

The following activities are included in this workflow: conceptual design, user model design, navigation design, presentation design, adaptation design, architecture design, detailed design of classes, and definition of subsystems and interfaces.

---

## **Conceptual Design**

The activity conceptual design aims to build a domain model including all the concepts that are relevant to the application and the different users or user groups identified in the requirements capture workflow. The main objective is to capture the domain semantics with as little attention as possible paid to the navigation paths, presentation and interaction aspects. Decisions as to whether each concept corresponds to one hypermedia page, a hypermedia document or the page is being generated on-the-fly based on the frame-based internal representation of domain concepts, are postponed to the implementation phase.

Activities related to the conceptual design are typical object-oriented modeling activities, such as identification of classes, determination of associations between classes, and definition of constraints. More details of these activities are given in Section 6.1 of the previous chapter.

Well-known object-oriented modeling techniques are used at this stage, such as composition, generalisation and specialisation. Classes are defined by a name, attributes, operations and variants. The compartment of a class named *variant* contains additional information required for the adaptive content functionality, i.e. to present different content to the user in accordance with the current state of her user model. UML packages can be used to group classes and associations. OCL

constraints can be included in the diagram or specified separately (see example below).

The results of this activity is a UML class model of the problem domain. Classes and associations defined in this step are used during navigation design to derive nodes of the hypermedia structure. Associations will be used to derive links.

*Example: Online Library*

The conceptual model for the *Online Library* is shown in Figure 7-8. The example is restricted to this kernel data and functionality although the *Online Library* should also include authoring functions. Authoring functions are needed e.g. to allow the user to visualise and perform changes in her user model.

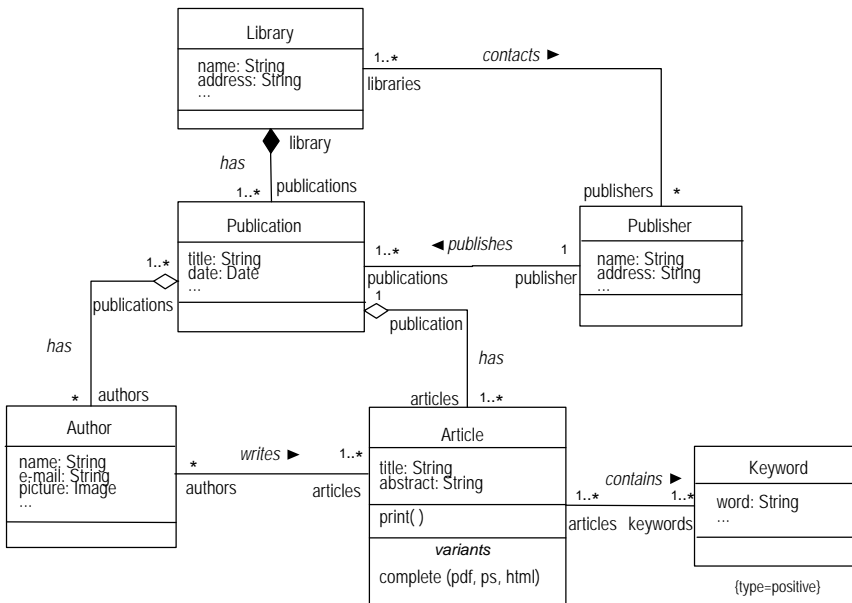


Figure 7-8: Conceptual Model of the Online Library Application

## User Model Design

The user model design aims at building a user model that represents knowledge, goals and/or individual features, such as preferences, interests and tasks of the

users. The model is the view the system has of the user. The main reason for including a user model is to support an application that dynamically adjusts itself to the user. Information contained in the user model will influence the layout of the user interface, navigation and content of the presentation the user accesses.

Activities of the user model design are between others the selection of type of user model (see Chapter 3), definition of a user class and user attribute class and categorisation of attributes in dependent and independent of the domain. More details of these activities are provided in Section 6.2 of the previous chapter.

In the case of a stereotyped-based user model, an instance of the user class is defined for each stereotype, i.e. for each user group. If an individual user model is created, an instance of the class user and of the user attributes are generated for this new user. This includes the beliefs the system has about the specific user. Sometimes stereotypes are used to initialise user models, i.e. stereotypes are used for initial assumptions instead of using an initial questionnaire completed by the user.

Note that stereotype has a different meaning than in Chapter 6 where a UML stereotype is a UML extension mechanism to define new modeling elements. Here, stereotype is used to define a small set of user models that are used then to classify users and assign the properties associated to the most appropriate model for each of them. The static aspects of the user model are described using a UML class model.

*Example: Online Library*

The following characteristics of the users are included in the user model of the *Online Library* application: articles the user visits, articles that are marked by the user, positive and negative keywords, preferences the user chooses to be informed about, new articles and the type of file she selects for the download of the articles. Some of the values of these user attributes are updated dynamically by the system; others are set by the user and can only be changed by her. The user model for the *Online Library* is shown in Figure 7-9. Class Article of the conceptual model is appended to the user model diagram to show how the user model is related to the conceptual model.

Class Visited registers how often an article is visited by the user. The class Marked models the articles that are marked by the user (bookmarks). They are part of the domain dependent knowledge, i.e. one instance of these classes is required for each instance of the class Article. Instead, class UserKeyword models themes of interest. It is considered background knowledge and its instances are not related to specific instances of domain classes. Classes FileType and News model preferences of the user.

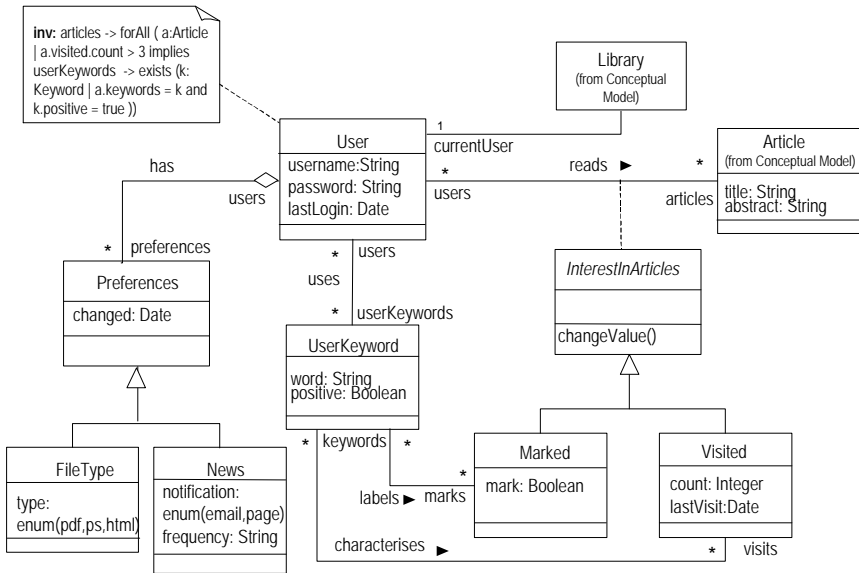


Figure 7-9: User Model of the Online Library Application

## Navigation Design

Navigation design is a critical step in the design of every hypermedia application. Even simple applications with a shallow hierarchical structure will very soon become complex as a result of the addition of new links. Additional links improve navigability on the one hand but, on the other hand, imply higher risk of losing orientation. Building a navigation model is not only helpful for the documentation of the application structure, it also allows for a more structured increase in navigability.

The *navigation design* defines the structure of the hypermedia application and describes how navigation can take place. The basis of the navigation design is the conceptual model and the outcome is a navigation model, which can be seen as a view over the conceptual model. The navigation model is defined in a two-step process. In the first step – the *navigation space model* – is specified, i.e. which objects can potentially be reached through navigation and in the second one – *navigation structure design* – how these objects are reached. Hence additional objects are required to access navigation objects.

The *navigation space model* can be seen as a sub-graph of the conceptual model where some classes which are not relevant for the navigation are eliminated and/or reduced to attributes of other classes.

The *navigation structure model* defines the navigation of the application, i.e. how navigation objects are visited. It is built starting with the navigation space model and including additional modeling elements, such as access primitives (menus, indexes, guided tours and queries) and properties to model adaptive navigation.

The activities that are performed for the navigation design in a three-step procedure are presented in detail in Section 6.3 of the previous chapter. There the modeling elements mentioned above are also defined. Navigation classes, access primitives and associations with navigability are graphically represented in UML class diagrams.

#### *Example: Online Library*

Following the steps of the navigation design method presented in Chapter 6 a navigation space model for the *Online Library* is built. This model is shown in Figure 7-10.

1. Classes of the conceptual model that are relevant classes for navigation are: Library, Publication, Author and Articles.
2. Conceptual classes Publisher and Keyword are not included as navigation classes, but as derived attributes of Library and Article, respectively.
3. Additional associations between Library and Author are included to allow direct navigation between instances of these classes.
4. Three additional associations between Library and Article, i.e. all articles, visited articles and news (new articles) are added, based on the scenarios described in the requirements capture workflow.

As shown in Figure 7-10 constraints can be attached in a UML note or they can be specified separately as the invariant for visited articles listed below.

**context** Library

**inv:** visitedArticles → select ( a:Article | a.visited.count > 1  
and lastVisit.year = currentYear )



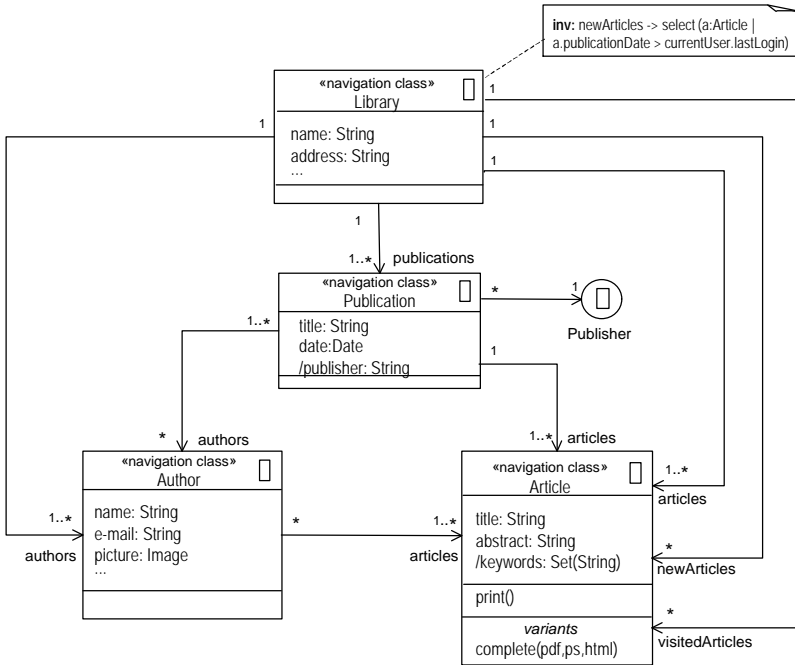


Figure 7-10: Navigation Space Model of the Online Library Application

The navigation structure model for the Web Site of the *Online Library* is built based on the methodical activities described in Section 6.3 of the previous chapter. Figure 7-11 shows the result of these activities:

1. Enhancement of the navigation space model with access elements for all navigation classes which have multiplicity greater than one at the directed association end. Movement of role names from navigation classes to the access elements. Access elements added in the example are: PublicationByTitle, ArticleByTitle, NewArticleByTitle, VisitedArticleByTitle, AuthorByName, ArticleByTitleByPublication (guided tour), SearchAuthorByName (query), etc.
2. Addition of menus to all navigation classes, which have at least one outgoing association, such as LibraryMenu, PublicationMenu and AuthorMenu. Menus have a relationship of type composition with the corresponding navigation classes. Associations between navigation

classes are transformed in associations between menu and target navigation classes.

3. Specification of properties {direct guidance} for guided tour, {sorted}, {annotated}, and {removed} for access primitives to indicate adaptation of navigation to the user model.
4. Addition properties of type {passive navigation} to indicate navigation performed by the system. In the sample application passive navigation is added from every navigation object to the NewArticleByTitle index; in the diagram only one of these associations adorned with the {passive navigation} property is shown.

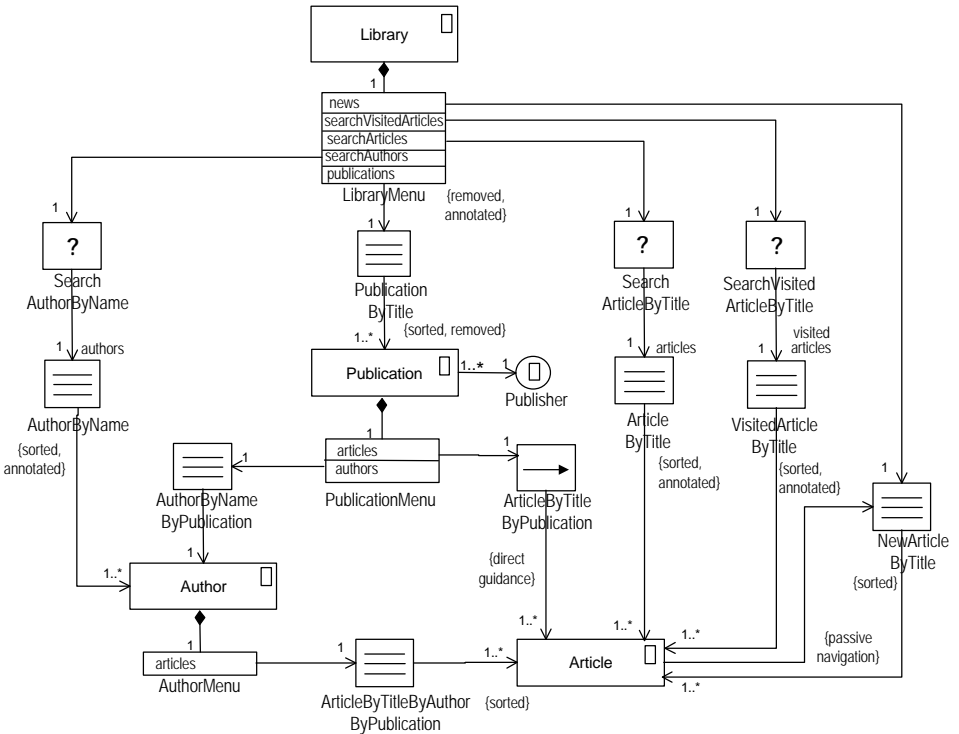


Figure 7-11: Navigation Structure Model of the Online Library Application

## Presentation Design

Presentation design consists of the definition of where and how the navigation objects are presented to the user. The user interface is modeled with the help of static and dynamic object-oriented models depicting the layout in a schematic way.

The designer chooses in this step between a multiple-window and a single-window technique, and between a presentation style with or without frames. The static presentation model then associates each presentation class to a window or frame and describes how the attributes of a presentation class are shown to the user. The static presentation models are derived from the navigation structure model. These models are: the presentation structure model and the abstract user interface model.

Different types of presentations can be constructed, such as the menu-based or map-based presentations. The first one consists of a collection of presentation objects, where navigation is guaranteed by a main menu and indexes. The last one is also called a tree-structured technique. It supports the visualisation of the navigation structure and has the advantage of mitigating the problem of “lost in hyperspace”. One application can use both presentation techniques in combination. Map-based presentations supports the visualisation of the total or partial navigation space. Both techniques use framesets to include all the presentation classes that are presented to the user at a glance.

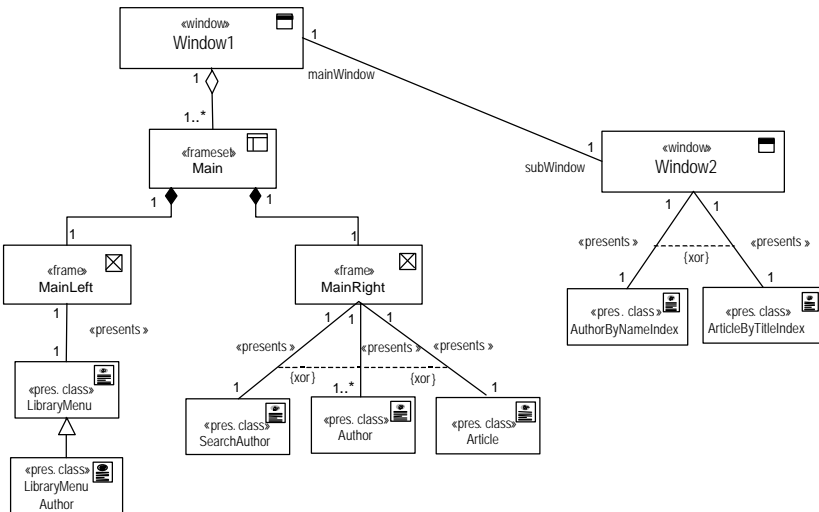


Figure 7-12: Presentation Structure Model of the Online Library Application (Partial View)

The goal of the *dynamic presentation design* is to describe the behaviour of the presentation objects, i.e. the changes on the user interface when the user interacts with it or when the system reacts to internal events such as timeouts. Two type of models can be constructed to represent different aspects of the dynamic of an (adaptive) hypermedia application. There are: object lifecycle models and presentation flow models.

*Object lifecycles models* are used to model the behaviour of complex presentation objects and the influence they have on the status of other presentation objects (see Chapter 6). UML state diagrams are used to represent these object lifecycles. As the design of state charts is time consuming, they are built only if the complexity of the behaviour of the presentation objects makes this necessary.

The *presentation flow model* is graphically represented by UML interaction diagrams, e.g. UML sequence diagrams. These diagrams show which windows and frames can be opened, which are active, and which objects are displayed in each window or frame at a certain moment.

See Section 6.4 of the previous chapter for a detailed description of the systematic construction of these models and the definition of the stereotypes used. A UML class diagrams and composition are used for the graphical representation of the presentation structure model and the abstract user interface model.

### Example: Online Library

Figure 7-12 shows a partial view of the presentation structure model.



Figure 7-13: Presentation Class of Library Main Menu

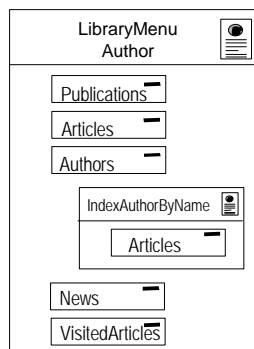


Figure 7-14: Presentation Class of Composite Library and AuthorMenu

Figure 7-13 to Figure 7-17 show some sketches of the abstract user interface model for the sample application *Online Library*. For a more detailed description see Section 6.4.

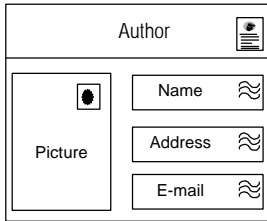


Figure 7-15: Presentation Class  
*Author*

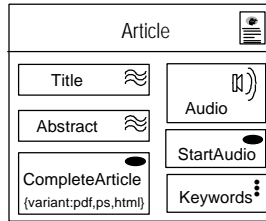


Figure 7-16: Presentation Class  
*Article*

Figure 7-17 depicts one frameset of the *Online Library* which has two frames. The left frame presents the presentation class of the main menu application and the right frame presents the selected content. Sometimes this kind of representation results useful as it gives an idea of how pages of the application will look, i.e. a sketch of the user interface. Note that the *LibraryMenu* includes an additional item that allows navigation back to the starting point of the application.

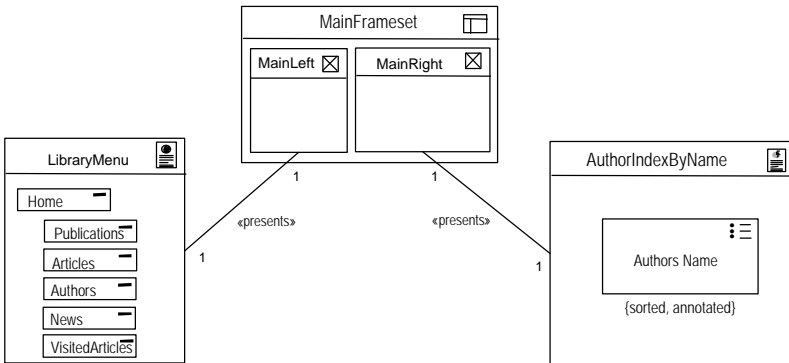


Figure 7-17: Abstract User Interface Model of one Online Library Page

A part of a presentation flow model for the sample application *Online Library* is shown in Figure 7-18. It consists of the representation of the message flow between user, window objects and frame objects when the user wants to go from the start page (root of the tree) to the presentation of an article of a certain author.

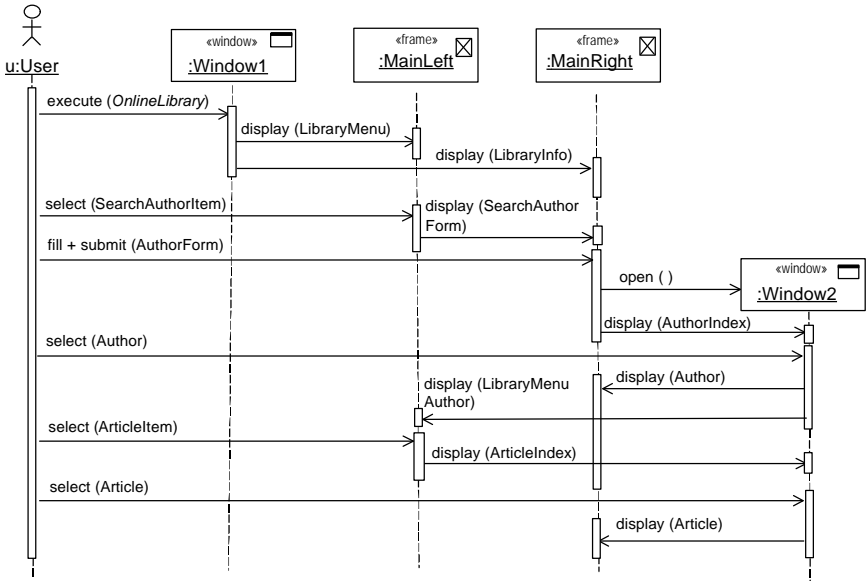


Figure 7-18: Part of Presentation Flow Model of the Online Library Application

## Adaptation Design

Adaptation design consists of the definition of adaptation rules and the graphical representation of these rules in a UML collaboration model. The rules specify the conditions under which the content, the navigation and the presentation are adapted, which actions are performed for the adaptation and how the user model is updated according to the observations of the user behaviour. The model shows how rules collaborate with user behaviour, navigation, presentation and user model elements.

Three types of hypermedia adaptation are distinguished: adaptive content (content-level adaptation), adaptive navigation support (link-level adaptation) and adaptive presentation (layout-level adaptation). The first presents to the user content that

has been adapted to the current state of her model. The second consists of suggesting the “best” link, sometimes forcing the user to follow a determined path. The most popular techniques for adaptive navigation are: direct guidance, adaptive ordering, adaptive hiding and adaptive annotation. The third type adjust the layout without making changes to the content choosing e.g. different fonts, size of images, colours, etc.

*Example: Online Library*

In Section 6.5 of the previous chapter a list of rules of the *Online Library* application is presented. The following three rules are those used in the adaptation model shown in Figure 7-19.

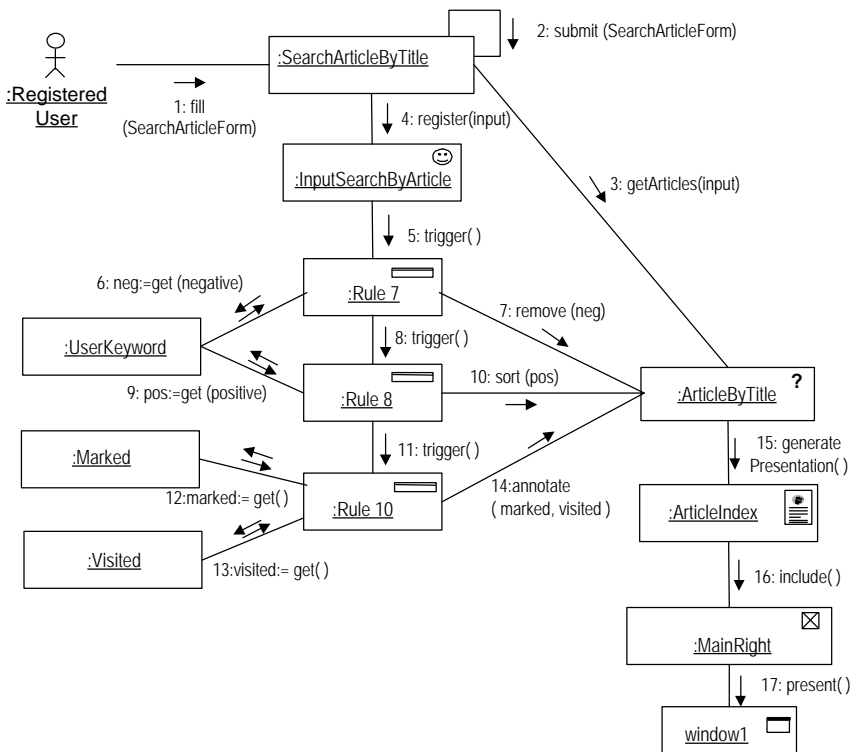


Figure 7-19: Adaptation Model of the Online Library Application (Partial View)

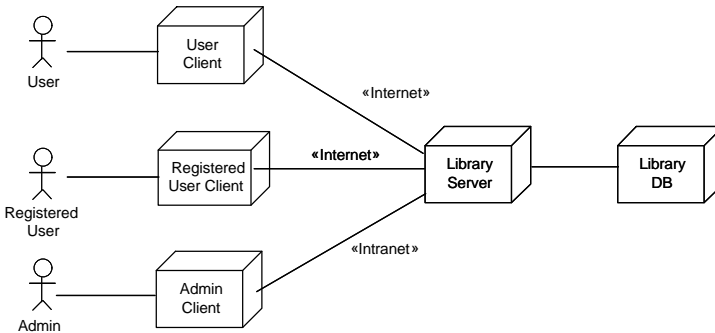
- Rule 7: An article included in any ArticleIndex or ArticleGuidedTour is removed if the negative keyword list includes two or more keywords from the article.
- Rule 8: The articles in the ArticleIndexByTitle, VisitedArticleIndexByTitle, NewArticle ByTitle and ArticlesGuidedTour are sorted based on the positive keywords of the user model.
- Rule 10: Annotation is performed in the article indexes as follows:
  - red bullets for articles not visited and not marked,
  - white bullets for visited but not marked, and
  - blue bullets for visited and marked.

The UML collaboration diagram shows the adaptation process that begins when a SearchArticleByTitle form is filled by the user. The list of articles is provided by the ArticleByTitle context, which is adapted through elimination of links and through addition of links given by positive keywords. The graphical visualisation of the model permits the recognition of loops in the flow of rules triggered by other rules.

## Architecture Design

The purpose of the architecture design is to outline the architecture (design view) by identifying the following:

- subsystems and their interfaces,
- design classes, that are relevant for the architecture,
- generic design mechanisms to handle functional and non-functional requirements, and



*Figure 7-20: Architecture of the Online Library Application*



- reuse possibilities, such as reusing parts of similar systems or general software products.

*Example: Online Library*

A simple architecture for the *Online Library* application is shown in Figure 7-20.

---

### **Detailed Design of Classes**

During the previous modeling activities of the analysis and design workflow: user model, conceptual, navigation, presentation and adaptation design a set of classes have been outlined. During the first iterations the classes are usually named, some attributes are defined and sometimes some operations are identified.

During successive iterations the classes are detailed. The activity of detailing a class is performed by the hypermedia engineer, who knows the implementation requirements for classes. This activity includes the following sub-activities: define the class operations, define class attributes, identify aggregation, association, inheritance and dependency of classes, describe its methods, determine its states and establish the requirements relevant to its implementation.

*Example: Online Library*

The description of the design class `AuthorByNameByPublication` is shown in Figure 7-21.

Name: <code>AuthorByNameByPublication</code>
Description: is an index of all authors of a publication ordered by name.
Inherits from class: none
Attributes: <code>authorName</code>
Operations: <code>getAuthor ()</code>
Relationships: <code>Author</code> , <code>PublicationMenu</code>
Diagrams: Navigation Structure Diagram
Special requirements: None
Trace: <code>Author</code>

*Figure 7-21: Class Description*

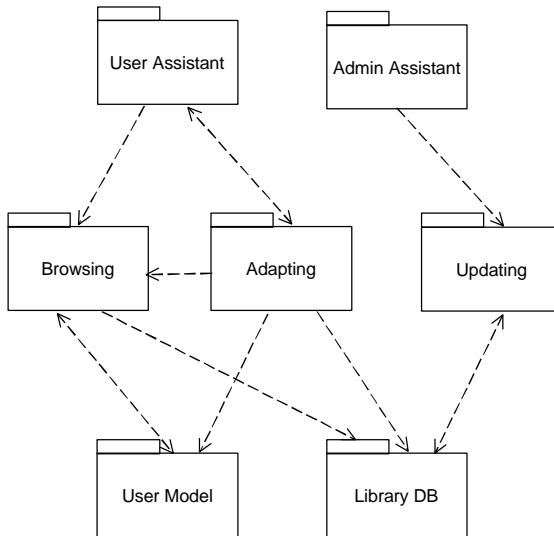
---

### **Definition of Subsystems and Interfaces**

The objective of dividing the system in subsystems is to obtain a set of subsystems as independent as possible, with the aim to be specified and implemented by different developers. This independence allows each subsystem to be implemented by another hypermedia engineer. Subsystems have to provide the right interfaces to fulfil their purposes. The number of dependencies from one subsystem to the others and to the interfaces must be minimised.

#### *Example: Online Library*

In a simplified *Online Library* application the following main subsystems are identified (Figure 7-22).



*Figure 7-22: Subsystems of the Online Library Application*

### **7.3.3**

### **Implementation**

Implementation consists of transformation of the results of the design phase, i.e. design classes, subsystems and interfaces into an implemented system in terms of components, e.g. source code, scripts, executables, etc. Implementation issues, such as conversion of documents, generation of templates and/or dynamic generation of

pages have to be considered in this workflow. Sometimes the inverse process of hyperdocument generation, i.e. the linearisation of hypertext is also part of the implementation process.

Implementation is accomplished in successive activities starting with generation of components, assembling subsystems and interfaces and finishing with software integration. To generate adaptive hypermedia systems the same software tools and languages are used as for the implementation of high interactive and complex non-adaptive hypermedia applications. Figure 7-23 presents the workflow implementation.

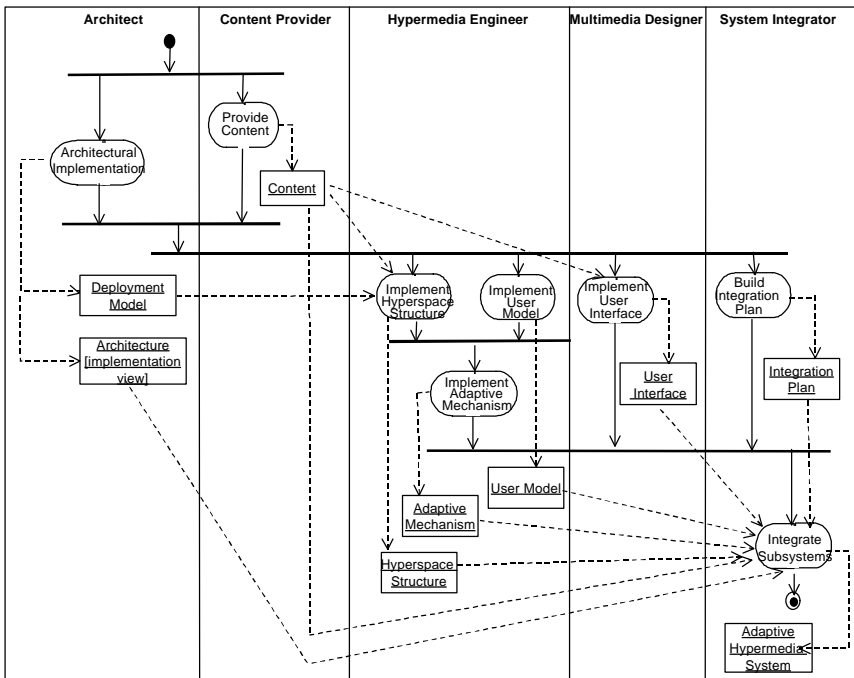


Figure 7-23: Implementation Workflow

The typical components to be produced during implementation of adaptive hypermedia systems are:

- media components, such as text, images, video, audio and animation (they constitute the content of the hypermedia application);
- databases to store the content, i.e. a relational database or a mark-up file system organised into a hierarchical or graph structure;
- structure components required by the hypermedia paradigm, such as menus, indices, guided tours and links;
- user interface components, such as windows, frames, buttons, logos, forms and banners;
- components for the dynamic page generation, such as CGI scripts or Java servlets;
- search engines;

Implementation is the main focus during construction, but implementation is also carried out during inception to create a prototype, during elaboration to create the baseline architecture, during transition to handle late defects and during maintenance to keep a running system.

The implementation activities are performed by five different types of workers: the architect, content provider, multimedia designer, hypermedia engineer and system integrator. They produce the following artifacts: the implementation view of the architecture, the implementation model, the deployment model, the content, the hyperspace structure, the user interface, the integration plan and the adaptive hypermedia system.

Adaptive hypermedia applications require the integration of different implementation techniques, such as static pages specified in markup languages like HTML, DHTML or XML and pages generated dynamically from information stored in databases. To build the bridge between Web and the database different technologies and architectures can currently be used. These include:

- Common Gateway Interface (CGI) that is the oldest method used for implementing Web database gateways. CGI script programs are written in programming languages, such as Perl, C++ or Visual Basic. The advantage of CGI scripts is it is simple to implement and available on all Web servers for free. The disadvantage is that database connection can not be maintained, i.e. each CGI script that queries the database establishes a new connection between the CGI and the DBMS.
- Active Server Pages (ASP) is a Microsoft technology that allows scripts embedded in the HTML code, e.g. VBScript, JScript, PerlScript. ASP are server scripts and support the execution of SQL statements for database access.

- JavaServlets run inside a Java Virtual Mashine on the server. The advantage of JavaServlets approach is its invocations are persistent. They also are portable across operating systems and Web servers.
- ODBC and JDBC are drivers delivered with databases that allow a direct connection between database and application.

In addition, technologies for animation, images, audio and video are used for the development of multimedia applications. No further details about current implementation techniques are given here, since implementation techniques are continually changing and would no longer be up to date in the near future.



### **Artifacts**

During the implementation workflow the artifacts produced are the components of the system. They are related to the content, user model, user interface, adaptation process and hyperspace structure. In order to support the development of these artifacts some models are build or refined. These are the deployment model and the implementation view of the architecture. The integration plan is also an artifact of this workflow as is the adaptive hypermedia system itself, which results from the integration activity.

### **Architecture (implementation view)**

The architecture description contains an architectural view of the implementation model showing the artifacts that are relevant to the architecture. These artifacts are: the subsystems, interfaces and their relationships as well as the key components of the application.

### **Deployment Model**

The deployment model is an object model that describes the physical distribution of the system among computational nodes. It shows the mapping between the software architecture and the system architecture, i.e. the hardware architecture.

### **Integration Plan**

The integration plan describes the sequence of incremental iterations required to construct the adaptive hypermedia system by successive integration of subsystems and components. It details the parts that are to be added in each iteration and the

functionality that is expected to be implemented after each integration. A partial system is constructed in each iteration.

---

## **Content**

The content includes different types of media, such as text, images, audio, video and animation. Each media type requires special treatment and appropriate tools to handle them during creation. Not all content elements are created from scratch, some are taken from other applications, i.e. they are reused or adapted.

Text is usually the predominant media type in hypermedia applications. If a text is not specially created for a hypermedia application, it usually requires some adaptation work. Texts included in adaptive hypermedia applications must be, whenever possible, specific, concrete and precise. Two types of images are used as a complement to the text: bitmap graphics and vector graphics. The major sources of bitmap work are photographs, scanned images, screen dumps and pictures created using special paint programs.

Audio, video and animation are dynamic time based media. The effective use of sound can seldom substitute written information, but has the advantage of attracting the user's attention. Video provides a rich source of documentation. The quality of the video material plays an important role in its usefulness in adaptive teaching applications. Animation adds impact to a presentation and may contribute enormously in a learning process. The storage of this multimedia content requires some kind of organisation, such as multimedia databases or a set of files with a hierarchical directory organisation.

---

## **Hyperspace Structure**

The hypermedia structure is given by a hierarchical HTML file organisation or by a set of templates, which are dynamically filled with data by CGI-scripts at run-time when these pages are requested. Other technologies are possible, such as the use of a database request using PHP. The dynamic generation of the pages ensures the separation of the content from the structure and presentation facilitating maintenance. In addition, menus and indices have to be created to include additional navigational support. The dynamic generation of pages requires the content administration in a database. It is important to improve the quality of the hypermedia structure with the aim of reducing the "lost in hyperspace" problem.

---

### **User Interface**

The user interface is the component of the application with which the user has the most direct contact to. It gives the user the look and feel of the application. Multimedia designer have to find the right balance in terms of size, colours and position of user interface objects in order to avoid cognitive overloading.

---

### **User Model**

The user model is a structure that represents goals, interests, preferences, tasks and knowledge of the user.

---

### **Adaptive Mechanism**

The adaptive mechanism is the implementation of the rules that ensure adaptive content, adaptive navigation and adaptive presentation as well as update of the user model with the information provided by the user behaviour observation. User observation is also responsible for the adaptive mechanism.

---

### **Adaptive Hypermedia System**

The hypermedia system is the sum of the content, user model, adaptation mechanism, hyperspace structure and user interface components that are integrated to provide the full functionality of the system.

The artifacts in the implemented adaptive hypermedia system (implementation model) are components, subsystems and interfaces. According to the functionality of these model elements in the hypermedia system, they are classified into content model elements, structure model elements and presentation model elements. The resulting system is a collection of components, and the implemented subsystems that contain them. Components include both deliverable components, such as executables, and components from which the deliverables are produced, such as source code files.



---

### **Workers**

The implementation of hypermedia applications requires a more heterogeneous group of workers than the development of traditional software. Content providers and multimedia designers are required in addition to the architect, the hypermedia engineer and the system integrator. The component engineer (Unified Process) is

called a hypermedia engineer since his profile contains specific skills required for hypermedia development.

---

### **Architect**

During the implementation phase, the architect is responsible for the integrity of the implementation model and ensures that the components of this implementation model are implemented and integrated. He is also responsible for the mapping of produced components into physical nodes.

The profile of an architect is given in Section 4.1 where the responsibilities of the architect are defined within the context of the requirements capture workflow.

---

### **Content Provider**

The content provider is responsible for providing the raw material that will be included in the adaptive hypermedia application. This material mainly consists of text and some images, video, audio and/or animations. He digitises them and chunks them into appropriate pieces of information as well as providing alternative content for different user profiles.

The profile of the content provider is characterised by the following skills:

- experience with the legacy applications which are the source of data, and
- text composer experience.

---

### **Multimedia Designer**

The multimedia designer is responsible for the production of all multimedia elements, such as windows, buttons, logos, images, etc. that the application requires, as well as for the creation or reworking of existing multimedia content, etc.

The profile of the multimedia designer includes the following skills:

- knowledge of tools for the creation and the manipulation of images, video, audio and animations,
- experience in the design, integration and synchronisation of multimedia elements, and
- creativity skills.



---

### **Hypermedia Engineer**

The hypermedia engineer defines and maintains the source code of one or several components which implement the structure of the hypermedia application. He assures that these components implement the correct functionality, i.e. the functionality specified by the design classes and the use cases realisation. Usually the hypermedia engineer of a complex application is responsible for all the components of one subsystem.

The profile of the hypermedia engineer is outlined in Section 4.2 together with his activities in the analysis and design workflow.

---

### **System Integrator**

System integration cannot be the responsibility of the hypermedia engineers. Instead, a system integrator is assigned to plan the sequence of the system's that are build in each iteration and the successive integration of the subsystems. A characteristic of hypermedia applications is the integration of text, images and time-dependent media, such as video, audio and animations as well as the integration of components developed using different technologies. The result of the system integrator's planning activities is the integration plan.

The profile of the system integrator includes the following skills:

- good communication skills,
- general domain knowledge,
- experience in system integration,
- knowledge of multimedia synchronisation, and
- experience in the implementation language that has been chosen.



---

### **Activities**

The main goal of the implementation workflow is to obtain an implemented system. To achieve this goal the system architect outlines the key components of the implementation model. Based on the implementation model and the content provided (by the content provider) the hypermedia engineer implements the hyperspace structure and the system integrator builds the integration plan. The multimedia designer implements the user interface using the implementation model and the content. The objective of the system integration activity is to combine content, hyperspace structure and user interface as well as user model and adaptation rules. In addition the aim is to test the functionality of each component.

---

### **Architecture Implementation**

The objective of architectural implementation is to outline the implementation model and its architecture and to identify the components that are relevant to the architecture, such as executable components. A further objective is to map components to nodes of the network configuration.

---

### **Provide Content**

An important activity during the implementation process is to capture or generate the underlying data for the content and convert it into an appropriate format. The resources needed for this activity are often underestimated. The conversion process is required for example when the data to be used in the hypermedia application comes from legacy applications, such as paper-based documentation, manuals or image, audio and video archives.

This process involves several activities, such as:

- obtaining the raw data from legacy records or by new recording of information,
- scanning text and images,
- digitising images and video,
- capturing audio,
- applying character recognition to scanned text,
- adjusting quality, size, colours of images,
- chunking the data into appropriate pieces of information, etc.

The main problem is the difficulty in automating some of these activities, such as the adjustment of quality of images and the correction of scanned text.

---

### **Implement Hyperspace Structure**

Once the suitable data was obtained, the data needs to be organised according to the structure defined in the navigation and presentation design activities. The implementation of the hyperspace structure thus involves the following activities:

- implementation of templates,
- introduction of linking mechanisms,

- storage and organisation of the data, and
- creation of mechanisms for automatic page generation.

---

### **Implement User Interface**

The implementation of the user interface is one of the greatest consumers of implementation time in the development process of adaptive hypermedia applications. During this activity a user interface is implemented based on the presentation design. The logical design determines which user interface elements are needed while the physical design provides a first approach to the visual aspects and distribution of these user interface elements.

The activity *implement user interface* is performed by the multimedia designer, who produces a user interface prototype during the first iterations and a final version in the last one.

---

### **Implement User Model**

The implementation of the user model consists of the definition of the schema of the user model tables or database, the specification of attributes and possible range of values for these attributes as well as the implementation of stereotyped profiles.

---

### **Implement Adaptive Mechanism**

The implementation of the adaptive mechanism consists of the codification of the global and/or local rules and the methods that trigger and execute these rules for adaptation and user model updating. These rules can be stored in a database or can be implemented as Java classes, for example.

---

### **Build Integration Plan**

The objective of this activity is to plan the new components and/or functionality to be included in the next iteration. The system that is built during an iteration should not include too many new components or improvements. It should be easy to test and should allow for easy documentation of the changes. The following steps can be used as a guide to constructing an integration plan:

- decide which use case to include,
- identify the subsystem or the design classes that participate in the use case realisation,

- identify the implementation subsystem or components in the implementation model which can be traced to the subsystem or design classes of the second step,
- plan to include the implementation of the subsystem or the components in the subsequent build.

---

### **Integrate Subsystems**

Following the plan elaborated during the activity “build integration plan” the components are included in the current system, compiled and linked. The new build is then ready for the testing process performed during the supporting workflow.

## **7.4 Project Management**

The development life cycle of a software application requires the support of a project management workflow. There exists an abundance of literature on this topic. The objective of this section is to outline the steps of the project management process and to show that there are a few aspects related to adaptation that have to be taken into account and that entail additional risks in comparison to non-adaptive systems.

All projects have a technical aspect and a management aspect. The purpose of project management is to control, trace and evaluate the project. Management and technical aspects of a project have to fit together. A series of milestones are therefore set. A milestone is a concrete defined or determinable event with precisely determined artifacts to be delivered. Milestones can be combined with reviews. There are few tools, which can be effectively used in project management. They are mostly used for documentation purposes, such as GANTT charts are.

The project management workflow consists of risk management, iteration planning and iteration evaluation workflows as depicted in Figure 7-1.

### **7.4.1 Risk Management**

Introducing new technologies increases the potential risks of a project. Technologies, such as servers, components and languages for the implementation of hypermedia applications are new or improving permanently. Adaptive hypermedia applications are still at an experimental development stage. It is

therefore important to identify risks for the project at an early iteration of the inception phase, to determine how critical these risks are and to define actions to mitigate them. The risk management workflow is presented in Figure 7-24.

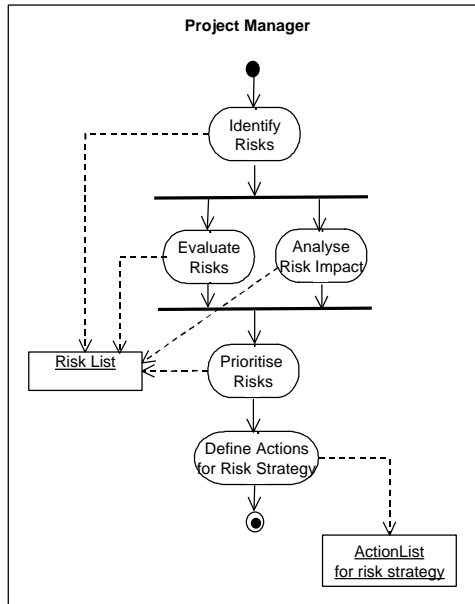


Figure 7-24: Risk Management Workflow

Risks are whatever prevents the success of the project. The success of a project can be defined as the meeting of all requirements and constraints specified in the project. Risks can be defined in the software development process more precisely as a variable that, within its normal distribution, can take a value that endangers or reduces the success of the project.

Risk management is a procedure the goal of which is to ensure awareness of risks as a prerequisite for managing them. Activities, which must be carried out, are the identification of the risks at an early stage, risk evaluation, analysis of risk impact and definition of a risk strategy to handle the risks. The four main strategies according to Boehm (1991) are:

- *Risk prevention*, i.e. the reorganisation of the project so that it cannot be affected by the risk.
- *Risk transfer*, i.e. changes in the project so that someone or something else bears the risk (customer, vendor or bank).
- *Risk acceptance*, i.e. monitoring the risk symptoms without changes in the project elaborating a contingency plan just in case the risk emerges.
- *Risk mitigation*, i.e. taking actions to reduce the probability of impact on the project.

The Euromethod framework (1996) as well as the Information Services Procurement Library – ISPL – (1999) are methodologies that include guidelines for risk management applicable to general software development. The ISP for Web Engineering book treats, amongst other aspects, risk management in the acquisition and planning process of Web applications (Koch & Helmerich, 2000).



---

## **Artifacts**

The artifacts of the risk management workflow are the risk list and the list of actions necessary to handle these risks as shown in Figure 7-24. The risk list is produced at the very beginning of the project and completed with the results of the risk evaluation and an analysis of the risk impact. Both the risk list and action list must be continuously updated along the whole project.

---

## **Risk List**

A risk list is a sorted list of known, open risks to the project, sorted in decreasing order of importance according to an associated probability of occurrence or the probability of impact on the project.

For the evaluation of risks it is important to distinguish between direct and indirect risks. A direct risk is one over which the project has some degree of control; indirect risks are ones which cannot be controlled.

---

## **Action List (for Risk Strategy)**

The action list is a list of appropriate actions that have been selected for each risk. Selection also implies strategy selection, i.e. a decision as to whether the risk can be prevented, transferred, accepted or mitigated. Possible actions for the different risks strategies are suggested by risk management methods.

**Workers**

Although a risk expert can be consulted, the project manager is the person responsible for the overall project development including risk management.

**Project Manager**

The project manager is responsible for the communication, planning and evaluation activities of the project. This includes communication and interactions between customers, the developing team and users. The project manager allocates resources, establishes priorities and co-ordinates review activities to ensure the quality of the project results.

The profile of the project manager includes the following skills:

- experience in planning and co-ordination activities,
- knowledge of the development process,
- knowledge of the business and hypermedia domain,
- knowledge of risk management,
- practice in budgeting and resource planning,
- experience in project documentation, and
- practice in review and versioning planning.

**Activities**

The activities performed by the project manager in the risk management workflow are: the identification and evaluation of risks, analysis of risk impact on the project, prioritisation of risks and the definition of actions for a strategy to handle the risks, i.e. avoid, transfer, accept or mitigate them (see Figure 7-24).

**Identify Risks**

To identify risks the project manager utilises check lists such as the general list of questions presented by the Rational Unified Process (2000), the list of specific situational factors for risks presented in ISP Web Engineering (Koch & Helmerich, 2000) or he organises a risk workshop. To give an idea of what risks may arise a few general risk factors are listed:

- commitment to the project,
- size of project in relation to other projects of the organisation,

- team configuration (work in other projects),
- availability of domain experts,
- dependency on other projects,
- measurement of results, and
- schedule.

A list of a few specific factors that may produce risks in an adaptive hypermedia development project is given here:

- use of innovative Web technologies,
- complexity of multimedia content, navigation structure and/or presentation,
- experience of the workers with the implementation of adaptive mechanisms, and
- difficulty of the user monitoring process.

*Example: Online Library*

The list of risks of the *Online Library* project consists of: a team, that has not worked together before, workers who have no experience in the development of adaptive systems and a project which plans to use new technologies to ensure a good performance.

---

### **Evaluate Risks**

The evaluation of risks consists of describing the risks and giving an estimation of how complex or how uncertain they are. For example selecting one of the following values: high, middle or low complexity or uncertainty.

---

### **Analyse Risk Impact**

As for risk evaluation, the impact the risk has on the success of the project can be classified by selecting the value high, middle or low or by applying a percentage. The list can be extended to a table with columns for evaluation and impact values.



---

### **Prioritise Risks**

The activity of risk prioritisation involves in the assignment of priorities to risks. The list can be ordered according to priority set or a priority number can be included in the table.

*Example: Online Library*

The following table shows the risk and impact evaluation as well as the priorities assigned for the treatment of the risks.

Risk	Evaluation	Impact	Priority
team configuration	low	low	3
no experience in development of adaptive hypermedia systems	high	high	1
use of new technologies	middle	high	2

*Table 7-25: Risks for the Online Library Project*

---

### **Define Actions for Risk Strategy**

Risk strategy actions are obtained from lists in the relevant literature compiled specifically for this purpose and then adapted to the project by the project manager. The adjustment of the list is based on the project manager's experience in similar projects, for example.

*Example: Online Library*

The following table shows risks, and actions to mitigate these risks.

Risk	Action
team configuration	organise workshops
no experience in development of AHS	plan additional time for development establish strict review process
use of new technologies	offer training plan tool evaluation plan additional test iterations

*Table 7-26: Actions to Mitigate Risks for the Online Library Project*

## 7.4.2 Iteration Planning

In the first iteration a project plan is elaborated that consists of an iteration plan for each basic phase: inception, elaboration, construction, transition and maintenance. During each iteration the plan for the next phase is adjusted, possibly including planning for additional iterations where necessary. This iteration planning is done within a risk management framework that allows the analysis of the critical success factors involved and actions, which are planned to reduce these risks. Special attention must be paid to the milestones to be included in the definition of the iteration plan for the design and implementation of the adaptive mechanisms.

The objectives have to be mapped onto a schedule, milestones must be established and measurements need to be selected in order to achieve the goals set. Costs and a schedule are established at this stage for the requirements capture, design, implementation, quality control and maintenance.

The definition of the initial and final states, cost, milestones and deliverables help to produce an iteration plan and a delivery plan.

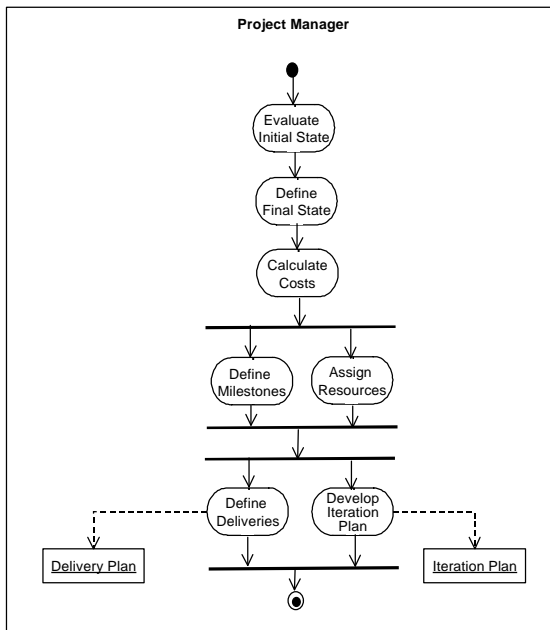


Figure 7-27: Iteration Planning Workflow



---

## **Artifacts**

Although a set of activities are performed during the iteration planning workflow, the results of these activities only support the production of documents for two main results of the workflow: the delivery plan and the iteration plan. The iteration planning workflow is shown in Figure 7-27.

---

### **Delivery Plan**

A delivery plan is a commitment of results to be delivered fulfilling a set of conditions relating to schedule, budget and resources conditions.

---

### **Iteration Plan**

The iteration plan is a fine-grained plan that includes a time-sequence set of activities and task, assigned to resources and containing task dependencies for the iteration. Detailed diagrams are used to show timelines, intermediate milestones, testing starts, beta releases, demos, etc. for the iteration. An initial iteration plan has to be elaborated at the beginning of the project, but in each phase the iteration plan for the next phase is reworked.

The contents of an iteration plan comprises:

- the current status of the project,
- a list of scenarios or use cases that must be completed by the end of the iteration,
- a list of risks that must be addressed by the end of the iteration,
- a list of changes that must be incorporated in the product,
- a list of activities to perform for the validation, verification and/or testing, and
- a date for the review of the deliveries.



---

## **Workers**

The project manager is responsible for the activities performed during the iteration planning. Project manager skills are described in the risk management workflow subsection.



---

## **Activities**

The activities of this workflow are: evaluation of initial state, definition of final state, calculation of costs, definition of milestones, assignment of resources, determination of deliveries and determination of review schedule (see Figure 7-27).

---

### **Evaluate Initial State**

Initial and final states are the starting point for a clear definition of the content, structure, layout and adaptive features. Typical initial states are a non-adaptive hypermedia system or a non computer-based environment.

---

### **Define Final State**

The definition of the final state is the description of the vision of the software system to be built. It is not only important for the project management; it is also the starting point of the requirements capture.

#### *Example: Online Library*

The idea is to build an online library that gives a personal support to the user. This *Online Library* will offer information about publications to registered and anonymous users. The publication information comprises journals, books and proceedings...

---

### **Calculate Costs**

It can be difficult to estimate the efforts required for software development, quality assurance and project management without the appropriate experience. In the case of adaptive hypermedia systems this risk is even greater than for hypermedia development. Tools and methods used in the development process of general software may be used for estimation, such as Function Points (Dekkers, 1999), COCOMO II (Boehm, Abts, Brown, Chulani, Clark & Horowitz, 2000), but there also exists recent works of De Bra (2000) and Olsina (2000), which have analysed hypermedia development and proposed new metrics.

It is important to document the resulting cost estimate for the project in order to use such results in future cost calculations for other projects.

---

### **Define Milestones**

The definition of milestones is established on the basis of a set of tasks or activities that are defined for the project, the risks that are identified and the overall project plan.

---

### **Assign Resources**

The resources needed for the iteration – human, financial, equipment, etc. have to be assigned for the next iteration otherwise it is not possible to produce the results planned in the delivery plan and scheduled in the iteration plan.

---

### **Define Deliveries**

For each milestone a report, at least, documenting the status of development must be delivered. Usually, the deliveries of a milestone are a composite of documents, models and software components.

A report of the requirements capture describing the use cases, the description of the architecture, the models of the analysis and design process, the use case realisation, a glossary, the implementation packages, a first beta release, the release 1.0, successive etc.

*Example: Online Library*

The deliveries of the analysis and design step are the detailed description of the use cases (Figure 7-4), the use case model (Figure 7-5), the glossary (Figure 7-6), the conceptual model (Figure 7-8), the user model (Figure 7-9) the navigation model (Figure 7-8), the presentation model (Figures 7-12 and 7-18), the adaptation model (Figure 7-19), the implemented components and the final integrated version.

---

### **Determine Review Schedule**

The review schedule is usually part of the milestones that are defined for the project. It establishes which milestone require an external review of the deliveries. The review dates complete the iteration plan.

*Example: Online Library*

An iteration plan for an elaboration iteration in the *Online Library* sample consists of the following items:

- status of the project: rough use case description and first glossary finished,
- to do in this iteration: detailed description of use cases belonging to the “update” package,
- workshop organisation: risk of the missing knowledge of the team members of each other must be solved through three workshops planned in this iteration,
- update of artifacts of previous iterations: new version of glossary,
- delivery date: 20.9.2000,
- review date: 20.10.2000.

### 7.4.3 Iteration Evaluation

To benefit from the iterative process, the project has to evaluate the results of an iteration at the end of each iteration and each phase. The project manager is responsible for this evaluation. The objectives of the iteration evaluation are:

- to adjust and to refine the plan of the next iteration, according to the lessons learned in the current iteration,
- to modify the process, adapt tools, extend training or change steps suggested by the experience of this iteration, and
- to review progress against the iteration and project plan.

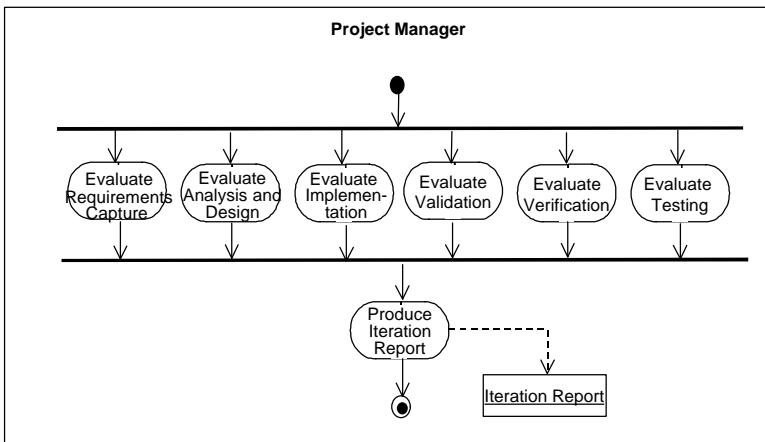


Figure 7-28: Iteration Evaluation Workflow

The iteration report has to give answers to questions, such as:

- Is work proceeding within budget and schedule? or
- Is the quality of the deliveries in line with the requirements specification?

◆ Artifacts

The artifact produced in the iteration evaluation workflow is the iteration report as it is shown in Figure 7-28.

Iteration report

The iteration report summarises all the activities and results of the iteration. It's main objective, however, is the critical evaluation of the iteration. The report focuses on problems that emerged during the iteration and solutions that proved to be effective as well as a description of situations that have not been properly resolved. The documentation of the lessons learned in the iteration is particularly important. In a new software development field such as adaptive hypermedia such material forms the basis of best practice documentation.

The iteration report has to fulfil the layout and level of granularity that is defined for all documents in the project. Documentation can be standardised through templates to be used or by guidelines. A useful place to start definitions such as guidelines or templates are the ISO and IEEE software engineering standards (IEEE, 1987/1993 and IEEE, 1988/1993).

◆ Workers

The project manager is responsible for the evaluation of the iteration. See the description above in the risk management workflow subsection.

◆ Activities

A set of activities are performed for the evaluation in each iteration. Not all of the activities mentioned below are relevant to each iteration; the appropriate activities are chosen by the project manager, who produces the iteration report based on these evaluations. The list of evaluation activities include:

- evaluate requirements capture,
- evaluate analysis and design,

- evaluate implementation,
- evaluate validation,
- evaluate verification, and
- evaluation testing.

These activities are not described in detail here. They do not differ from the evaluation activities of the development process of other kinds of software. The evaluation is a critical step in an iteration and should not be skipped. If iteration assessment is not done properly, many of the benefits of an iterative approach will be lost. The results of the evaluation activities are processed during the report production activity (see Figure 7-28).

---

### **Produce Iteration Report**

As a result of the evaluation an iteration report is produced by the project manager. This document is not updated. It should include the following information at least: to what evaluation it applies, what is affected or influenced by the document, a list of reference documents, models or software that were used as a basis for evaluation, evaluation criteria established by the iteration plan for functionality, performance and quality, references to test results, external events, success of the iteration, problem areas that need to be reworked in upcoming iterations. Focus should be put on adaptive and innovative aspects of the projects.

*Example: Online Library*

The following is the iteration report of an elaboration phase in the *Online Library* sample:

<p><b>Iteration Report</b> Elaboration Phase: Iteration 3 – Navigation Specification</p> <ol style="list-style-type: none"> <li><b>1. Objectives</b> Completeness of navigation model.</li> <li><b>2. Scope</b> Compare textual requirements description, use case model and navigation model.</li> <li><b>3. Documentation used</b> Requirements description Use case model Navigation model</li> <li><b>4. Objectives reached in the iteration</b> Yes, the navigation model is completed.</li> <li><b>5. Accordance with schedule plan</b> One week delay.</li> <li><b>6. Results relative to evaluation criteria</b></li> </ol>
---



<p>The navigation model includes the browsing functionality specified by the use cases (see 8).</p> <p><b>7. External changes</b> Additional comparison with a non-adaptive librarian system X was performed.</p> <p><b>8. Rework required</b> Keywords should also apply to authors, not only for publications and articles.</p>
---

*Figure 7-29: Sample Iteration Report for the Online Library Application*

## 7.5

## Quality Management

Although quality management is treated as a separate workflow that supports the development process, it does not mean that quality checks are performed after implementation has been completed. Quality management comprise validation, verification and testing. The quality management activities should begin early and are integrated in the process as shown in Figure 7-1.

- Validation checks whether the result really is what the customer actually wants.
- Verification checks whether the results agree with the specification.
- Testing checks whether the produced software is correct, i.e. it runs without failures.

The most important aspect of quality assurance is the attitude of the workers towards it. They must be conscious that quality management takes time and its costs must be included in the budget. Validation, verification and testing activities are part of the life cycle process and must be planned in the same way as design and implementation activities.

Hypermedia applications as with other software applications require quality assurance activities. Validation plays an important role, even more so if the customer is new to the electronic business world. Hypermedia applications require tests for multimedia components, to prove the quality of the navigation structure and special test to detect problems in the navigation space, such as the existence of dangling links. Some tests, like the detection of dangling links must be performed regularly, especially during maintenance.

The user plays a central role in adaptive hypermedia applications. These applications therefore require, validation, verification and testing of an appropriate adaptation to the individual user or user group. Test cases for potential user groups

have to be defined and it is very effective to have the test be performed by a heterogeneous test group.

The quality of Web sites can be assessed by methods, such as Web-site quality Evaluation Method (QEM). It is prescriptive and descriptive method based on the evaluation and comparison quality characteristics and attributes in different phases of the hypermedia cycle (Olsina, Godoy, Lafuente & Rossi, 1999).

### 7.5.1 Validation

Validation checks whether the result really is what the customer actually wants, i.e. it ensures the customer’s satisfaction. Boehm (1981) describes validation with the question: “Are we building the right product?”. The definition given by the IEEE Standard Glossary of Software Engineering Terminology (1983) says: validation is the process by which software conformance to the requirements specification is tested. The use of use cases and prototypes are good techniques to facilitate a validation process.

In Web applications users’s needs and typical user behaviour must be analysed to learn what the user wants. It is difficult to marry the user requirements and the customer’s vision. Hypermedia systems must be implemented in such a way that it is possible to incorporate new technologies and new design alternatives in a near future.

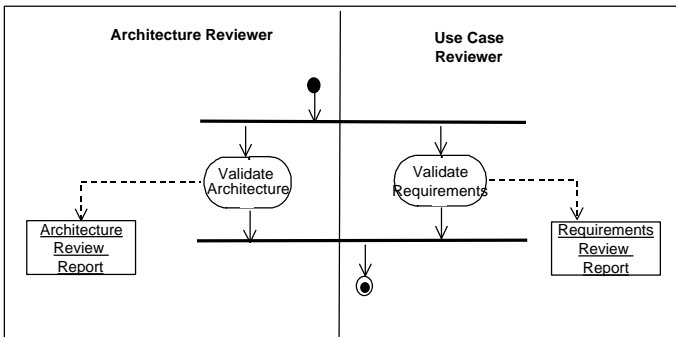


Figure 6-30: Validation Workflow



---

## **Artifacts**

Two artifacts are produced by the architect and requirements reviewer in this workflow: the architecture review report and the requirements review report respectively, as shown in the validation workflow of Figure 7-30.

---

### **Architecture Review Report**

The architecture review report includes the software architecture document with an annotation indicating if it is accepted or rejected. In the latter case a list of changes to be performed has to be included.

---

### **Requirements Review Report**

The requirements review report includes the following documents with an annotation indicating if they are approved, or rejected with indications as to what reworking is necessary:

- use case model,
- use cases,
- supplementary specifications, and
- glossary.



---

## **Workers**

The validation activities, i.e. the architecture and requirements validation are performed by the architecture reviewer and the use case reviewer.

---

### **Architecture Reviewer**

The architecture reviewer plans and conducts the formal reviews of the software architecture in general. The profile of the architecture reviewer has to include the same skills as that of the architect, focusing more on the technical issues and critical analysis of the architecture model.

---

### **Use Case Reviewer**

The use case reviewer plans and conducts the formal review of the use case model. The profile of the use case reviewer has to include the following skills:

- knowledge of the business and hypermedia domain,
- general knowledge of user modeling and adaptive systems, and
- UML use case modeling techniques.



---

## **Activities**

Activities of this workflow are the validation of the architecture and requirements (see Figure 7-30). Different techniques can be used for the validation:

- *walk-throughs* – These consist of a detailed going through of all the documentation, and a comparison of results, such as models, with the requirements description,
- *audits* – These consist of a comparison of results against a predefined checklist or a checklist elaborated at the beginning of an iteration, or
- *prototyping* – This is the implementation of results in order to check if the functionality and user interface design conforms with the requirements.

---

## **Validate Architecture**

The validation of the architecture mainly consists of:

- error detection in the architecture model,
- finding requirements that are missing in the architecture design,
- assessing the observation of the user behaviour,
- assessing the adaptive functionality, and
- avoiding architecture over-design.

---

## **Validate Requirements**

To validate use cases lists of checkpoints that are part of the Rational Unified Process (2000) can be used. These lists are not included here, although a few important checkpoints for hypermedia applications are mentioned and some specific checkpoints for adaptive hypermedia have been added.

- Do all actors be identified, which is particularly difficult in case of users of Web applications?
- Are all navigation requirements of the variety of users considered?
- Do customers, users and designers alike understand the names and descriptions of the use cases?

- Do use cases reflect the adaptive needs of the actors?
- Do use cases include the capture of user behaviour?

*Example: Online Library*

Validation of the requirements in the sample application is carried out using the walk-through technique. The requirements review report includes, for example the following feedback for the use case *select visited articles*: add a reference to the use case that handles the setting of marks by the user.

## 7.5.2 Verification

Boehm (1981) describes verification with the question: “Are we building the product right?”. The IEEE Standard Glossary of Software Engineering Terminology (1983) defines verification as the process of determining whether the product conforms to the requirements specified in previous phases and it serves as a good basis for the implementation. All the models developed must therefore be checked to ensure that they satisfy the functional, non-functional and supplementary requirements.

Walk-throughs, audits and prototyping are techniques that are also used for verification. The verification must follow the verification plan included in the project plan and in the iteration plans. One verification of the design model is required per iteration in the elaboration and construction phases. Transition and maintenance phases require reviews if these models are changed.

## ◆ Artifacts

The following review reports are produced as results of the verification activities: reports on the architecture, on the models and on the design classes as shown in the verification workflow of Figure 7-31.

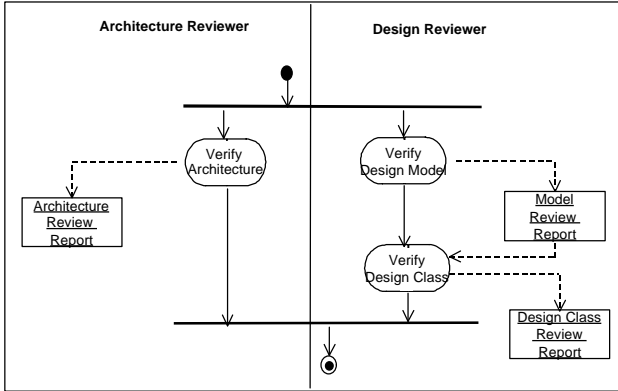
### Architecture Review Report

See Validation subsection.

---

## **Model Review Report**

The model review report is elaborated to list all defects detected in the models and includes, whenever possible, suggestions for correction and changes.



*Figure 7-31: Verification Workflow*

---

## **Design Class Review Report**

The design class review report provides the classification approved or rejected for each class of each subsystem or package. Defects, corrections and changes are added to the document.




---

## **Workers**

The architecture reviewer is responsible for the architecture model review process and the design reviewer is responsible for the design models and the design classes.

---

## **Architecture Reviewer**

See Validation subsection.

---

## **Design Reviewer**

He produces a review plan for the design that supports a systematic verification of all design classes and models against the requirements specification. The design reviewer profile has to include the following skills:

- general domain knowledge,
- general knowledge of user modeling and adaptive systems, and
- expertise in UML modeling techniques.

◆ 

---

 **Activities**

The verification workflow consists of activities related to a thorough inspection of the design models in order to reduce as much as possible changes in the design decision during implementation workflows as much as possible. These activities are the architecture, design model and design class verification (see Figure 7-31).

---

 **Verify Architecture**

See Validation subsection. The verification focuses on the comparison between architecture model and requirements specification.

---

 **Verify Design Model**

The main purpose of the design model verification activity is:

- to ensure that the conceptual model is representative of the domain,
- to verify that the navigation structure is appropriate for the functionality of the application,
- to compare the adaptive model with the adaptive functionality specification,
- to detect problems in the presentation model, such as overloading or missing links, and
- to ensure that the behaviour is allocated to the correct model elements,

Checklists provided by the Rational Unified Process, for example, can be used for design model verification and specialised for adaptive hypermedia systems.

*Example: Online Library*

Verification of the models built for the *Online Library* is for example performed by:

- verifying that the conceptual model is complete, i.e. library, publications, publishers, authors, articles and keywords (with their attributes, methods and associations) are all concepts that are needed for an online library;
- verifying that for the navigation structure model all possible searches of articles, authors, publications are covered, e.g. articles by publications, articles already visited, relevant new articles, etc.;
- verifying that the each template includes the anchors necessary to guarantee the navigation specified in the navigation structure model. (This verification is not necessary if the generation of the templates is totally automated).

---

### **Verify Design Class**

The design class verification must review each model element, i.e. each class, interface and subsystem. In the case of subsystems, this means ensuring that the subsystem realises that the behaviour specified in the interfaces has been allocated to one or more contained classes or subsystems. For classes, this means that the description of each operation is sufficiently defined so as to be implemented unambiguously. General checkpoints provided by software development processes can be used for verification.

### 7.5.3

### **Testing**

Testing is a process of checking the correctness of the implementation results by running a system. Testing primarily checks whether the produced software is correct, i.e. it runs without failures.

The goal is to test functionality, performance, usability, compatibility and reliability of the implemented system in general and to perform specific tests for adaptive hypermedia systems, such as:

- *an orientation test*. This analyses how many steps a user has to go through to get some piece of information,
- *dangling links test*. This checks if there is a target node for each link,
- *test for unreachable nodes*. This checks whether for each node there is a path starting from a root node of the application to the node,



- *appropriate adaptation test.* This checks that there is an adaptation functionality defined for each attribute value of a user profile attribute.

It is important to test and analyse the impact of adaptive components on user satisfaction levels, such as it is done by Strachan, Anderson, Sneeby & Evans (1997).



## Artifacts

Testing artifacts are the test plan, test cases, test procedures, test components and test reports as shown in the testing workflow of Figure 7-32.

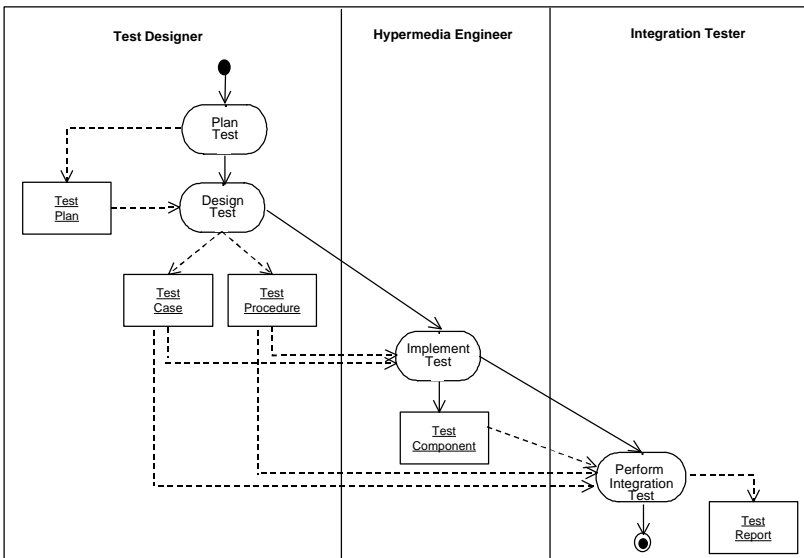


Figure 7-32: Testing Workflow

## Test Plan

The test plan contains information about the purpose and goals of testing within the project as well as the strategies to be used to perform testing and the resources

needed. It is important to communicate the intention of the testing activities through the test plan.

---

### **Test Case**

A test case is a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to run a particular program path.

---

### **Test Procedure**

A test procedure is a set of detailed instructions for the set-up, execution, and evaluation of results for a given test case (or set of test cases) and a method used to compare the expected and actual results. The results of a test procedure can be evaluated by simple visual comparison or by a non-visual method.

---

### **Test Component**

The test component is the code, which automates the execution of a test procedure (or portion of a test procedure). Test components may be created using a test automation tool, or programmed using a programming language.

---

### **Test Report**

The test report contains results from the testing workflow giving feedback about changes and improvements that must be performed in the next iterations.



---

### **Workers**

The testing activities are performed by the test designer, hypermedia engineer and integration tester.

---

### **Test Designer**

The test designer is responsible for the elaboration of the test plan together with the project manager, for the design of the test cases and test procedure. The profile of the test designer should include the following skills:

- domain knowledge,
- knowledge of the adaptive hypermedia system or application-under-test,
- knowledge of testing and test automation tools, and
- diagnostic and problem solving skills.

---

### **Hypermedia Engineer**

The hypermedia engineer is responsible for implementing the test cases based on the test design. See the analysis and design workflow for the description of the profile of a hypermedia engineer.

---

### **Integration Tester**

The integration tester is responsible for executing the integration tests and producing the final test report. His profile skills are similar to the skills required for a test designer plus programming knowledge.

### ◆ **Activities**

The activities of testing are: plan of tests, design and implementation of tests and execution of these integrated tests (see Figure 7-32).

---

### **Plan Test**

The plan test activity consists of collecting test-planning information and creating an appropriated test plan based on this information. For the plan an acceptable test sequence has to be defined based on risks, requirements and test resources.

---

### **Design Test**

The following steps must be performed for the definition of test cases and test procedures:

- definition of test conditions,
- identification of use cases to test focusing on adaptive functionality,
- preparation of the appropriate test data, and
- specification of the expected test results.

Test cases may be reused or adapted from an iteration to the next one.

---

### **Implement Test**

To implement test means almost always writing or reusing code, so called test scripts. The test environment has to be set-up including data, hardware, software, tools, etc. Implemented tests are then executed by the integration tester. Usability has to be tested, too.

*Example: Online Library*

The *Online Library* application is tested by a group of users, who test the appropriateness of the adaptive functionality. In addition, tests for navigation path length, a comparison of menu-based and map-based approaches may be performed.

---

### **Perform Integration Test**

Test procedures are executed during this activity manually or automatically. The activity consists of:

1. setting up the environment,
2. executing the test scripts,
3. evaluating results,
4. determining the next action:
  - if results are as expected, no action is necessary;
  - if results are unexpected, the cause of the problem must be determined and resolved.

*“...the power of a student model does not lie in its fidelity but in the differences it indicates”*

*John Self,*

*Computer-Aided Learning and Instruction  
in Science and Engineering,  
July 1996.*

## **8** Development of SmexWeb Applications – A Case Study

The development process of UWE defined in the previous chapter is an approach to a systematic development of adaptive hypermedia applications. The use of a software engineering approach instead of an ad hoc implementation improves the quality of adaptive and user-model-based hypermedia applications, reduces error-prone implementation and facilitates documentation and maintenance. The proposed methodology is validated with several applications: two important case studies – the EBNF-application and the Taxonomy application – and other smaller non-adaptive applications. These EBNF and the Taxonomy are both, adaptive Web-based applications for which implementation the SmexWeb framework was used. EBNF stands for *Enhanced Backus-Naur Formalism*, a grammar-like technique used for describing the syntax of programming languages. The EBNF-application is an exercising system for students visiting an introductory course in computer science. The Taxonomy application is an exercising system that was developed for botany students. It supports students to train and test their knowledge in subjects, such as generative morphology and naming of plants.

The SmexWeb framework and the EBNF-application were developed by Albrecht (1998) and by Tiller (1998). The Taxonomy application was implemented by Pezdirc (1999)<sup>11</sup>.

---

<sup>11</sup> These works have been developed within the scope of their diploma thesis under the author's advise at the Institut für Informatik, Ludwig-Maximilians-Universität München, Germany.

In this chapter the development process of the EBNF-application is described in detail while the Taxonomy application only is outlined, as the development process is very similar. The first section provides an overview of the SmexWeb framework. Section 2 describes the activities performed and the results obtained in each particular phase following the UWE approach. In Section 3 the design models performed for this application are presented. The fourth section outlines the Taxonomy-application. Section 5 gives some conclusions to the learning process supported by SmexWeb.

## 8.1 The SmexWeb Framework

SmexWeb (**S**tudent **m**odelled **e**xercising on the **W**eb) is a framework for implementing learning systems on the Web. In this chapter the user is also called the learner or student and the developer is called the author (of the learning material).

SmexWeb consists of a collection of abstract and concrete classes that permits the development of teaching applications through instantiation, i.e. SmexWeb applications (Albrecht, Koch & Tiller, 1999). The framework is generic enough to create courses in any domain. It is a modular approach that allows for the reuse of domain independent components, so that the author need only define content, structure and presentation of the lesson as well as the adaptation rules, but not how the adaptation mechanism is implemented.

SmexWeb applications, similarly to other adaptive Web applications observe each learner's behaviour and builds a user model for her. Based on this user model the application dynamically adapts the material to be taught to the learner's characteristics and needs. SmexWeb's user model is general enough to include cognitive, knowledge and general abilities of the students. SmexWeb implements a higher amount of interaction between the system and the learner than common Web-based systems have achieved so far. Compared to similar systems, the number of human-machine interactions that can be observed by the system has been increased in the SmexWeb framework, hence allowing better a estimation of the user's needs and making learning more efficient. This is accomplished by using an extra communication channel between the learner's computer and the server, in addition to the stateless Hypertext Transfer Protocol (HTTP) underlying the WWW rules (Albrecht, 1998).

SmexWeb framework supports adaptive content – adjusting the content of the pages to the learner's knowledge and preferences – as well as adaptive navigation support like link removing, link annotation and link ordering. The author of an

adaptive hypermedia application thus supports the learning process of the student, suggesting links and annotating them individually. So far, the user still controls the way through the course material in an active, self-regulated and goal-oriented acquisition process.

Yet sometimes in an application it is necessary for the system to take control (Vassileva & Watson, 1996), contrary to the hypermedia paradigm, where the locus of control always lies in the hands of the user. SmexWeb applications allow the system to take control over the process of navigation offering to the learner some help or guidance, when she seems lost, remains inactive or her behaviour corresponds to a pattern behaviour (Albrecht, 1998 & Tiller, 1998). This is a new concept in adaptive hypermedia systems – called passive navigation – used in the SmexWeb applications for the first time. If the assumptions about a user and her inactivity indicate that a different page to the one currently displayed is more appropriate, the system navigates to that page.

Passive navigation widens the classical navigation paradigm of hypermedia systems as it transfers part of the control from the user to the system, which in pure hypermedia applications lies entirely on the user side. In this respect the learning environment resembles more closely a classical teacher/learner situation. The user's feeling of being lost and subsequent frustration and demotivation is avoided.

The SmexWeb framework provides a basis for building an authoring tool for adaptive hypermedia applications.

## **8.1.1** The Architecture

The components that make up the SmexWeb framework are the Web client, a HTTP server and the SmexWeb server. The focus of the description is the SmexWeb server, a collection of reusable abstract and concrete classes written in the Java<sup>12</sup> programming language. The subsystems of the SmexWeb server framework correspond to the typical Intelligent Teaching Systems (ITS) components as described in (Beck, Stern & Haugsjaa, 1996) although the Domain Knowledge and Expert Model components are substituted by the Hyperspace component. Those modules of the framework that have to be instantiated for a concrete application are outlined in the subsections below.

The SmexWeb architecture resembles a classical client/server concept that is built upon the WWW as shown in Figure 8-1. Most information exchanged between

---

<sup>12</sup> Java 1.1

client (learner) and server (SmexWeb) via the Internet is transported using HTTP, which is the native protocol of the WWW. A standard Web server is used to transfer all the content material to the learner and to pass data on to the SmexWeb server application and back to the client. The content presented to the learner is composed of standard HTML pages, which may contain any media type a Web browser is capable of displaying. Learners have to identify themselves in order to use a SmexWeb application. The identification is accomplished by using HTTP access authentication. The adaptation of the pages presented to the learner is performed by JavaScript<sup>13</sup> programs embedded in the HTML code.

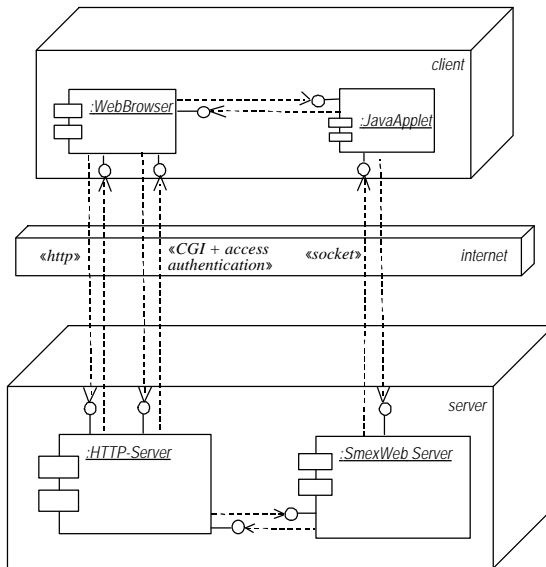


Figure 8-1 Architecture of SmexWeb

A SmexWeb application to be utilised by a learner requires only a system that has Internet access, a standard Web browser, and the SmexWeb URL address. The utilisation of standard products and technologies and the fact that there is no need to install another software on the client side, provide the advantages of proven functionality and efficiency of common products, and greater platform independence.

<sup>13</sup> JavaScript 1.0



The SmexWeb server consists of the following components: a Tutor, a User Model, a Hyperspace and Communication subsystems (see Figure 8-2). These subsystems implement the functionality described in the Munich Reference Model for adaptive hypermedia systems in Chapter 4. The Hyperspace subsystem contains the concepts of the domain model and the adaptation rules. The Communication subsystem implements the adaptation model. The functionality of the Run Time Layer can be found in the Tutor.

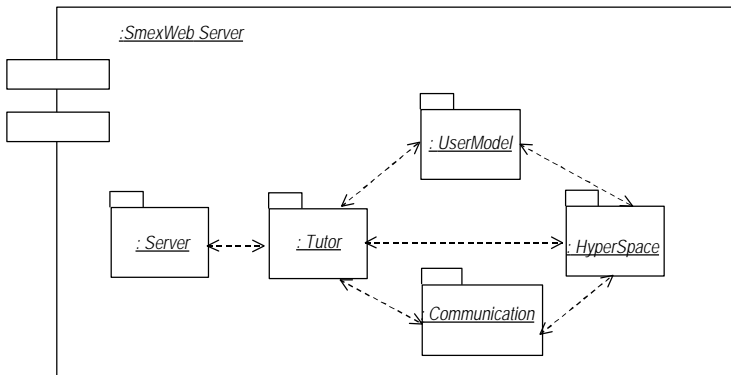


Figure 8-2: The SmexWeb Server

A typical learner's request is processed as follows:

1. The HTTP server passes the request on to the SmexWeb server.
2. SmexWeb creates a component called Tutor for every learner logged on to the system. The Tutor is responsible for keeping all the session's information about the learner and for processing her requests.
3. The tutor analyses the information package sent by the learner's browser and incorporates relevant information about the learner into the user model.
4. Once the user model is up to date, the Communication subsystem generates a small JavaScript program which includes control information that is necessary in order to constructing an answer to the request. The answer is based on the data contained in the Hyperspace and adapted according to the current values of the UserModel.
5. This program is sent back to the browser via the HTTP server.

6. The client browser executes the JavaScript, which retrieves all the media necessary for assembling the pages of information and displays them according to the user model.

In this way the workload is distributed between server and client side. Furthermore, as the content adaptation takes place on the client side, the amount of data transferred over the network can be minimised. For the learner this means a faster response time of the system, which is an important factor in preventing loss of motivation.

### 8.1.2

### The Tutor

The Tutor subsystem implements an improved Dexter Run-Time Layer. It is the heart of the server application. Following the metaphor of a private teacher a Tutor object keeps track of all information bound to a single learner. Several users may use one SmexWeb application at the same time, each one with her own Tutor, and with the Tutors working in parallel like task manager of TANGOW (Carro, Pulido & Rodriguez, 1999). The Tutor assists the learner during the whole session; it observes the learner working with the material, builds a model of her/his preferences and capabilities, and gives her assignments and answers to questions according to this model.

For an accurate representation of the learner's characteristics, the framework allows a higher degree of interaction and observation than common Web-based systems. The WWW paradigm only transmits data when a user follows a link. This allows for an easy implementation of fill-out questionnaire tests in a learning environment. Many Web-based teaching environments work in this way. (Kay & Kummerfield, 1994; Weber & Specht 1997; Nakabayashi et al. 1997) A private teacher, however will usually judge a learner's performance not only by the result of a test but also by how she achieves the solution. The teacher might want to give the learner some hints along the way.

The WWW paradigm is extended in SmexWeb by an additional communication channel between client and server to allow for a higher degree of interaction. Java applets running in the learner's Web-browser communicate directly with the SmexWeb server as shown in Figure 8-1. This extension gives the author of a teaching system the ability to collect more information about the learner, which in turn enables better adaptation of the material to the learner's needs.

The Tutor subsystem can be directly used without modification for every SmexWeb application, since it does not contain any information related to the domain, the user or the adaptation rules.

### 8.1.3

### The Communication

The task of the Communication subsystem is to produce the answer to a given request from the client. Information from UserModel and Hyperspace is collected and integrated to form the answer. As mentioned above, the SmexWeb generates control information, while the client browser has the task of assembling content pages.

The author of a concrete SmexWeb application may use the Communication subsystem without modification, as it provides a default handling. However, he may change it if desired. The Communication subsystem contains all the heuristics to translate the information for adaptation provided by the Hyperspace module into concrete adaptive navigation techniques.

The SmexWeb framework guarantees navigation consistency. This is a difficult issue in adaptive hypertext systems. A user may want to walk back a path she has followed for a couple of pages. The material on a page is adapted to the user according to the user model state at display time. Navigating back to a previously read page might become confusing, as the user model might have changed over time and the page might look completely different from what the learner expects. The SmexWeb framework provides a built in mechanism to avoid frustration arising in this situation: it maintains a personal *history* including the already *visited pages* and the *user model states*. When a learner revisits a page using the back button, the page automatically is displayed as it was when visited the last time.

### 8.1.4

### The UserModel

The user model maintains the assumptions the system has about the user (Kobsa & Pohl, 1995). Topics to be covered when building an adaptive system are: what information about the user is to be modeled, how is the model organised and how is the information about the user acquired.

As discussed in Chapter 3 the user model is divided into three sub-models: learner's domain knowledge, the user's profile including interests and knowledge in other domains and cognitive characteristics such as preferred media and learning strategies. The latter two are likely to remain constant for a longer period and may be reused in different applications.

The framework proposes the use of a short-term as well as a long-term user model. When a user log-outs, her user model together with the above mentioned history is stored in a long-term user model so that it can be restored at the beginning of the

next session. This way the learner can continue a new session where he finished in the previous one and the system is set to the previous state.

The SmexWeb framework supports the author in creating a user model for a SmexWeb application by providing a basic user model and a number of sub-models as well as access and manipulation mechanisms of these sub-models. The current version of the framework contains three sub-models, data structure of which is based on key-value pairs. The author, however, is free to create sub-models of any kind and complexity reusing only some parts of the UserModel subsystem. Different values of a user model may depend on each other. To maintain consistency within the model, SmexWeb provides a mechanism of so-called consistency rules. Interdependent parts of the model may be connected by those rules and constraints may be stated. How these constraints are formulated, as simple conditions or as more complex calculations, is up to the author.

Acquiring the knowledge about a user is achieved using so-called acquisition rules based on condition-action pairs. Any screened interactions as well as all the information about a user are available to formulate conditions and subsequent actions. By instantiating the condition-action pairs, the author has a straightforward way of implementing his strategies to estimate the learner's characteristics in a procedural, hence intuitive way.

### 8.1.5

### The Hyperspace

The Hyperspace subsystem consists of a set of nodes and links. Each node in the structure is linked to one physical page.

SmexWeb clearly separates the structure of a hypertext document from the physical pages presented to the learner. This separation is suggested by many well-known hypertext reference models, such as the Dexter Model (Halasz, 1994). The author has the freedom to take different steps in the creation process of a course, which are untied from each other: The representation of the mental model and the suggestion of individual ways through it are a matter of the courseware structure. The author builds a graph of links and nodes by instantiating the respective classes of the framework. She/he declares the importance of the links and nodes for the individual learner. Different link representations and annotations may be specified, dependent upon different user model states.

The presentation and adaptation of material is covered during the page creation step. Each page is a downloadable file referenced by the hypertext structure in the framework. The idea of writing an adaptive page is based on page fragments, as suggested by Kay and Kummerfield (1994). SmexWeb widens this concept by

allowing any kind of displayable items. An author can explain concepts in a variety of ways and assign each fragment to certain users. According to the state of the user model, the most appropriate alternatives are presented to the user. Examples are alternative versions of a single word or a whole video that might only be presented to learners preferring this kind of media.

## **8.2 Development of the EBNF-Application**

The SmexWeb framework was used as basis for the EBNF-application developed for students visiting an introductory course in computer science. The application offers the possibility to practice EBNF, a formal grammar used for programming language description taught on the course. The SmexWeb application on EBNF is intended for the heterogeneous group of students on this course. These are students of various subjects whose minor is computer science from the first up to the eighth semester, together with elderly people. Neither experience in working with computers and interactive systems nor the ability to understand and apply abstract formalisms has to be taken as prerequisite.

The idea of the EBNF-application is the following: The student has to respond to an initial interview to allow the system to build first assumptions about her knowledge and abilities. Then a first exercise is presented to the learner and the opportunity to choose alternative exercises from other categories. In the next steps, the student interactively solves exercises from different categories. The system supports the student activities in different ways, depending on the estimated cognitive abilities of the user. Finally, the student is asked to apply the acquired knowledge and skills to solve a similar exercise without support. The student may request pages containing context sensitive help and explanation of the domain concepts at any time. Under certain circumstances, the system may present these pages to the user by means of passive navigation. The system always suggests one exercise but offers other exercises with different degree of difficulty to the learners, e.g. to whom already knows EBNF well.

The activities that have been performed in each development phase to produce the EBNF-application are briefly explained in the following sub-sections. These activities follow the guidelines of the UWE methodology described in Chapter 7.

## 8.2.1

## Inception Phase (Iteration 1)

The goal of the first iteration in the development process – this is part of the inception phase – is to determine the feasibility of the project. A project plan is thus prepared on the basis of a risk analysis and requirements elicitation.

The idea of the exercising session of the EBNF case study was to develop an application to offer exercises and definitions to a very heterogeneous group of learners (different background and interests, ages from 20 to 70, etc.). The SmexWeb-based application should select exercises with adequate degrees of difficulties, described in a formal or pragmatic way and with or without examples for each individual learner.

One of the main objectives is to identify and characterise the learners as accurately as possible. This is a precondition for adapting the exercise material so that it satisfies individual needs. The characteristics to be modeled are determined based on the given decisions about the topic and the potential group of users. Classifying potential users before designing the user model may prevent modeling characteristics that are too generic. Consequently, the developer can keep the user model small, the system works more efficiently and the learner's needs are met more effectively.

The creation of the user model is often a creative and non-deterministic process relying heavily on the pedagogical experience of the developer and his knowledge of existing psychological models. To elicit the information required in this phase a set of interviews were performed with students, tutors and teachers (Albrecht, 1998). Students were asked to fill in a questionnaire including queries related to their person (age and genre), their studies (major and minor subjects), their lessons (theory and exercise sessions that are attended), general computer knowledge (frequency of use, knowledge about operating systems, programming languages, and tools), missing information and difficulties in traditional lessons in the introductory course.

The current user model of the EBNF-course comprises three sub-models as introduced in Chapter 3. Here they are called the domain, navigation and individual models. Values for the domain model represent the learner's knowledge about the topic of the course; students might have different degrees of EBNF knowledge when they use the SmexWeb course to practice. Background knowledge, derived, for instance, from the learner's major subject, is included in the user model to support learning by drawing analogies. The most important attribute of the navigation model captures the learner's navigation experience. The individual model represents learning preferences, for instance with brief or extended explanations, more formal or more pragmatic descriptions.

Table 8-3 summarises the most relevant activities performed during the first iteration that is part of the inception phase. The table distinguishes which workflow they belong to and details the main results of the workflow.

	Workflows	Activities	Results
Project Management	Risk Management	<p>Identify risks</p> <p>Evaluate risks and analyse risk impact</p> <p>Define risk strategy</p>	<p>Risk list: inappropriate group of users, complexity of the user model, performance problems, etc.</p> <p>Impact: inadequate testing, inadequate adaptation, no user acceptance</p> <p>Strategy: find testing group, test technologies (java applets, server). Simple user model is a high risk: no strategy</p>
	Iteration Planning	<p>Evaluate initial state</p> <p>Define final state</p> <p>Define milestones and deliveries</p> <p>Determine Schedule Review</p>	<p>Initial state: none</p> <p>Final state: running EBNF exercising session based on the SmexWeb framework prototype</p> <p>Milestones are:</p> <ul style="list-style-type: none"> <li>- Definition of goal and requirements,</li> <li>- Architecture and design models,</li> <li>- Refinement of the design models,</li> <li>- Construction in two or three iterations (to be defined in the second iteration),</li> <li>- Test and adjustments.</li> </ul> <p>Iteration 1 (goal and requirements): 15 days Iteration 2 (architecture and design models): 1 month Iteration 3 (refinement of design models): 1 month Iteration 4-5/6 (construction): 4 month Iteration 6 or 7 (test and adjustment): 15 days</p>
	Iteration Evaluation	<p>Evaluate requirements capture</p> <p>Evaluate validation</p>	<p>Iteration report:</p> <ul style="list-style-type: none"> <li>- proof of completeness of use case models,</li> <li>- sufficiency of procedure to capture the requirements (interviews),</li> <li>- sufficiency of review process</li> </ul>

	Workflows	Activities	Results
Development Process	Requirements Capture	Identify users Elicit information and navigational needs Find actors and use cases Prioritise use cases Capture common vocabulary	Preparation of questionnaire for interviews Interviews with students and tutors  Outlined use case model  First architecture model  Glossary
	Analysis and Design	User Model Design	Definition of user attributes, classification of attributes in three groups: domain, navigation and individual.
	Implementation	Implement UI	Evaluation of applets performance
Quality Management	Validation	Validate requirements	Interviews with teachers and tutors
	Verification		
	Testing	Plan and design tests	Definition and co-ordination of test activities for the lessons of the next semester

*Table 8-3: Activities performed during the Inception Phase for the Development of the EBNF-Application*

## 8.2.2 Elaboration Phase (Iterations 2 and 3)

The focus of these two iterations is to produce analysis and design models, which are the basis for the construction process in iterations 4 and 5. The activities of the elaboration phase are centred on the design of the conceptual model, user modeling, structuring of the hyperspace and authoring of the domain pages. A clear separation of the development of content, navigation structure and presentation can be observed in this stage and the resulting artifacts. The main activities performed during these iterations are: the support of a refinement of the use cases, test planning, implementation of a user interface prototype, actions to mitigate risks and planning of the sequel iterations.

Important milestones during this phase are the user model initialisation and update mechanisms. The initialisation of the EBNF-application user model is performed



using the user's answers to the initial interview. This interview comprises questions on:

- how frequently the learner uses Web applications,
- which kind of WWW services she uses,
- her major and minor study subject,
- her age,
- why the student is visiting the introductory course in computer science,
- which kind of explanations or examples she prefers (e.g. formal or pragmatic),
- how does she evaluates her programming experience, and
- a test on basic EBNF knowledge.

The hypertext structure of the EBNF-application reflects the non-linear representation of the domain knowledge and provides different ways through it. The developer has to use pedagogical knowledge as well as his domain knowledge to structure the document and to offer different individual ways of navigating through the hypertext, based on the current user model.



Figure 8-4: Frameset proposed for by the SmexWeb Framework

The developer of an adaptive SmexWeb-based application has great freedom when designing a user interface based on the notion of logical windows. SmexWeb supports a multiple-window technique as well as using frames in a window. The

EBNF-application uses the standard window partition proposed by SmexWeb. It consists of four frames, as shown in Figure 8-4. Two of these frames are then subdivided into another two and three frames, respectively. The metaphor that was selected is a desk covered with learning material, with a clear separation of exercising and reference material.

1. The *exercising area* (upper left) contains the pages with the material the student is currently working on, e.g. interactive pages for solving exercises or answering test questions.
2. The *learning area* (upper right) presents pages related to the current page in the exercising area, and provides context sensitive help, definitions, examples or a navigation map.
3. The *global navigation area* (lower left) comprises navigation facilities that are always available to the learner, such as access to help, navigation map, back button (i.e. displaying the previously presented page), forward or an end application button.
4. The *thematic navigation area* (lower right) provides the user with the navigation facilities for the different exercise categories. These anchors are ordered according to the estimated individual importance and annotated using icons and greyscale levels. In addition, navigation to the reference material and to the task definition is provided.

Table 8-5 summarises the main activities performed during the second and third iterations that are part of the elaboration phase. The table distinguishes which workflow they belong to and details the main results of the workflow.

	Workflows	Activities	Results
Project Management	Risk Management	Define actions for risk strategy	Risk mitigation by performance tests and by page generation at client-side using JavaScript
	Iteration Planning	Define milestones and deliveries	Updated iteration and delivery plan: construction phase has to be performed in 2 iterations (4 and 5) Documentation plan
	Iteration Evaluation	Evaluate requirements capture, analysis and design  Evaluate validation and verification  Produce iteration report	Iteration report: - Proof of consistency of design models - Sufficiency of review process - Quality of validation and verification

	Workflows	Activities	Results
Development Process	Requirements Capture	Elicit additional requirements Detail use cases Structure use cases  Elicit adaptation capabilities Elicit UI needs Prototype UI  Capture common vocabulary	Software requirements: Apache server <sup>14</sup> , JavaApplets <sup>15</sup> , HTML <sup>16</sup> , JavaScript, Java Use case detailed description Structured use case model with packages: initialisation, exercising, assistance and adaptation Initial interview Definition of adaptation rules  User interface description (sketches) Prototype (look and feel), mock ups, storyboards Glossary is updated
	Analysis and Design	Conceptual design	Conceptual model (see 8.3.2)
		User model design	User model (see 8.3.3)
		Navigation design	Navigation model (see 8.3.4)
Presentation design		Presentation model (see 8.3.5)	
Adaptation design	Adaptation model (see 8.3.6)		
Implementation	Provide content	Content: exercises enunciation, exercises solutions, help texts, reference material, exercises implemented as applets	
Quality Management	Validation	Validate requirements	Use Cases review report
	Verification	Verify design model	Review of design models
	Testing		

*Table 8-5: Activities performed during the Elaboration Phase for the Development of the EBNF-Application*

<sup>14</sup> <http://www.apache.org>

<sup>15</sup> JavaApplets 1.0

<sup>16</sup> HTML Version 3.2

Iteration 2 focuses on the elaboration of a detailed description of the use cases, user model and the conceptual model. The main focus of iteration 3 is the development of the navigation, presentation and adaptation model.

### 8.2.3 Construction Phase (Iteration 4 and 5)

The construction of the EBNF-application is performed in two iterations (4 and 5). Iteration 4 focuses on the implementation of the lessons content, the navigation structure and the test procedure. Iteration 5 focuses on the implementation of the adaptive functionality, which includes the capture of the user behaviour and the adaptation of content and navigation to the current values of the user model.

The initial values of all sub-models are assigned on the basis of a learner's answers to the interview questions. From there on, values will be changed dynamically according to the system's observation of the learner's actions while navigating or solving an exercise. The tests of the EBNF-application with integrated adaptive functionality are performed in iteration 5. During the construction phase a detailed plan for the transition and maintenance has to be elaborated.

Table 8-6 summarises the main activities performed in the construction phase. It includes activities of both the fourth and fifth iterations.

	Workflows	Activities	Results
Project Management	Risk Management	Evaluate risks and analyse risk impact  Define actions for risk strategy	Risk of technology dependency (Versions of Java, browsers, etc)  Action: maintenance plan
	Iteration Planning	Define milestones and deliveries	Updated iteration and delivery plan: transition plan for regular use in introductory courses  Documentation plan
	Iteration Evaluation	Evaluate requirements capture, analysis and design, implementation  Evaluate validation, verification and testing Produce iteration report	Iteration report: - Proof of consistency of design models - Sufficiency of tests plan - Quality of testing

	Workflows	Activities	Results
Development Process	Requirements Capture	Detail use cases Elicit adaptation capabilities Capture common vocabulary	Use case detailed description is completed Refinement of adaptation rules Glossary is updated
	Analysis and Design	User model design Conceptual design Navigation design Presentation design Adaptation design  Design classes	All design models are refined  Classes for the user model, domain and adaptation are designed
	Implementation	Provide content Implement hyperspace structure Implement user model Implement adaptive mechanism Implement UI  Build integration plan  Integrate subsystems	Content pages: exercise enunciation, help texts, reference texts Coding of hyperspace components  Coding of user model components Adaptation rules and mechanisms Templates for the user interface  Version of iteration 4 without adaptive functionality Version of iteration 5 includes user model and adaptation Integration in iteration 5
Quality Management	Validation	Validate requirements	Requirements review report
	Verification	Verify design model	Design models review report
	Testing	Implement test  Perform integration test	Test of components  Test of application without and with adaptive functionality

*Table 8-6: Activities performed during the Construction Phase for the Development of the EBNF-Application*

Figure 8-7 to Figure 8-9 show pages of the user interface of the implemented EBNF-application. Figures 8-7 and 8-8 depict the same concept (the initial exercise) to the learner. The cognitive preferences and abilities of a learner represented in the user model, dictate the appropriate form of presentation provided. A formal way of explaining the problem is shown in Figure 8-7, while a more pragmatic formulation appears in Figure 8-8.

The screenshot shows a web browser window titled "Metzpage: SmexWeb". The main content area is divided into three sections:

- Die Aufgabe:** A yellow box at the top right containing a flowchart. It starts with "Die Aufgabe", which leads to "Spielen" (containing two empty boxes). From "Spielen", an arrow labeled "drehen Lösen" points to "Lösen" (also containing two empty boxes). From "Lösen", an arrow points to "Anwenden".
- EBNF-Regeln für "Gebirge" sollen gefunden werden:** A list of five rules:
  - Ein Gebirge besteht aus Schrägstrichen "/" und "T" und aus flachen Strichen.
  - Ein Gebirge endet auf dem gleichen Niveau, auf dem es begonnen hat.
  - Ein Gebirge hängt links auf dem Niveau null an und darf nie unter dieses Niveau fallen.
  - Für jeden Strich mit dem das Gebirge wächst, also "T" muß es an irgend einer späteren Stelle wieder fallen, also "/".
  - Ein Gebirge muß das Niveau null mindestens einmal verlassen.
  - Gebirge müssen zusammenhängen.
- Einige Beispielsgebirge:** A link to examples.
- Folgende Navigationsmöglichkeiten finden Sie für gewöhnlich im Frame rechts unten:**
  - Die Auswahlpunkte oberhalb der Linke bieten verschiedene Möglichkeiten, zu einer alternativen Seite oder zum nächsten Schritt zu gelangen. SmexWeb versucht immer, Ihnen die für Sie beste Möglichkeit als oberste anzubieten.
  - Unterhalb der Linke finden Sie Erläuterungen, und Beispiele zur momentanen Seite. Diese Zusatzinformationen werden immer im Frame rechts oben dargestellt.

At the bottom of the page, there are navigation buttons: "Ende", "Zurück", "Vorwärts", "Hilfe", and "Karte". On the right side, there are links for "EBNF-Definition" and "EBNF-Gewichte". Below the main content, there are three smiley icons with labels: "Gebirge basteln", "Gebirge erkennen", and "direkt Lösen".

Figure 8-7: Formal Description of the Exercise's Task of the EBNF-Application

Figure 8-9 shows an interactive exercise in the EBNF-application implemented with an applet in the left hand part of the window. Hence, not only final solutions to the exercise, but also important steps the user follows to reach the solution are transmitted to the SmexWeb server and are used in the observation process of the user's behaviour.

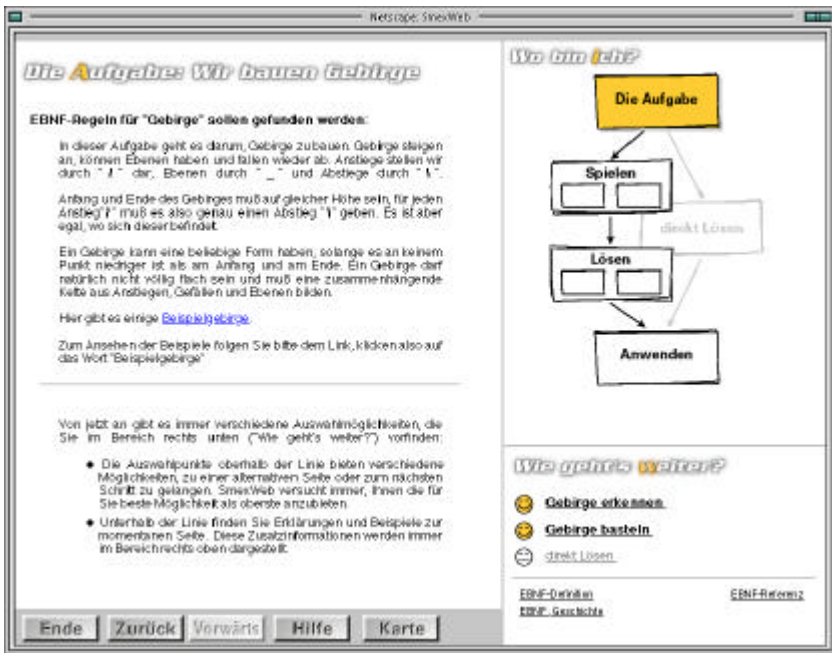


Figure 8-8: Pragmatic Description of the Exercise's Task of the EBNF-Application

## 8.2.4

## Transition Phase (Iteration 6)

The focus of the transition phase is on the testing of the EBNF-application, the corrections of defects discovered during testing and the deployment of the application. Table 8-10 summarises the main activities performed during the sixth iteration. Note that activities detailed on the lower part of the table are performed before activities detailed in the first rows.

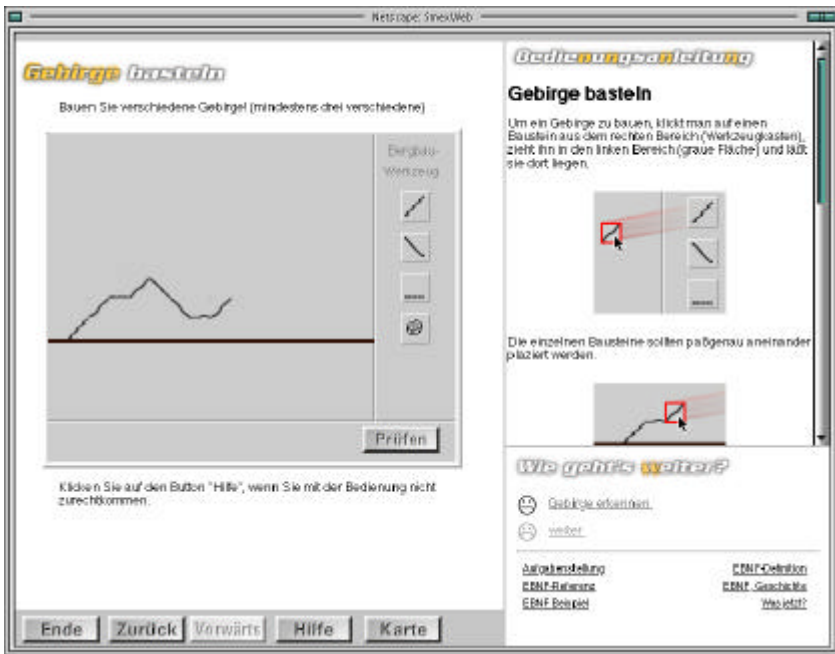


Figure 8-9: An Interactive Exercise of the EBNF-Application

The EBNF-application was tested twice by students visiting an introductory course in computer science<sup>17</sup>. Both tests were performed by a group of about 15 students aged between 22 and 66. They used the system for an average of about 40 minutes. After they had answered the initial questionnaire about general and topic-dependent themes presented to them by the system, the students were able to use the EBNF-application without further explanations.

The students were not explicitly told that the system would try to adapt to their needs. The answers to the questions allowed the system to obtain initial values for the user model. One of the goals of SmexWeb is to facilitate the development of applications that not require any specific guidance or introduction to the system.

<sup>17</sup> "Einführung in die Informatik für Studierende anderer Fachbereiche", Institut für Informatik, Ludwig-Maximilians-Universität München, WS 97/98 (Wirsing/Frühwirth) and WS 98/99 (Wirsing).



	Workflows	Activities	Results
Project Management	Risk Management	Evaluate risks and analyse risk impact  Define actions for risk strategy	Risks: insufficient time for reworking, insufficient or inadequate tests  Strategy: Assign more resources for tests and corrections
	Iteration Planning	Define milestones and deliveries  Determine review schedule	Plan for corrections and tests  System is deployed after finalisation of tests and corrections
	Iteration Evaluation	Evaluate implementation  Evaluate testing Produce iteration report	Iteration report: - Corrections done - Changes in documentation - Test results: students feedback  Online documentation is delivered
Development Process	Requirements Capture	Structure use cases  Capture common vocabulary	Use cas model is updated according to implemented version  Glossary is updated
	Analysis and Design	Conceptual design User model design Navigation design Presentation design Adaptation design	Design models are updated according to implemented version
	Implementation	Implement hyperspace structure Implement user model Implement adaptive mechanism Implement UI	Defects discovered in implemented version are corrected
Quality Management	Validation		Review
	Verification		Review
	Testing	Perform integration tests	Integrated version is tested by developers and group of learners

*Table 8-10: Activities performed during the Transition Phase for the Development of EBNF-Application*

An off-line interview was performed after each student finished the EBNF exercising session. Most of the students gave very positive feedback and were highly motivated during the use of the system. Neither performance difficulties nor system problems arose during the tests sessions. The EBNF-application worked in a highly efficient and stable way. System response time was very short compared to network latency and data transfer time.

## 8.2.5 Maintenance Phase (Iteration >= 7)

Table 8-11 summarises the main activities performed during maintenance.

	Workflows	Activities	Results
Project Management	Risk Management	Define actions for risk strategy	Test modifications before deployment of changed version
	Iteration Planning	Define milestones and deliveries	Plan changes and tests New version is delivered
	Iteration Evaluation	Evaluate implementation Evaluate testing Produce iteration report	Iteration report: - Changes done - Changes in documentation - Test results
Development Process	Requirements Capture	Elicit information requirements Elicit navigation needs Elicit adaptation capabilities Elicit UI needs  Elicit additional requirements	Use cases detail description are changed to show - Changes in contents - Modification in navigation structure - Modifications of adaptation rules - Changes in presentation needs - Changes in user descriptions List of non-functional requirements is changed
	Analysis and Design	Conceptual design User model design Navigation design Presentation design Adaptation design	Design models are modified to reflect changes in requirements
	Implementation	Provide content Implement hyperspace structure Implement user model Implement adaptive mechanism Implement UI	Components are modified to maintain the application updated

	Workflows	Activities	Results
Quality Management	Validation	Validate requirements	Review documentation
	Verification	Verify design model	Review documentation
	Testing	Perform integration tests	Changed version is tested

*Table 8-11: Activities performed during the Maintenance Phase for the Development of the EBNF-Application*

The main focus of the maintenance phase is on performing the changes that are necessary to ensure a stable, efficient and updated EBNF-application. Many iterations may be performed until the application is no longer used or replaced by another one.

## 8.3 Analysis and Design Models for the EBNF-Application

The following sections focus on the models that represent the results of the requirements capture workflow, and the analysis and design workflow. These models are: the use case, conceptual, user, navigation, presentation and adaptation model. They are constructed according to the methodology detailed in Chapter 6. A summary of the UML Profile used for the graphical notation is the subject contained in the Appendix.

Some details of the EBNF-application are omitted in the diagrams of the models presented in the following sections so as not to overload these diagrams. These details require additional views of the models that are not included in this chapter.

### 8.3.1 Use Case Model

The main result of the requirements capture is the use case model. In the requirements capture workflow the following actors and use cases are identified for the EBNF-application.

- Actors: Learner and SmexWeb-Tutor. A tutor is an instance of a SmexWeb component that is assigned to each learner who registers herself for a SmexWeb application.

- Use cases: read introduction, respond interview, read session’s goal, read exercise task, read example, read reference material, solve exercise, evaluate solution, ask for help, follow link, observe navigation, present exercise, update user model and execute adaptation rule.

These use cases can be grouped into the following packages: initialisation, exercising, assistance and adaptation. The use cases of the first three packages are triggered by the actor Learner; the use cases of the last package are initiated by the actor SmexWeb-Tutor. The distribution of the use cases is as follows:

- Initialisation: read introduction, respond interview, and read session’s goal.
- Exercising: read exercise task, solve exercise, and follow link.
- Assistance: read example, read reference material, and ask for help.
- Adaptation: evaluate solution, present page, update user model, observe navigation, and execute adaptation rules.

Note that some use cases express the functionality provided by the SmexWeb framework. These use cases include activities that are application independent. Hence, the EBNF-application need not realise the use cases update user model,

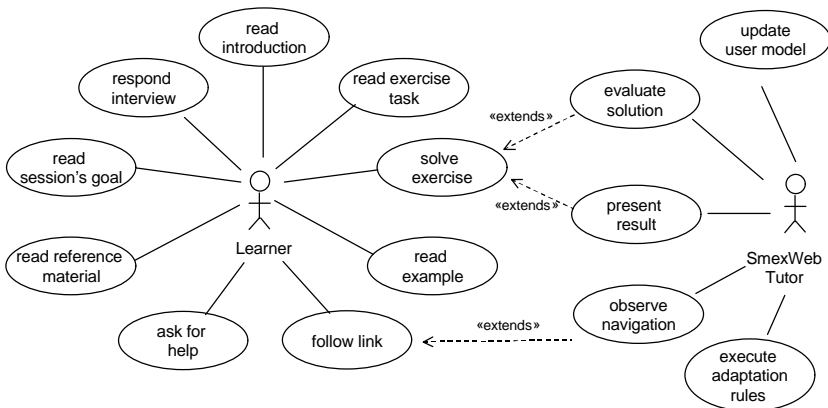


Figure 8-12: Use Case Model of the EBNF-Application

observe navigation or execute adaptation rules. Figure 8-12 shows the actors and the use cases that are identified for the EBNF-application.

### 8.3.2

### Conceptual Model

The conceptual model is a model of the problem domain, i.e. an EBNF exercising session. It includes all the concepts that are relevant to the EBNF-application. The main objective is to capture the domain semantics. Navigation and presentation aspects are treated separately in the elaboration of navigation and presentation models. Adaptive content is represented by the variant compartments of the classes Exercise, Task, Example and ReferenceMaterial.

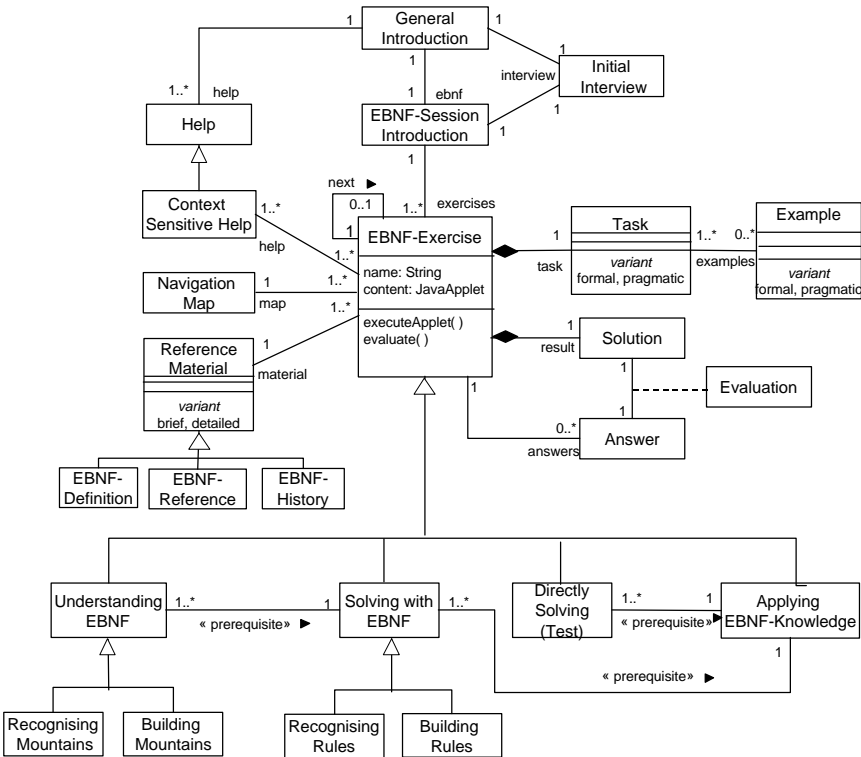


Figure 8-13: Conceptual Model of the EBNF-Application

Exercises are classified into the following categories in the attempt to simulate a classroom session:

- Exercises to help understand EBNF (also called playing exercises). This category includes two subcategories, i.e. exercises for:
  - recognition of mountains, and
  - construction of mountains.
- Exercises to help on using EBNF. This category also includes two subcategories, i.e. exercises for:
  - recognition of EBNF rules, and
  - construction of EBNF rules.
- Exercises that are part of a multiple choice test. These can be solved directly.
- Exercises to apply the EBNF knowledge acquired while solving exercises in the other categories.

The conceptual model is represented as a UML class diagram based on the methodology presented in Section 6.2. Figure 8-13 shows the conceptual model of the EBNF-application.

### 8.3.3

### User Model

The interviews performed as part of the requirements capture workflow in the inception phase have identified a heterogeneous group of students. There are learners with or without computer experience or Web experience and with or without knowledge of the EBNF subject. They have preferences for formal or pragmatic explanations and require different levels of help.

The topic of EBNF requires that the learner's cognitive abilities to formalise and to think in abstract terms be represented. The information gained in this early interviewing process is used to define the structure of the user model of the application. It comprises three sub-models, that is the domain model, the background knowledge model and the individual model (i.e. cognitive preferences). Values for the domain model represent the learner's knowledge about EBNF. The background knowledge model captures her experience with Web-applications. The individual model represents learning preferences, such as brief or detailed explanations, exercise's tasks and examples described formally or pragmatically and a presentation with or without examples.

The domain knowledge level is the result of the knowledge the system believes the learner has. This knowledge is calculated on the basis of the knowledge that the learner acquires by solving the exercise or she proves to have (knowledge related to

the exercise). The exercise-related knowledge is measured on the basis of the amount of errors the learner makes while solving the exercise and a final status of the solving process. This status indicates whether the exercise was never intended to be solved, whether the resolving process was not completed, whether the result is incorrect or whether it was solved correctly. The attribute relevance indicates whether the system believes that an exercise is recommendable for the learner at a certain moment. The possible values are: none, not recommended, neutral and recommended.

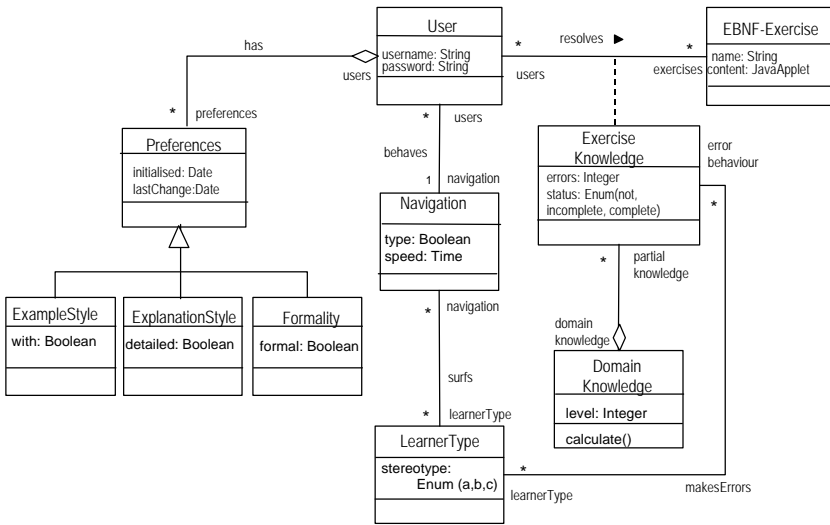


Figure 8-14: User Model for the EBNF-Application

The EBNF-application distinguishes different types of learners (stereotypes) from beginner to expert. The learner type is adjusted using the current exercise domain knowledge and the navigation experience of the learner.

The aim was to build a very simple user model for the EBNF-application. Even though the user model and the update mechanism are simple, the results have been very effective (Albrecht, Koch & Tiller, 2000). Figure 8-14 shows the user model for the EBNF-application.

## 8.3.4

Navigation Model

The navigation model of the EBNF-application consists of the navigation space model that shows *which* objects can be navigated and the navigation structure model that shows *how* they can be visited during an EBNF exercise session.

Two views of the navigation space model (*exercise and general view*) are shown in Figure 8-15 and Figure 8-16, respectively. They are constructed based on the conceptual model of Figure 8-13 and the guidelines provided by Section 6.4.1 as follows:

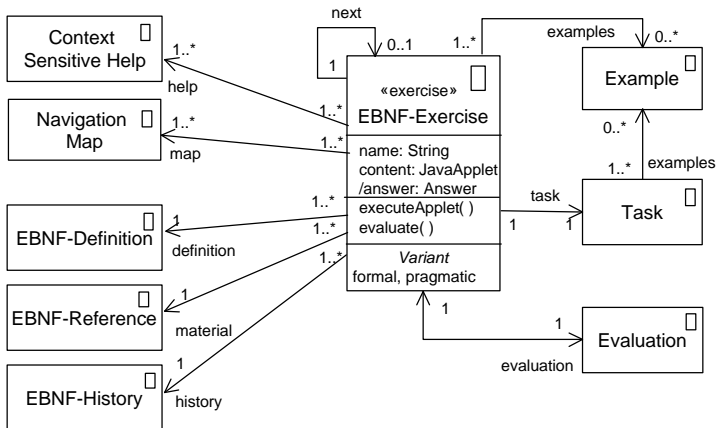


Figure 8-15: Exercise View of the Navigation Space Model of the EBNF-Application

- Navigation classes for all conceptual classes are created with the exception to the abstract classes Help, ReferenceMaterial, Understanding EBNF, UsingEBNF and the class Solution. The latter is not a navigation target as the exercise solution is not shown to the learner. The tutor merely informs the learner as to whether the exercise has been correctly resolved or not, and uses this information for adaptation and to update the user model.
- The conceptual class Answer is transformed in an attribute of the navigation class Exercise since the information is relevant for the application, but it is not a navigation target.



- A stereotype «exercise» is introduced, with the aim of obtaining a more readable UML class diagram and making a more general navigation space model possible. The relationship between objects of the stereotyped classes «exercise», i.e. classes RecognisingMountains, BuildingMountains, RecognisingRules, BuildingRules, DirectSolving, Applying EBNF-Knowledge and classes Task, Example, Evaluation, Help and ReferenceMaterial are described in the *exercise view* depicted in Figure 8-15.
- An association of type direct navigability is added between Exercise and Example. Another association between Evaluation and EBNF-Exercise shows that after an evaluation the learner can try to solve the exercise again.
- The association between GeneralIntroduction and ThemeIntroduction is not included (see Figure 8-16), as the system needs the learner's interview

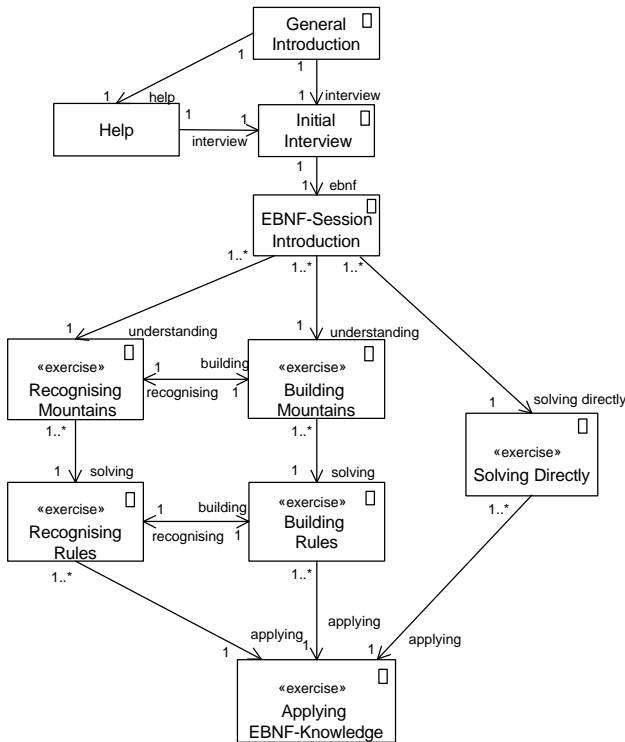


Figure 8-16: General View of the Navigation Space Model of the EBNF-application

responses to initialise the user model.

- Associations of type «prerequisite» are explicitly modeled at subclass level.
- Some role names are omitted in the diagram of Figure 8-16 to avoid overloading.

The navigation space model is transformed in successive steps in the navigation structure model. Again two views of the model are presented. Figure 8-17 shows the *exercised view* of the navigation structure model of the EBNF-application and Figure 8-18 shows the *general view* of the navigation structure model. In Figure 8-17 the variants compartment should be added for completeness; they are omitted here to obtain a simpler diagram.

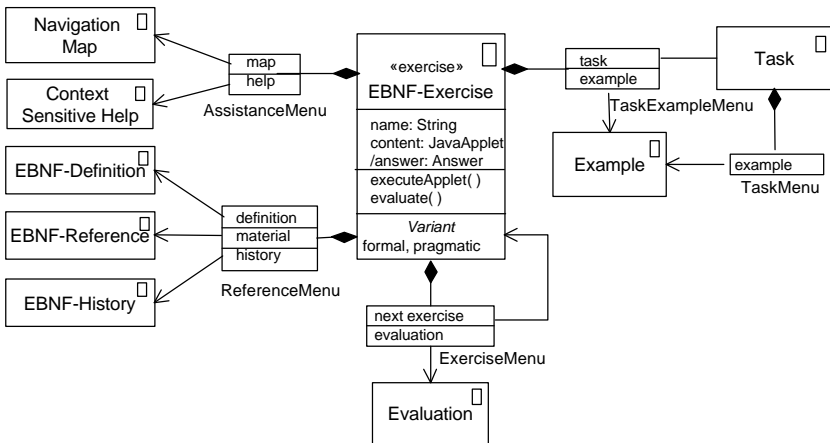


Figure 8-17: Exercise View of the Navigation Structure Model of the EBNF-Application

The *exercise view* of the navigation structure model is obtained by the addition of four menus and a guided tour. Neither indices nor queries are used in this application. The *general view* of the navigation structure model is built by the enhancement of the navigation space model by menus and properties. Adaptive navigation consists of sorted, annotated and removed links and the utilisation of the passive navigation technique (Albrecht, Koch & Tiller, 2000). The properties {annotated}, {removed} and {sorted} are used to indicate where the adaptive mechanism is applied.

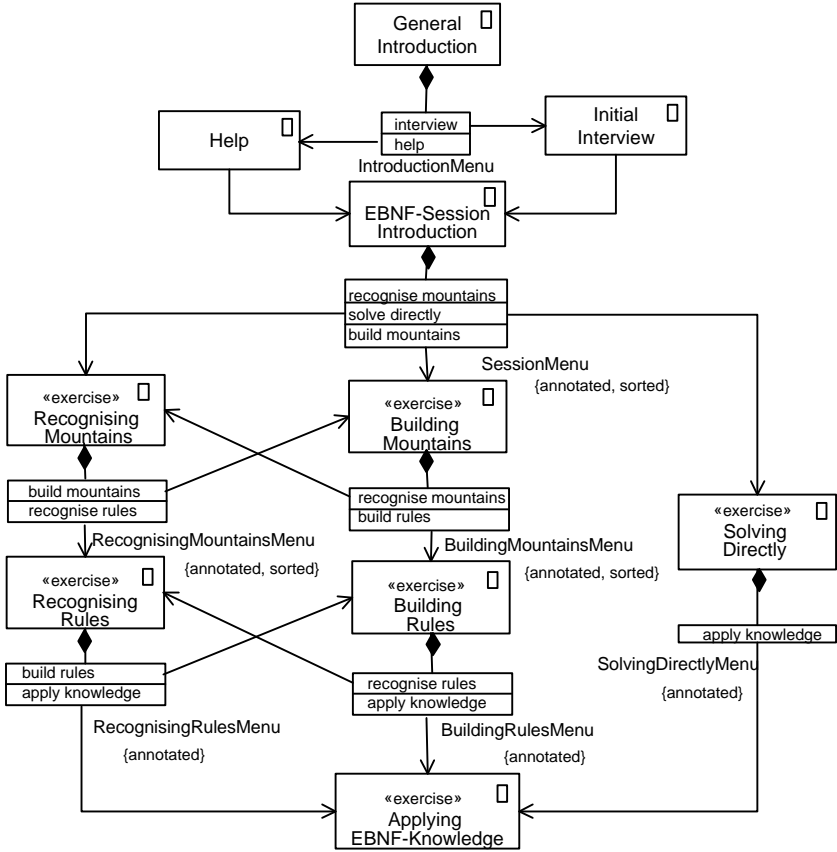


Figure 8-18: General View of the Navigation Structure Model of the EBNF-Application

### 8.3.5

### Presentation Model

The presentation model covers the static and dynamic aspects of the presentation. For the EBNF-application a presentation structure model, a presentation flow model and an abstract user interface model are outlined. Some of these results are presented in this section.

The presentation structure model provides the static description of the EBNF-application's presentation. This is defined as a multiple-window application.

Presentations of the general introduction, the interview or general help are shown in the EBNF-Introduction window; they do not require any frames.

For the exercising session itself, i.e. for the presentation of all pages related to the exercises, an EBNF-Exercising window is used. A Session frameset is defined, which contains four areas: an exercising area, a learning area, an area for global navigation and a thematic navigation area. See Section 8.2.2 for more details about the objective and content of these frames. For the learning area and for the global navigation two frames are defined, LearningArea and GlobalNavigation, respectively. For each of the other two areas a frameset is defined, which includes two and three frames, respectively.

The ExercisingArea frameset thus consists then of an InteractiveExercising frame, where the user resolves the exercise and an ExerciseNavigation frame that enables the student to ask for evaluation and to access the next exercise. The ThematicNavigation is also a frameset with three frames: the ExerciseCategory Navigation, TaskNavigation and ReferenceNavigation. The first is a list of annotated and sorted links to other categories of exercises. The second supports navigation to the exercise task definition and examples. The third supports navigation to general EBNF-material, such as a definition of EBNF, References to EBNF and EBNF-history.

The number of menus contained in the navigation structure model requires this nested, and complex frameset structure. For each navigation class and access primitive a presentation class with the same name is defined.

The designer decided to use three windows: one for the introduction and interview, one for the exercises and one for the context sensitive help. The presentation structure model shown in Figure 8-19 depicts in which frames, framesets and windows these presentation classes are presented to the user.

Some simplifications have been made to avoid an overloaded diagram. Only the association on the left side of a group of associations starting from the same class is provided with the stereotype label «presents». Not all presentation classes have been included in the diagram for reasons of space. In the frame InteractiveExercising the following classes can be presented as alternatives to those already depicted in Figure 8-19: BuildingMountains, RecognisingRules, BuildingRules, SolvingDirectly and Evaluation. The frame LearningArea can present the presentation classes Task, Example, NavigationMap, Help, Definition, Reference and History. The list of presentation classes shown in the ExerciseCategoryNavigation is also incomplete; the classes BuildingMountainsMenu, RecognisingRulesMenu and BuildingRulesMenu have on purpose not been included.

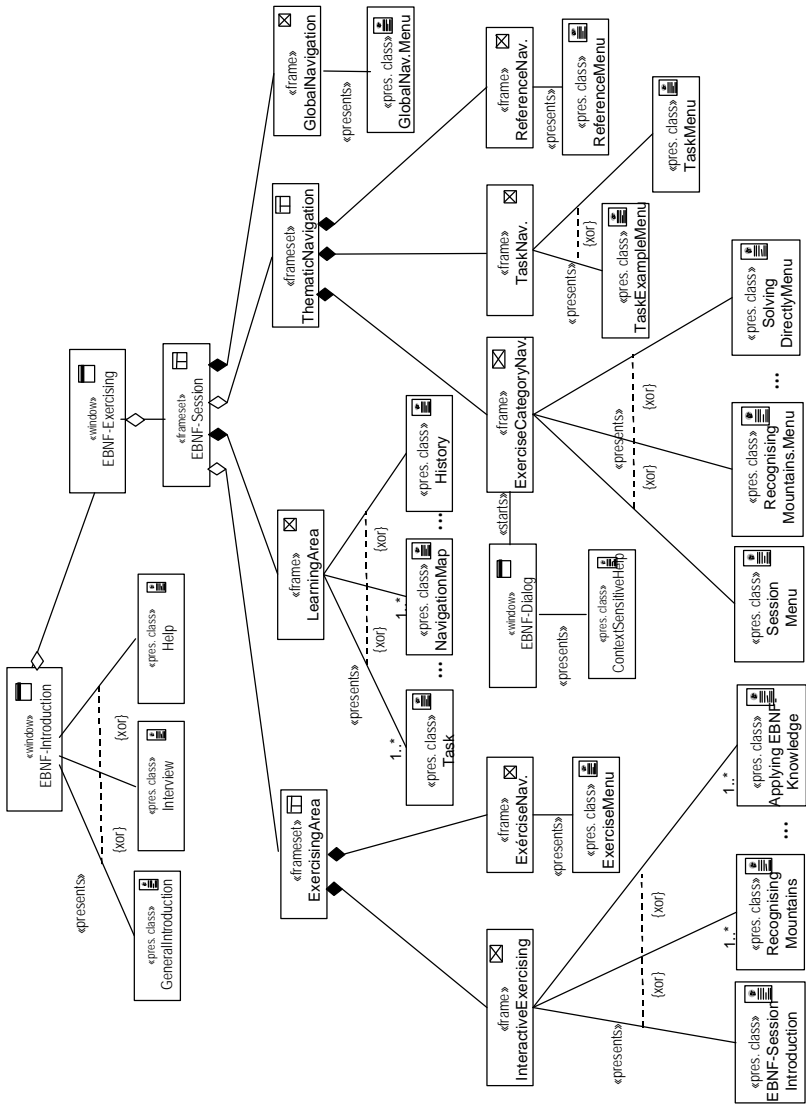


Figure 8-19: Presentation Structure Model of the EBNF-Application

The presentation flow model describes how the control flows from one frame to another, i.e. shows which frame is active at each moment. The UML sequence diagram depicted in Figure 8-20 shows the flow of control of the following typical scenario in the EBNF-application: the user tries to solve an exercise, after looking at the result of the evaluation, she asks for some help and looks at the related examples, then she solves the exercise again and continues with an exercise from another category of exercises.

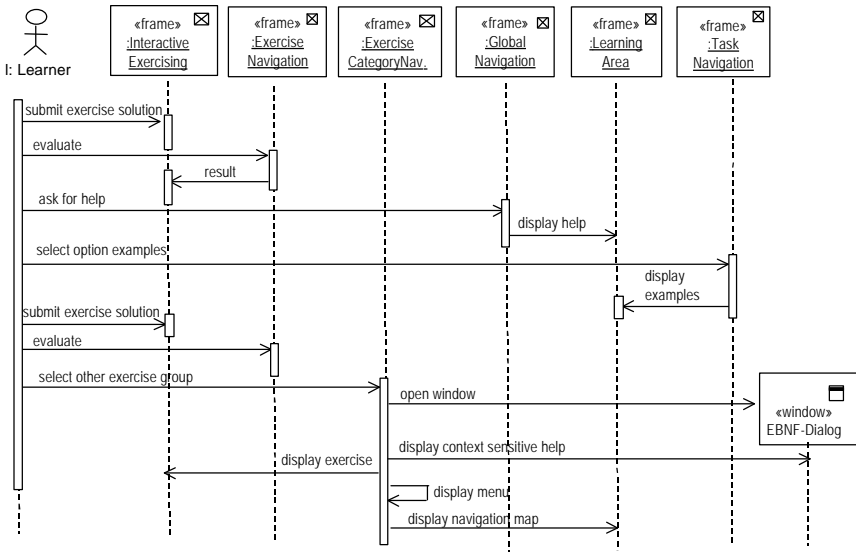


Figure 8-20: Presentation Flow Model for one Scenario of the EBNF-Application

The abstract user interface design proposed in Section 6.5.1. of Chapter 6 provides a sketching technique with UML notation. It can be used in a previous stage to the elaboration of a user interface prototype. An abstract user interface diagram for an exercise of the category “building rules” is shown in Figure 8-21.

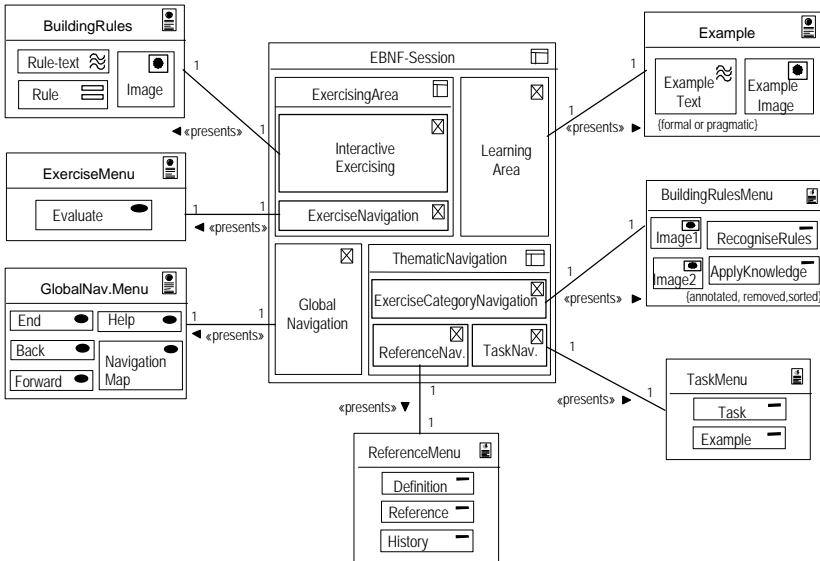


Figure 8-21: Abstract User Interface Model of the Exercise “Building Rules”

### 8.3.6 Adaptation Model

The adaptation process is rule-based. Two types of rules can be defined in the SmexWeb framework: local rules and global rules. The primary trigger of a rule is the user behaviour. Local rules are based on current behaviour while global rules are based on a set of recent activities of the user.

The EBNF-application observes the following user behaviour:

- navigation through categories of exercise chosen,
- amount of back and forward navigation,
- navigation to additional information, such as examples, help, and reference material,
- final result resolving an exercise is evaluated,
- errors made resolving an exercise, and

- inactivity, i.e. no browsing activity during, for example, 5 minutes.

These behaviours can be classified into browsing (first three items), input (fourth and fifth) and inactivity (last one).

The following rules are part of the EBNF-application. A textual description of some rules is presented here. In a further iteration a formal language can be used for the description of these rules.

For the EBNF-application only *acquisition* and *adaptation rules* are defined. Rules for finding concepts are not needed in this application, as each concept is also a page. Two types of acquisition rules are distinguished: *initialisation rules* and *user model update rules*. The initialisation rules are used to construct a user model providing user attributes with initial values. These values are determined using the information obtained from the interviewing process. User model update rules will change these values based on observations of the user's behaviour while she navigates and solves the exercises.

The following is an informal description of some initialisation rules:

- Rule 1: Navigation experience is determined by the kind of Web services the learner uses and the frequency with which she uses them.
- Rule 2: The values of the learner's preferences (ExampleStyle, ExplanationStyle and Formality) are initialised on the basis of the major and minor subjects studied by the student, her age and her interests.
- Rule 3: The initial DomainKnowledge is set as a function of the result of the test on EBNF basic concepts.
- Rule 4: The initial LearnerType is chosen in accordance with the kind of lectures the learner visits, her navigation experience and the domain knowledge.

The following is an informal description of some updating rules:

- Rule 5: The result of the evaluation of the answer to an exercise is stored as the status of the exercise. Possible values are: not resolved (if it was not intended to be resolved), incomplete, incorrect and correct.
- Rule 6: Each error performed by the learner while resolving an exercise increments the errors counter of the exercise.
- Rule 7: DomainKnowledge is update using the current values of errors and the status of ExerciseKnowledge.



- Rule 8: Navigation expertise is updated on the basis of the learner's navigation activities, such as back and forward navigation or selection of certain navigation paths.
- Rule 9: The LearnerType is updated according with the changes in navigation expertise and domain knowledge level.
- Rule 10: For each exercise node the relevance is adjusted in accordance with the current value of the DomainKnowledge.

The following are examples of adaptation rules:

- Rule 11: If the user prefers formal descriptions, the task and the examples are displayed in their formal variant, otherwise they are presented as the more pragmatic variant.
- Rule 12: If the user prefers brief explanations, all type of reference material, such as EBNF-definition, EBNF-Reference and EBNF-History are presented in their brief variant, otherwise in their detailed variant.
- Rule 13: Annotation is performed in the menus, e.g. SessionMenu, RecognisingRulesMenu and SolvingDirectlyMenu, as follows:
  - happy smiley for a recommended category of exercises,
  - neutral smiley for a category of exercises, which are whether specially encouraged to be solved or discouraged, and
  - unhappy smiley for a non-recommended category.
- Rule 14: If the learner remains inactive for more than three minutes, then the context sensitive help associated to that exercise is shown (passive navigation).
- Rule 15: Items of the navigation menus (i.e. SessionMenu, RecognisingMountainsMenu, BuildingMountainsMenu, RecognisingRulesMenu, BuildingRulesMenu and SolvingDirectlyMenu) are sorted in accordance with the relevance of the target node the item links to.

Figure 8-22 shows part of an adaptation model for the EBNF-application. The model is represented as a collaboration diagram. The graphical visualisation permits the recognition of loops in the flow of rules triggered by other rules.

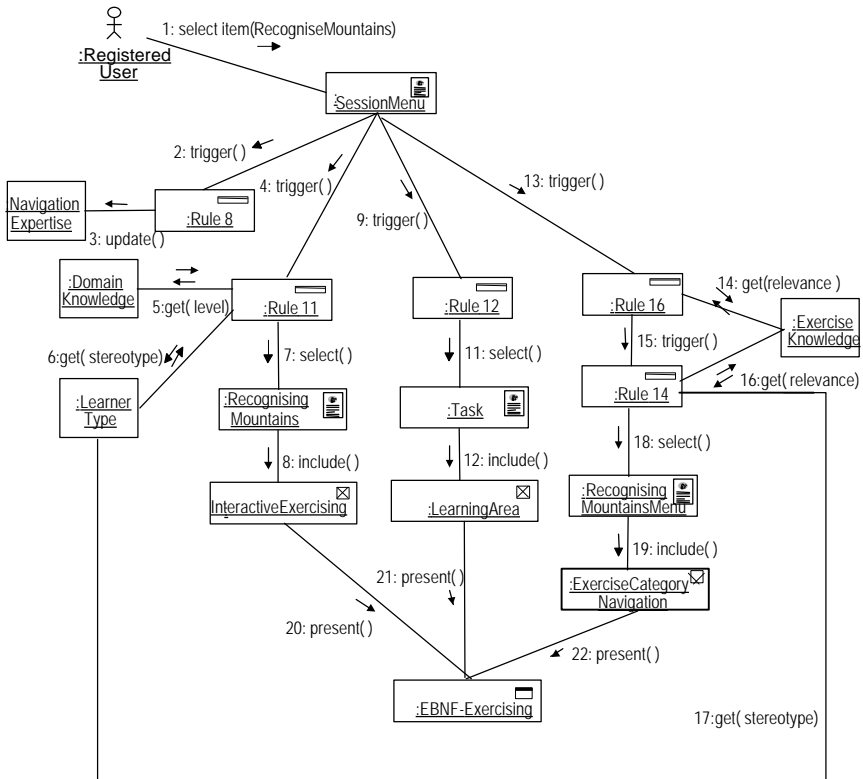


Figure 8-22: Adaptation Model of the EBNF-Application (Partial View)

## 8.4 The Taxonomy Application

The Taxonomy application is an adaptive hypermedia application based on the SmexWeb framework that was built to support students in their studies of taxonomy in the botanical field.

The application consists of the presentation of different types of questions to the user, an evaluation of their responses, an update of the user model based on these answers and the selection of another appropriate question for a particular user based on the current values of the user model. As the Taxonomy application is

supposed to be used in one to six sessions, the system's beliefs about the user are also stored in a long-term user model. The exercises (questions) are classified according to the type of answer expected for these questions: true/false, a single statements, multiple statement or multiple choice (Pezdirc, 1999).

The development of the Taxonomy application also implied an improvement in the SmexWeb framework, as a database connection was added to the framework. It enables the generation of the page content from the information stored in a database. The JDBC<sup>18</sup> driver and the MySQL<sup>19</sup> database were used for this purpose.

The Taxonomy application was tested using a group of students from a seminar on plants identification and classification<sup>20</sup>.

## 8.5 Learning process supported by SmexWeb

SmexWeb supports an *active, constructive, cumulative, self-regulated* and *goal-oriented knowledge acquisition process* in which the learners play an important role. Classical psychological theories view learning as something that happens from the outside in – passive reception –, as knowledge is transferred from the expert to the novice. Nowadays cognitive theories have reversed this orientation emphasising that learning occurs from the inside out although the importance of the learner's environment is not questioned (Shuell, 1992). Web applications developed based on the SmexWeb framework are adaptive Web-based applications that support learning processes based on the well-known metaphor of problem solving.

Learning with a SmexWeb applications is:

- *Active* as the learner must carry out cognitive operations (learning by doing).
- *Constructive* in the sense that it helps every learner to create her own knowledge structures. New information is perceived and interpreted in a unique manner based on the learner's prior knowledge and other personal factors.

---

<sup>18</sup> JDBC (Java DataBase Connectivity) <http://splash.javasoft.com/database/jdbc/jdb c.drivers.html>

<sup>19</sup> MySQL Version 3.21.33. <http://www.mysql.com>

<sup>20</sup> "Praktikum zur Artenvielfalt/Pflanzenbestimmung", Döbbeler/Rambold, Botanischen Institut der Ludwig-Maximilians-Universität München, SS99.

- *Cumulative*, because it builds upon and is influenced by the learner's prior knowledge, which is registered in the user model.
- *Self-regulated* as the learner determines the duration and frequency of the sessions. Students are also free to decide which link to choose next, although they are assisted by the system with some guidance and help.
- *Goal-oriented* for the learner. The application presents clear goals to be achieved during each session within the context of the general goal of acquiring knowledge about certain topics. The learner can then establish her goal for the session.

*“If we end up producing a structure in hyperspace  
that allows us to work together harmoniously,  
that would be a metamorphosis”  
Tim Berners-Lee,  
Weaving the Web, 1999.*

## 9

## Conclusions

---

The impact of new information technologies on society has not only stimulated the development of systems using these technologies, but has also increased the interest in studies related to the development process of such systems. Object-orientation, hypermedia, components and distributed systems are typical examples of information technologies, which became popular in the nineties. The expansion of the Web brought hypermedia systems, in particular, to the attention of managers, business and marketing people, developers, designers, programmers, and, last but not least, researchers. The ubiquity of the Web brought the need of personalisation adapting hypermedia applications to the user. The expansion of the Web made us consider disciplined ways to master the complexity of these applications.

Until now adaptive hypermedia applications have mostly been constructed as successive refinements of initial prototypes generally in experimental environments. But an important increase in industrial personalised applications is expected, based on the maturity reached by these adaptive systems and the user modeling techniques. However, the construction will have to be performed in a more systematic way following guidelines, constructing adequate models, and using appropriate tools, i.e. the production of adaptive hypermedia systems thus requires a tailored software engineering process.

In this work such an engineering approach for adaptive hypermedia systems (UWE) is presented. The UWE approach supports hypermedia issues, such as navigation and hypertext structure, and adaptive issues, such as user modeling and adaptation mechanisms.

The main characteristics of the UWE approach are:

- It is an entirely object-oriented approach.
- It presents a reference model formally specified in OCL.
- It supports visual design modeling techniques.
- It provides a UML profile for adaptive hypermedia applications.
- It defines a development process that covers the whole lifecycle of adaptive hypermedia applications.

A clear separation of adaptive and non-adaptive topics also makes of the engineering approach presented here an ideal methodology for the analysis and design of general hypermedia applications.

The next three sections outlines concluding remarks on the main issues relating to the engineering approach on which this work focuses: the reference model, the modeling techniques and the development process. The last section proposes some future research.

## **9.1 Concluding Remarks about the Reference Model**

The *Munich Reference Model* for adaptive hypermedia systems presented in this work (Chapter 4) is a Dexter-based approach, which uses the well-known Dexter metamodel language. It is a formal, object-oriented approach that benefits from a combination of graphical specification in UML (1999) and constraints specified in OCL.

UML class diagrams allow for a visual representation of the Munich reference model showing the concepts of the system and how they are related. This graphical representation is missing when specification languages such as VDM (Jones, 1990), Z (Halasz & Schwarz, 1990) or ObjectZ (Van Ossenbruggen & Eliëns, 1995) are used.

OCL (Warmer & Kleppe, 1999) is used intensively for the specification of invariants for the model elements and the pre- and post-conditions of the operations, which describe the functionality of an adaptive hypermedia system.

During the development of this thesis De Bra, Houben and Wu (1999) independently wrote an interesting work in the area of adaptive hypermedia. Their work presents the Adaptive Hypermedia Application Model (AHAM). Both that model and the reference model presented in this thesis are Dexter-based models.

AHAM addresses pedagogical applications, which perform adaptations based on user models represented by tables. AHAM is described with tuples; a formal description in Z is in preparation. The Munich reference model focuses on an object-oriented approach presenting a semi-formal visual model in UML and a formal specification in OCL. It has no restriction for the type of adaptive hypermedia systems.

OCL is quite a new language and only few works report about the experience using OCL, such as the article of Baar (2000). Besides some minor improvements that would optimise the specification, it transpires that OCL is adequate for a specification of this type. The readability of the specification could be enhanced by the addition of some constructs, such as *domain* and *range*, for example. Some difficulties have been observed in the definition of the transitive closure. The specification presented by Mandel and Cengarle (1999) was improved by the use of the construct “let in”, which has been included in the UML version 1.3. Even though computing length of the transitive closure has been reduced, it remains unnecessarily complex. An additional problem is that OCL requires an *isQuery* value equal true for each function included in a post-condition. In contrast, the visualization using UML diagrams and visual representation of some constraints in the diagrams through associations and multiplicity allows for a more compact and intuitive specification.

The Munich reference model serves as a basis for the definition of the modeling techniques used in the design of adaptive hypermedia applications (Chapter 6). The domain model requires a conceptual design of the problem domain, which will evolve into a navigation model and a presentation model. The user model and the adaptation model find their pendant in the design. The user model is used to define user attributes and their relationships to the domain model. The adaptation model is used to specify the set of acquiring and adaptation rules as well as the collaborations between these rules and elements of the domain model and user model.

## 9.2

## **Concluding Remarks about the Modeling Techniques**

Particular attention is paid to the analysis and design of adaptive hypermedia applications. Within the scope of this work special modeling techniques were developed to support the analysis and design workflow of the development process. These are centred on the hypermedia and adaptive design issues, making a clear

separation between content, structure, presentation, user modeling and adaptation mechanisms.

The modeling techniques – known as UHDM – consists of a set of modeling elements, models and a method that specifies how to build these models. The set of modeling elements is defined as a UML profile for adaptive hypermedia applications based on the UML extension mechanisms, including mainly descriptive and restrictive stereotypes (Berner, Glinz & Joos, 1999). The models are represented with UML diagrams, i.e. the techniques support visual modeling. Some of the modeling elements occurring in such diagrams are defined by stereotypes. The definition of new stereotypes means that extra effort needs to be put in reading the diagrams, but once one gets used to them, the diagrams are more meaningful in terms of Web analysis and design. The advantages of the presented techniques are the use of UML, the consideration of specific Web aspects in designing Web applications through the definition of specialised modeling elements, and the creation of tailored models to express navigation, presentation and adaptation.

The strength of this approach is that for each model, a detailed list of construction steps is provided, many of which can be performed automatically, for instance, when constructing the navigation structure model from the navigation space model. In addition, the method describes how templates for the Web application can be systematically generated from the navigation structure model. However, there are still several steps for which decisions by the designers are essential. This is true, in particular, for the construction of the navigation space model based on the conceptual model, the user model definition and the specification of the adaptation mechanism. The design is partly a creative process where a complete automation is not possible. In such a process it is extremely supportive to follow modeling guidelines and to use patterns in order to achieve a systematic construction.

The modeling techniques specify how to build:

- the navigation space model based on the conceptual model,
- the user model with the aim of capturing the user's knowledge and preferences,
- the navigation structure model from the navigation space model,
- the static and dynamic presentation model from the navigation structure model, and
- the adaptation model needed to update the user model and to adapt the application.



These modeling techniques (UHDM), described in Chapter 6, are part of the UWE approach for adaptive hypermedia (Web) applications.

### 9.3 Concluding Remarks about the Development Process

The *methodology proposed for the development of adaptive hypermedia applications* (UWE) presented in this work (Chapter 7) is based on the Unified Process. It uses the UML profile and modeling techniques described briefly in the above section (for more details see Chapter 6). The aim is to cover the whole lifecycle of such adaptive hypermedia systems. Therefore specific artifacts, workers and activities required to support user modeling and adaptation have been defined and supporting workflows for project and quality management have been added as part of the here presented engineering approach.

UWE is a specialisation of the Unified Process for the adaptive hypermedia domain and at the same time it is an extension of the Unified Process to include project management and quality management support. UWE is an object-oriented, iterative and incremental process, which consists of a set of workflows. The workflows of the development process are the requirements capture, analysis and design, and implementation. The workflows of the project management are risk management, iteration planning and iteration evaluation. Validation, verification and testing are part of the quality management. Based on time, the process is divided into iterations, which belong to one of the following phases: inception, elaboration, construction, transition or maintenance. UWE describes the set of *activities, workers and artifacts* required for each workflow focusing on the user modeling and adaptive issues of the application to be developed.

UWE, such as the Unified Process and the Rational Unified Process is a best practice approach. It therefore describes the software development as an iterative process that manages requirements, uses component-based architectures, uses visual modeling techniques and controls quality. The disadvantage of being best practice-based is that the methodology will evolve and will be continuously updated. UWE is flexible enough to allow tailoring over time and adaptation to the developers' needs.

## 9.4

## Proposing Future Research

Web engineering – hypermedia engineering for the Web – is a new discipline, which is still evolving. Even more innovative is the personalised Web discipline based on adaptive hypermedia techniques. There is still much work needed to improve the current hypermedia and Web engineering approaches. This will be done mostly in small steps through the continuous adjustment of methodologies, techniques, notations and tools. This work aims to be one of those small steps.

There are many open issues, which still need to be addressed and integrated, such as patterns for adaptive hypermedia systems, sharing of user models, tool support and incorporation of agent technologies.

Throughout the description of the modeling techniques used for the design of adaptive hypermedia systems a set of domain patterns are presented (Chapter 6), such as the access structure, presentation structure and adaptation. It is shown how these patterns can be combined with the design method (Gamma et. al., 1995). Patterns identified in this work have to be described with an appropriate level of abstraction and formalisation, an adequate classification, focusing the problem they address and with a discussion of advantages and disadvantages that their use implies (Paolini & Garzotto, 1999 and Nanard & Nanard, 1999).

The use of user models across many applications will simplify the development of adaptive applications as the complete user modeling is outsourced. A future methodology for Web design will also have to scope with the design of multi-modal interfaces including e.g. speech. Synchronisation problems will have to be solved in these kinds of flexible Web applications.

Another important future step will be to construct a case tool to support the UWE methodology or to extend the functionality of an existing case tool to support it. The stereotypes defined for hypermedia (Web) applications in Chapter 6 will be implemented as a plug-in feature of the open-source tool ArgoUML (2000). Tools for UML are developing fast, but there is an enormous scope for improvement; indeed there is widespread dissatisfaction with the state of the art. They need to increase their capabilities to provide automatic verification of models, to support the use of patterns, and to allow for constraint specification with OCL. In addition, they must include special features for Web development since Web applications are becoming one of the most important group of software applications.

Agent technology will also play a more essential role in software applications than they do nowadays. Digital agents act in an independent manner (not only when the system is used) collecting information, negotiating with other agents, performing

adaptation and learning from past actions. There will be agents suggesting patterns and workflow activities, agents helping in modeling and implementation, and agents finding information. Agents collect information about the user, create personalised environments to search and work and offer assistance in all types of user-computer interactions, i.e. agents are autonomous and adaptive (Lieberman, 1995, Thomas & Fischer, 1996, and Mladenic, 2000).

Future research should result in the development of more formal methods for eliciting requirements and modeling these requirements in an unambiguous way. Formal methods are also required for the development of verification and testing procedures for adaptive hypermedia applications. The methodology proposed in this work, for example, still requires validation and testing for a wide spectrum of Web applications (Chapter 8), in particular for e-commerce applications.

The UML-based Web engineering approach must also evolve to cope with technological changes. A new Web, called Semantic Web – World Wide Knowledge (Berners-Lee, 1999) will add certain intelligence to the current Web. The Semantic Web is defined as a knowledge-based model for hypermedia that will support a Web structure enriched by computer power, which allows interpretation of nodes and links and reasoning based on the interpretation. Thus, it should be investigated how the proposed typed and semantic links influence the design and implementation of adaptive hypermedia applications.



*" Reading about using the UML is one thing,  
but it's only through using the language  
that you will come to master it"*  
Grady Booch, James Rumbaugh and Ivar Jacobson,  
*The Unified Modeling Language: User Guide,*  
1999.

## **Appendix UML Extension for Hypermedia**










This Appendix presents the UML Profile for adaptive hypermedia applications. The stereotypes are defined and explained in Chapter 6. It is a UML extension based on the general extension mechanisms provided by the UML. The extension includes specific stereotypes to model the navigation, presentation and adaptive aspects of hypermedia applications. The models that benefit from the UML extension are the navigation space model, the navigation structure model, the adaptation model and the presentation models, i.e. presentation structure model, the presentation flow model and the abstract user interface model.








There are only a few stereotypes that are specific to the modeling of adaptive features. Thus, the stereotypes are grouped in stereotypes for general hypermedia and specific stereotypes for adaptive hypermedia.





For standards elements of the UML and standard stereotypes defined in the UML see Booch, Rumbaugh and Jacobson (1999).

### **Stereotypes for Modeling Hypermedia Applications**

The UML Profile defines the following stereotypes for the design of general hypermedia applications. They are used in the construction of the navigation space model, the navigation structure model and the presentation models.

Stereotype / Icon	Applies to	Meaning
anchor 	class	Specifies a class whose objects are clickable areas, which have associated links to other nodes.
anchored collection 	class	Specifies a class whose objects are collections of anchors.
audio 	class	Specifies a class whose objects are audio sequences that can be started, stopped, rewind and forwarded.
button 	class	Specifies a class whose objects are clickable areas, which have actions associated to them.
collection 	class	Specifies a class whose objects are a set of other elements, such as text, image, etc. It is not specified how the set will be displayed.
direct navigability 	association	Specifies that the target object is accessed by direct navigation from the source object. The direction of navigation is shown by an arrow that is attached to one or both ends of the association (bidirected).
external node 	class	Specifies a class whose objects are targets belonging to another hyperspace.
form 	class	Specifies a class whose objects are used to request information, which will be supplied in one or more input fields or will be selected by options from a browser or checkbox.
frame 	class	Specifies the lower level area a frameset is divided into.

Stereotype / Icon	Applies to	Meaning
frameset 	class	Specifies a class whose objects are top level elements modeled by a composite that contains lower level objects (frames). Framesets are always contained in a window and may be nested.
guided tour 	class	Specifies a class whose objects provide sequential access to instances of a navigation class. The directed association that connects a guided tour to a navigation class has the property {ordered}.
image 	class	Specifies a class whose objects are a visual and displayable multimedia object.
index 	class	Specifies a class of composite objects that contain an arbitrary number of index items. Each index item is in turn an object which has a name and owns a link to an instance of a navigation class.
menu 	class	Specifies a class of composite objects that contain a fixed number of menu items. Each menu item has a constant name and owns a link either to an instance of a navigation class, an index, a guided tour, a query or another menu.
navigation 	class	Specifies a class whose objects are obtained from corresponding conceptual objects and are visited by the user during navigation.
presentation 	class	Specifies a class whose objects are the presentation of navigation objects or an access primitive, such as an index, a guided tour, query or menu. It is a container, which comprises elements like texts, images, video, anchors, etc.



Stereotype / Icon	Applies to	Meaning
presents	association	Specifies that the target object is displayed in the location indicated by the source object.
query 	class	Specifies a class whose objects have query strings as attributes. These strings may be, for instance, OCL select operations.
text 	class	Specifies a class whose objects are sequences of characters.
video 	class	Specifies a class whose objects are video sequences that can be started, stopped, rewind and forwarded.
window 	class	Specifies a class whose objects have assigned an area of the user interface, where framesets or presentation objects are displayed. They can be moved, resized and reduced to icons. Each window object includes at least two buttons, one to be transformed into an icon and one to be closed.

## **Stereotypes for Modeling Adaptive Features**

The UML Profile defines the following stereotypes to model the adaptive functionality of adaptive hypermedia systems. They are used in the construction of the navigation structure model, the adaptation model and the presentation models.

Stereotype / Icon	Applies to	Meaning
adapted language	presentation class	Specifies that the presentation object's text is presented in a language that depends on the user model.



Stereotype / Icon	Applies to	Meaning
annotated	menu item index item anchor	Specifies that an index item, a menu item or an anchor has a visual annotation indicating its relevance.
direct guidance	guided tour anchor	Specifies that the system decides, which is the "best" target node.
layout variant	presentation class	Specifies that the presentation object's layout is user model dependent.
passive navigation	association	Specifies that the navigation following the association is performed by the system, e.g. when the user remains inactive.
removed	menu item index item anchor	Specifies that if the system believes that an item or anchor is not relevant for the user, it is visually removed.
rule 	class	Specifies a class whose objects contain principles that determines how to update the user model, how to find appropriate concepts or how to adapt the application.
sorted	menu item index item anchor	Specifies that the corresponding object belongs to a group of items or anchors that are sorted to indicate their relevance.
user behaviour 	class	Specifies a class whose objects contain the result of user observation.



*“Wisdom is not a product of schooling,  
but the lifelong attempt to acquire it”  
Einstein*

---

## References

- Albrecht F. (1998). SmexWeb: Ein adaptives Web-basiertes Übungssystem. *Diplomarbeit, Ludwig-Maximilians-Universität München.*
- Albrecht F., Koch N. & Tiller T. (1999). Making Web-based Training More Effective. *Proceedings of the Seventh Workshop ABIS-99: Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen*, T. Joerding (Ed.).
- Albrecht F., Koch N. & Tiller T. (2000). SmexWeb: An Adaptive Web-based Hypermedia Teaching System. *Journal of Interactive Learning Research, Special Issue on Intelligent Systems/Tools in Training and Lifelong Learning*. Kommers P. & Mizoguchi R. (Eds.), 367-388.
- Anderson J. & Skwarecki E. (1986). The Automated Tutoring of Introductory Computer Programming. *Communications of the ACM*, 29, 9, 842-849.
- Ardissono L. & Goy A. (1999). Tailoring the Interaction with Users in Electronic Shops. *In User Modeling Conference*, 35-44.
- ArgoUML (2000). <http://www.tigris.org>
- Armstrong R., Freitag D., Joachims T. & Mitchell T. (1995). WebWatcher: A Learning Apprentice for the World Wide Web. *AAAI Spring Symposium on Information Gathering from Distributed Heterogeneous Environments*.
- Asnicar F. & Tasso C. (1997). ifWeb: A Prototype of User-Model-Based Intelligent Agent for Document Filtering and Navigation in the World Wide Web. *Proceedings of the Workshop Adaptive Systems and User Modeling on the World Wide Web*. User Modeling Conference '97.
- Avison D. & Fitzgerald G. (1995). *Information Systems Development: Methodologies, Techniques and Tools*. Mc Graw-Hill.

- Åberg J. & Shahmemehri N. (1999). Web Assistants: Towards an Intelligent and Personal Web Shop. *Proceedings of the 2<sup>nd</sup> Workshop on Adaptive Systems and User Modeling on the WWW*.
- Baar T. (2000). Experiences with the UML/OCL-Approach in Practices and Strategies to Overcome Deficiencies. *Net.ObjectDays 2000*, Germany.
- Balasubramanian V., Bieber M. & Isakowitz T. (1996). Systematic Hypermedia Design. *Technical report, CRIS Working Paper Series, Stern School of Business, New York University*.
- Ballacker K., Lawrence S, & Giles L. (2000). Discovering Relevant Scientific Literature on the Web. *IEEE Intelligent Systems*, 15 (2), 42-47.
- Baumeister H., Koch N.& Mandel L. (1999). Towards a UML Extension for Hypermedia Design. *Proceedings of The Unified Modeling Language Conference: Beyond the Standard (UML'99)*. France R. and Rumpe B. (Eds). LNCS 1723, Springer Verlag, 614-629.
- Beck J., Stern M. & Haugsjaa E. (1996). Applications of AI in Education. *ACM Crossroads*, 3 (1). <http://www.acm.org/crossroads/xrds3-1/aied.html>.
- Benyon D. & Murray D. (1993). Applying User Modeling to Human-Computer Interaction Design. *Artificial Intelligence Review*, 7, 199-225, Kluwer Academic Publishers.
- Berner S., Glinz M. & Joos S. (1999). A Classification of Stereotypes for Object-oriented Modeling Languages. *In Proceedings UML'99-The Unified Modeling Language: Beyond the standard Conference*. France R. and Rumpe B.(Eds.).LNCS 1723. Springer Verlag, 249-264.
- Berners-Lee T. (1999). *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. Harper San Fransisco.
- Bichler M. & Nusser S. (1996). Developing Structured WWW-Sited with W3DT. *WebNet 96*.
- Boehm B. (1981). *Software Engineering Economics*. Prentice Hall.
- Boehm B. (1988). A Spiral Model for Software Development and Enhancement. *Computer*, May, 61-72.
- Boehm B. (1991). Software Risk Management: Principles and Practices. *IEEE Software*, 32-41.
- Boehm B., Abts C., Brown A., Chulani S., Clark B. & Horowitz E. (2000). *Software Cost Estimation with Cocomo II*. Prentice Hall.
- Booch G. (1994). *Object-oriented Analysis and Design with Applications*. Cummings Publishing Company.
- Booch G., Rumbaugh J. & Jacobson I. (1999). *The Unified Modeling Language: A User Guide*. Addison Wesley.
- Boyle T. (1997). *Design for Multimedia Learning*. Prentice Hall.
- Boyle T. & Encarnação M. (1994). MetaDoc: An Adaptive Hypertext Reading System. *User Modelling and User Adapted Interaction*, 4 (1), 1-19.

- Brajnik G. & Tasso C. (1992). A Flexible Tool for Developing User Modeling. *Proceedings of UM '92 User Modeling Workshop*.
- Brusilovsky P. (1996a). Adaptive Hypermedia: An Attempt to Analyze and Generalize. *Proceedings of First International Conference on Multimedia, Hypermedia and Virtual Reality 1994*. Brusilovsky P. & Streitz N. (Eds.) LNCS 1077, Springer Verlag, 288-304.
- Brusilovsky P. (1996b). Methods and Techniques of Adaptive Hypermedia. *International Journal of User Modeling and User-Adapted Interaction*. Kluwer Academic Publishers, Vol 6, 2-3, 87-129.
- Brusilovsky P. (1997). Efficient Techniques for Adaptive Hypermedia. *Intelligent Hypertext: Advanced Techniques for the World Wide Web*. Nicholas C. & Mayfield J. (Eds.), Springer Verlag, 12-30.
- Brusilovsky P. (1998). Methods and Techniques of Adaptive Hypermedia. *Adaptive Hypertext and Hypermedia*. Brusilovsky P. et al. (Eds.), Kluwer Academic Publishers, 1-43.
- Brusilovsky P. & Cooper D. (1999). ADAPTS: Adaptive Hypermedia for a Web-based Performance Support System. *2<sup>nd</sup> Workshop on Adaptive Systems and User Modeling on the WWW*.
- Brusilovsky P. & Eklund, J. (1998). A Study of User Model Based Link Annotation in Educational Hypermedia. *Journal of Universal Computer Science*, 4 (4), 429-448, Springer Science Online.
- Brusilovsky P., Schwarz E. & Weber G. (1996a). ELM-ART: An Intelligent Tutoring System on World Wide Web. *Proceeding of Third International Conference on Intelligent Tutoring Systems ITS-96, LNCS 1086, Springer Verlag*, 261-269.
- Brusilovsky P., Schwarz E. & Weber G. (1996b). A Tool for Developing Adaptive Electronic Textbooks on WWW. *Proceeding of WebNet '96, World Conference of the Web Society*, 64-69.
- Bull S. & Smith M. (1997). A Pair of Student Models to Encourage Collaboration. *User Modeling Proceedings of the Sixth International Conference, UM97*, A. Jameson, C. Paris and C. Tasso (Eds.), Springer Verlag Wien, 339-341.
- Bulterman D., Rutledge L., Hardman L. & van Ossenbruggen J. (1999). Supporting Adaptive and Adaptable Hypermedia Presentation Semantics. *The 8<sup>th</sup> IFIP 2.6 Working Conference on Database Semantics (DS-8): Semantic Issues in Multimedia Systems*.
- Bush, V. (1945). As We May Think. *The Atlantic Month*, 176 (1), 101-108.
- Campbell B. & Goodman J. (1988). HAM: A General Purpose Hypertext Abstract Machine. *Communications of the ACM*, Vol. 31 (7).
- Carneiro L., Cowan D. & Lucena C. (1993). Introducing ADV Charts: A Graphical Specification of Abstract Data Views. *Proceedings of CASCON '93*.
- Carro R., Pulido E. & Rodriguez P. (1999). TANGOW: Task-based Adaptive learner Guidance on the WWW. *Proceedings of the Second Workshop on Adaptive Systems and User Modeling on the World Wide Web*, 49-57.

- Cas K. & Bingler D. (1998). Adaptive Briefing Books basierend auf einer client-server Architektur. *ABIS-98: Workshop on Adaptivity and User Modeling in Interactive Software Systems*, FORWISS Report.
- Ceri S., Fraternali P. & Bongio A. (2000). Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. *Proceedings of WWW9*.
- Chin D. (1993). Acquiring User Models. *Artificial Intelligence Review*. Kluwer Academic Publishers. 185-197.
- Collins J., Greer J., Kumar V. & McCalla G. (1997). Inspectable User Models for Just-in-time Workplace Training. *User Modeling Proceedings of the Sixth International Conference, UM97*, Jameson A., Paris C. and Tasso C. (Eds.), Springer Verlag Wien, 327-337.
- Conallen J. (1999). *Building Web Applications with UML*. Addison-Wesley.
- Conklin J. (1987). Hypertext: A Survey and Introduction. *IEEE Computer* 20 (9), 17-41.
- Corbett A. & Anderson J. (1995). Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge. User Modeling and User-Adapted Interaction, *Kluwer Academic Publishers*, 4, 253-278.
- Cordingley E. (1989). Knowledge Elicitation Techniques for Knowledge-based Systems. *Knowledge Elicitation: Principles, Techniques and Applications*, Diaper (Ed.), Ellis Horwood.
- De Bra P. & Calvi L. (1998). AHA: A Generic Adaptive Hypermedia System. *Proceeding of the 2<sup>nd</sup> Workshop on Adaptive Hypertext and Hypermedia, HYPERTEXT'98*, 5-11.
- De Bra P. (1999). Design Issues in Adaptive Web-Site Development. *Proceedings of the 2<sup>nd</sup> Workshop on Adaptive Systems and User Modeling on the WWW*.
- De Bra P., Brusilovsky P. & Houben G.J. (1999). Adaptive Hypermedia: From System to Framework. *ACM Computing Surveys*, Vol 31 (4).
- De Bra P., Houben G.-J. & Kornatzky Y. (1992). An Extensible Data Model for Hyperdocuments. *Proceedings of the 4<sup>th</sup> ACM Conference on Hypertext*, 222-231.
- De Bra P., Houben G.-J. & Wu H. (1999). AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. *Proceedings of the ACM Conference on Hypertext and Hypermedia*, 147-156.
- De Bra, P. (2000). Using Hypertext Metrics to Measure Research Output Levels. *Scientometrics*, *Kluwer Academic Publishers*, 47 (2), 227-236.
- Dekkers, C. (1999). Function Points and Use Cases – Where's the Fit? *IT Metrics Strategies*, January.
- de La Passardiere B. & Dufresne A. (1992). Adaptive Navigational Tools for Educational Hypermedia. *Proceedings of Computer Assisted Learning*, Springer Verlag, 55-567.
- De Marco T. (1979). *Structured Analysis and System Specification*. Prentice Hall.

- De Troyer O. & Leune C. (1997). WSDM: A User-centered Design Method for Web Sites. *Proceedings of the 7<sup>th</sup> International World Wide Web Conference*.
- de Vries E., Tiberhien A. & Guy P. (1995). Learning Processes and Knowledge Representation in the Design of Educational Hypermedia. *Proceedings of Hypermedia Design 95*.
- Diaz A., Isakowitz T., Maiorana V. & Gilbert G. (1995). RMC: A Tool to Design WWW Applications. *Proceedings of the 5<sup>th</sup> International World Wide Web Conference*.
- Dillenbourg P. & Self J. (1990). A Framework for Learner Modeling. *Technical Report AI-49. Lancaster University, England*.
- Eklund J. & Zeiliger R. (1996). Navigating the Web: Possibilities and Practicalities for Adaptive Navigational Support. *Proceedings of Second Australian World Wide Web Conference, AusWeb96*.
- Encarnação M. (1997). Concept and Realization of Intelligent Support in Interactive Graphics Applications. Ph.D. Thesis.
- Encarnação M. & Stork A. (1996). An Integrated Approach to User-centered Interface Adaptation. *Technical Report WSI-96-10, University of Tübingen*.
- Euromethod (1996). *Euromethod Framework*.
- Espinoza F. & Höök C. (1996). An interactive WWW Interface to an Adaptive Information System. *Proceeding of User Modeling '96 Conference*.
- Evans A., France R., Lano K. & Rumpe B. (1998). The UML as a Formal Modelling Notation. *Proceedings of the UML 98 Workshop, Bézivin J. & Muller P. (Eds.), LNCS 1618, Springer Verlag, 336-348*.
- Fink J. (1998). Implikationen aus dem Datenbank- und Transaktionsmanagement für anwendungs-orientierte Serversysteme zur Benutzermodellierung. *ABIS-98: Workshop on Adaptivity and User Modeling in Interactive Software Systems, FORWISS Report, 7-15*.
- Fink J., Kobsa A. & Nill A. (1997). Adaptable and Adaptive Information Access for all Users Including the Disabled and the Elderly. *User Modeling Proceedings of the Sixth International Conference, UM97, Jameson A., Paris C. and Tasso C. (Eds.), Springer Verlag Wien, 171-173*.
- Fink J. & Kobsa A. (2000). A Review and Analysis of Commercial User Modeling Servers for Personalization on the World Wide Web, *User Modeling and User Adapted Interaction, Kluwer Academic Publishers, 10 (2-3), 209-249*.
- Furuta R. & Stotts P. (1990). The Trellis Hypertext Reference Model. *Proceeding NIST Hypertext Standardization Workshop*.
- Gamma E., Helm R., Johnson R. & Vlissides J. (1995). *Design Patterns*. Addison Wesley.
- Garlatti S., Iksal S. & Kervella P. (1999). Adaptive On-line Information System by Means of a Task Model and Spatial Views. *Second Workshop on Adaptive Systems and User Modeling on the WWW, 59-66*.
- Garzotto F., Mainetti L. & Paolini P. (1995). Hypermedia Design Analysis. *Communications of the ACM, 8(38), 74-86*.

- Garzotto F., Paolini P. & Schwabe D. (1993). HDM: A Model-based Approach to Hypertext Application Design. *ACM Transactions of Information Systems*, 11(1), 1-26.
- Gellersen H-W. & Gaedke M. (1999). Object-oriented Web Application Development. *IEEE Internet Computing*, Jan-Feb, 60-68.
- Giangrandi P. & Tasso C. (1997). Managing Temporal Knowledge in Student Modeling. *User Modeling Proceedings of the Sixth International Conference, UM97*, Jameson A., Paris C. and Tasso C. (Eds.), Springer Verlag Wien, 415-426.
- Gogolla M. & Richters M. (2000). Definition of UML with UML and OCL: State of the Art (in German) *GROOM Workshop*.
- Graham J., Henderson-Sellers B. & Younessi H. (1997). *The OPEN Process Specification*. Addison Wesley.
- Greer J. (1996). *Student Modeling Tutorial*. User Modeling Conference.
- Greer J., McCalla G., Collins J., Kumar V., Bishop A. & Vassileva J. (1998). The Intelligent Helpdesk: Supporting Peer-Help in a University Course. *Proceeding of ITS '98*.
- Grønbaek K. & Trigg R. (1994). Design Issues for a Dexter-Based Hypermedia System. *Communications of the ACM 37(2)*, Grønbaek K. and Trigg R. (Eds.), 40-49.
- Grønbaek K. & Trigg R. (1996). Towards a Dexter-based Model for Open Hypermedia: Unifying embedded references and link objects. *Proceedings of the Hypertext '96 Conference*.
- Gutierrez J., Pérez T., Usandizaga I. & Lopistéguy P. (1996). HyperTutor: Adapting hypermedia systems to the user. *Proceedings of the 5<sup>th</sup> International Conference on User Modeling UM-96*.
- Halasz F. & Schwartz M. (1990). The Dexter Hypertext Reference Model. *NIST Hypertext Standardization Workshop*.
- Halasz F. & Schwartz M. (1994). The Dexter Hypertext Reference Model. *Communications of the ACM 37(2)*, Grønbaek K. and Trigg R. (Eds.), 30-39.
- Hardman L., Bulterman C. & van Rossum G. (1994). The Amsterdam Hypermedia Model. *Communications of the ACM 37(2)*, Grønbaek K. and Trigg R. (Eds.), 50-62.
- Harel D. (1987). Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3).
- Harel D. & Gery E. (1997). Executable Object Modeling with Statecharts. *IEEE Computer*, 30 (7), 31-42.
- Henderson-Sellers B. (1995). Who Needs an Object-oriented Methodology Anyway? *Journal of Object Oriented Programming*, 8 (6), 6-8.
- Henderson-Sellers B. & Firesmith D. (1997). Evaluating Third Generation OO Software Development Approaches. Submitted to *Information and Software Technology*.



- Hennicker R. & Koch N. (2000a). A UML-based Methodology for Hypermedia Design. *Proceedings of the Unified Modeling Language Conference, UML 2000*, Evans A. and Kent S. (Eds.). LNCS 1939, Springer Verlag, 410-424.
- Hennicker R. & Koch N. (2000b). Systematic Design of Web Applications. *Unified Modeling Language: Analysis, Design, and Development Issues*, Siau K. & Halpin T. (Eds.). Idea-Group Publishing, 1-20.
- Henze N. & Nejd W. (1999). Adaptivity in the KBS Hyperbook System. *Workshop on Adaptivity and User Modeling on the WWW*, International Conference on User Modeling UM'99.
- Hitz M. & Kappel G. (1999). *UML@Work*. dpunkt.verlag.
- Höök K. (1998). Evaluating the Utility and Usability of an Adaptive Hypermedia System. *Knowledge-Based Systems*, Elsevier, 10, 311-319.
- Höök K., Karlgren J. & Waern A. (1995). A Glass Box Intelligent Help Interface. *Proceedings of First Workshop on Intelligent Multimodal Interfaces*.
- Horvitz E. (1997). Agents with Beliefs: Reflections on Bayesian Methods for User Modeling. *Tutorial UM97: User Modeling Proceedings of the Sixth International Conference*.
- Huang X., McCalla G., Greer J., Neufeld E. (1991). Revising Deductive Knowledge and Stereotypical Knowledge in a Student Model. *User Modeling and User-Adapted Interaction Journal*, Vol 1, 87-115.
- IEEE Standard Glossary of Software Engineering Terminology (1983).
- IEEE (1987/1993). IEEE Standard for Software Project Management. 1058.1-1987 (R1993).
- IEEE (1988/1993). IEEE Standard for Software Review and Audits. 1028-1988 (R1993).
- IEEE (1989/1995). IEEE Standard Quality Assurance Plans (ANSI) together with IEEE Guide for Software Quality Assurance Planning (ANSI), 730-1989 and 730.1-1995.
- Iivari J. & Maansaari J. (1998). The Usage of Systems Development Methods: Are we Stuck to Old Practices? *Information and software Technology, Elsevier*, 40, 501-510.
- Isakowitz T., Stohr E. & Balasubramanian P. (1995). A Methodology for the Design of Structured Hypermedia Applications. *Communications of the ACM*, 8(38), 34-44.
- ISO/IEC 9126 (1991). International Standard: "Information technology – Software Product Evaluation – Quality Characteristics and Guidelines for their Use".
- ISPL (1999). *Information Services Procurement Library*. Ten Hagen & Stam Verlag.
- iVALS (2000). Internet Values and Lifestyles. Stanford Research Institute. <http://future.sri.com/>
- Jacobson I. (1992). *Object-oriented Software Engineering: A Use case driven Approach*. Addison Wesley.

- Jacobson I., Booch G., & Rumbaugh J. (1999). *The Unified Software Development Process*. Addison Wesley.
- Jacobson I. & Thomas D. (1995). Extensions: A Technique for Evolving Large Systems. *SIGS, Report on Object Analysis & Design*, 1 (5), 7-9.
- Jameson A. (1995). Numerical Uncertainty Management in User and Student Modeling: An Overview of Systems and Issues. *International Journal of User Modeling and User-adapted Interaction*, 5 (3), Kluwer Academic Publishers.
- Jameson A. (1998). User Modeling: An Integrative Overview. *Tutorial ABIS98:Workshop on Adaptivity and User Modeling in Interactive Software Systems*, FORWISS Report.
- Jörding T., Michel S. & Popella M. (1998). TELLIM – Ein System für adaptive multimediale Produktpräsentationen im World Wide Web. *ABIS-98:Workshop on Adaptivity and User Modeling in Interactive Software Systems*, FORWISS Report.
- Jones, C. (1990). *Systematic Development using VDM*. Prentice Hall International.
- Jungmann M. & Paradies T. (1997). Adaptive Hypertext in Complex Knowledge Domains. *Proceedings of the Flexible Hypertext Workshop (Hypertext '97)*.
- Kass R. & Finin T. (1988). The Need for User Models in Generig Expert System Explanations. *Technical Report University of Pennsylvania*.
- Kass R. (1989). Building a User Model Implicitly from a Cooperative Advisory Dialogue. *Proceedings of the Second International Workshop on User Modeling*.
- Kay J. & Kummerfield R. J. (1994). An Individualised Course for the C Programming Language. *Proceedings of the Second International WWW Conference '94*.
- Kay J. (1993). Reusable Tools for User Modelling. *Artificial Intelligence Review*, Vol 7, Academic Publishers, 241-251.
- Kay J. (1995). The UM Toolkit for Cooperative User Modeling. *User Modeling and User-Adapted Interaction (UMUAI)*, 4, Kluwer Academic Publisher, 149-196.
- Kobryn C. (1999). UML 2001: A Standardization Odyssey. *Communications of the ACM*, 42 (10), 29-37.
- Kobsa A., Müller D. & Nill A. (1994). KN-AHS: An Adaptive Hypertext Client of the User Modeling System BGP-MS. *Proceedings of Fourth International Conference on User Modeling, Mitre Corporation*.
- Kobsa, A. & Pohl W. (1995). The User Modeling Shell System BGP-MS. *User Modeling and User-Adapted Interaction*, 4 (2), 59-106.
- Kobsa A. & Wahlster W. (1989). *User Models in Dialog Systems*. Springer Verlag.
- Koch N. (1998). Towards a Methodology for Adaptive Hypermedia Systems Development. *Proceedings of the Sixth Workshop ABIS-98: Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen*, U. Timm, and M. Roessel (Eds.), 41-52, FORWISS.

- Koch N. (1999). A Comparative Study of Methods for Hypermedia Development. *Ludwig-Maximilians-University Munich, Institute of Computer Science*, Technical Report 9905.
- Koch N. (2000a). Hypermedia Systems Development based on the Unified Process. *Ludwig-Maximilians-University Munich, Institute of Computer Science*, Technical Report 0003.
- Koch N. (2000b). UML+OCL Specification of the Dexter Hypertext Reference Model. *Ludwig-Maximilians-University Munich, Institute of Computer Science*, Technical Report 0008.
- Koch N., Baumeister H., Hennicker R. & Mandel L. (2000). Extending UML to Model Navigation and Presentation in Web Applications. *Workshop on the UML and Modeling Web Applications*, UML'2000.
- Koch N. & Helmerich A. (2000). *Information Services Procurement for Web Engineering*. Ten Hagen & Stam Verlag.
- Koch N. & Mandel L. (1999) Using UML to Design Hypermedia Applications. *Ludwig-Maximilians-University Munich, Institute of Computer Science*, Technical Report 9901.
- Koch N. & Turk A. (1997). Towards a Methodical Development of Electronic Catalogues. *International Journal of Electronic Markets. University of St. Gallen, Switzerland*, Vol 7 (3), 28-31.
- König R. (1976). *Das Interview: Formen, Technik, Auswertung*. Kiepenheuer & Witsch. 10. Auflage.
- Kruchten P. (1998). *The Rational Unified Process: An Introduction*. Addison Wesley.
- Lang H.-W. (1988). Transitive Closure on the Instruction Systolic Array. *Proceedings of the International Conference on Systolic Arrays*, K. Bromley K., Kung S., Swartzlander E. (Eds.), 295-304.
- Lange D. (1996). An Object-oriented Design Approach for Developing Hypermedia Information Systems. *Journal of Organizational Computing and Electronic Commerce*, 6(3), 269-293.
- Lee H. Lee C. & Yoo C. (1998). A Scenario-based Object-Oriented Methodology for Developing Hypermedia Information Systems. *Proceedings of 31<sup>st</sup> Annual Conference on Systems Science*, Sprague R. (Ed.).
- Lieberman H. (1995). Letizia: An Agent That Assists Web Browsing. *International Joint Conference on Artificial Intelligence*, Montreal.
- Linard M. & Zeiliger R. (1995). Designing Navigation Support for an Educational Software. *Proceedings of 5<sup>th</sup> International Conference EWHCI'95*. Blumental, Gornostaev, Unger (Eds). LNCS 1015, SpringerVerlag, 63-78.
- Lowe D. & Hall W. (1999). *Hypermedia & the Web: An Engineering Approach*. John Wiley & Sons.
- Lowe D. & Webby R. (1998). The IMPACT Process Modelling Project: Work in Progress. *Workshop on Hypermedia Development. Hypertext '98*.

- Magglio P. & Barret R. (1997). How to Build Modeling Agents to Support Web Searchers. *User Modeling Proceedings of the Sixth International Conference, UM97*, Jameson A., Paris C. and Tasso C. (Eds.), Springer Verlag Wien, 5-16.
- Mandel L. & Cengarle M.V. (1999). On the Expressive Power of OCL. *Proceedings of the World Congress on Formal Methods (FM '99)*. LNCS 1708, Springer Verlag, 854-874.
- Mandl H. & Reinmann-Rothmeier G. (1997). Zukunft Cyberspace? Europas Weg in die globale Informationsgesellschaft. Presentation at the European Colloquium in Regensburg, Germany.
- Marinilli M., Micarelli A. & Sciarone F. (1999). A Case-based Approach to Adaptive Information Filtering for the WWW. *Second Workshop on Adaptive Systems and User Modelling on the World Wide Web*.
- Mathé N. and Chen J. (1996). User-centered Indexing for Adaptive Information Access. *User Modeling and User-Adapted Interaction*, 6 (2-3), 225-261.
- McTear M. (1993). User modelling for Adaptive Computer Systems: A Survey of Recent Developments. *Artificial Intelligence Review*, 7, 157-184, Kluwer Academic Publishers.
- Mislevy R. & Gitower D. (1995). The Role of Probability-based Inference in an Intelligent Tutoring System. *International Journal of User Modeling and User-adapted Interaction*, Kluwer Academic Publishers, 5 (3).
- Mladenic D. (2000). <http://www.cs.cmu.edu/afs/cs/project/theo-4/text/learning/www/pww/index.html>
- Murphy M. & McTear M. (1997). Learner Modeling for Intelligent CALL. *User Modeling Proceedings of the Sixth International Conference, UM97*, Jameson A., Paris C. and Tasso C. (Eds.), Springer Verlag Wien, 301-312.
- Murugesan S., Deshpande Y. Hansen S. & Ginige A. (1999). Web Engineering: A new Discipline for Development of Web-based Systems. *Proceedings of the First ICSE Workshop on Web Engineering, International Conference on Software Engineering*.
- Nakabayashi K., Maruyama M., Koike Y., Kato Y., Touhei H. & Fukuhara Y. (1997). Architecture of an Intelligent Tutoring System on the WWW. *Proceedings of the Eighth World Conference of the AIED Society*.
- Nanard J. & Nanard M. (1995). Hypertext Design Environments and the Hypertext Design Process. *Communication of the ACM*, August 1995, 38 (8), 49-56.
- Nanard J. & Nanard M. (1999). Toward and Hypermedia Design Pattern Space. *Hypertext '99 Workshop on Design Pattern in Hypermedia*.
- Nelson T. (1960). *Computer Lib/Dream Mashines*, 1974. Microsoft Press.
- Nielsen J. (1999). User Interface Directions for the Web. *Communications of the ACM*, 42 (1), 65-72.
- Nwana H. (1990). Intelligent Tutoring Systems: An Overview. *Artificial Intelligence Review*, 4, 251-277.

- Oestereich B. (1999). *Developing Software with UML: Object-Oriented Analysis and Design in Practice*. Addison-Wesley.
- Olsina L. (1998). Building a Web-based Information System Applying the Hypermedia Flexible Process Modeling Strategy. *1<sup>st</sup> International Workshop on Hypermedia Development, Hypertext '98*.
- Olsina L., Godoy D., Lafuente G. & Rossi G. (1999). Assessing the Quality of Academic Web Sites. *In New Review Hypermedia Multimedia Journal*, Taylor Graham Publishers, UK, Vol 5, 81-103
- Olsina L. (2000). Metodología Cuantitativa para la Evaluación y Comparación de la Calidad de Sitios Web. PhD. Thesis. UNLP, Argentina (in Spanish).
- OMG (2000). <http://cgi.omg.org/news/pr97/umlprimer.html>
- Paiva A., Self J. & Hartley R. (1995). Externalising Learner Models. *Proceedings of AIED95. AACE Publication*.
- Paiva A. & Self J. (1995). TAGUS: A User and Learner Modeling Workbench. *International Journal of User Modeling and User-adapted Interaction*, Kluwer Academic Publishers, 5 (3), 197-224.
- Paiva A., Self J. & Hartley R. (1995). Externalising Learner Models. *Proceedings of AIED95, AACE Publication*.
- Palvia P. & Nosek J. (1993). A Field Examination of System Life Cycle Techniques and Methodologies. *Information and Management*, 25(2), 73-84.
- Paolini P. & Garzotto F. (1999). Toward and Hypermedia Design Pattern Space. *Hypertext '99 Workshop on Design Patern in Hypermedia*.
- Pastor O., Insfrán E. Pelechano V., Romero J. & Merseguer J. (1997). OO-Method: An OO Software Production Environment Combining Conventional and Formal Methods. *Proceedings of CAISE '97*, LNCS 1250, Springer-Verlag, 144-158.
- Paterno F. & Mancini C. (1999). Designing Web Interfaces Adaptable to Different Types of Use. *Proceedings of the Workshop Museums and the Web*. <http://www.acrhimuse.com/mw99/>
- Pezdiric L. (1999). Entwicklung einer Datenbankunterstützung und eines Kommunikationsforums für ein webbasiertes Tutoring-System. *Diplomarbeit, Ludwig-Maximilians-Universität München*.
- Pohl W. (1999). Logic-Based Representation and Reasoning for User Modeling Shell Systems. *User Modeling and User-Adapted Interaction International Journal*, Kluwer Academic Publishers, Vol 9 (3), 217-282.
- Pohl W. & Höhle J. (1997). Mechanisms for Flexible Representation and Use of Knowledge in User Modeling Shell Systems. *User Modeling Proceedings of the Sixth International Conference, UM97, Jameson A., Paris C. and Tasso C. (Eds.)*, Springer Verlag Wien, 403-414.
- Preece J., Rogers Y., Sharp H., Benyon D., Holland S. & Carey T. (1994). *Human-Computer Interaction*. Addison-Wesley.

- Ragnemalm E (1995). Student Diagnosis in Practice; Bridging the Gap. *User Modeling and User-Adapted Interaction (UMUAI) International Journal*, 5, Kluwer Academic Publishers 93-116.
- Ramscar M., Pain H. & Lee J. (1997). Do We Know What the User Knows, and Does It Matter? The Epistemics of User Modelling. *Proceedings of the International Conference of User Modeling 97*. Jameson A. & Paris C. (Eds.) Springer Verlag, 429-431.
- Rational Unified Process (2000). Rational Software. <http://www.rational.com/rup>
- Rich E. (1979). User Modeling Via Stereotypes. *Cognitive Science* 3, 329-354.
- Richters M. & Gogolla M. (1999). A Metamodel for OCL. *Proceedings of the Conference The Unified Modeling Language beyond the standard (UML'99)*. LNCS 1723, Springer Verlag, 156-171.
- Ritter S. (1997). PAT Online: A Model-tracing Tutor on the World Wide Web. *Proceedings of Workshop on Intelligent Systems on the World Wide Web, 8<sup>th</sup> Conference of the AIED Society* .
- Robilliard P. (1999). The Role of Knowledge in Software Development. *Communications of the ACM*, 42 (1), 87-92.
- Rössel M. (1998). Pragmatische Benutzermodellierung im adaptiven multimedialen Präsentationssystem AMPres. *ABIS-98:Workshop on Adaptivity and User Modeling in Interactive Software Systems*, FORWISS Report.
- Rossi G. (1996). OOHDM: Object-Oriented Hypermedia Design Method. PhD thesis, PUC-Rio, Brazil (in Portuguese).
- Rossi G., Schwabe D. & Garrido A. (1996). Towards a Pattern Language for Hypermedia Applications. *Proceedings of the 3<sup>rd</sup> Annual Conference on Pattern Languages of Programs 96*.
- Rossi G., Schwabe D., & Lyardet F. (2000). Web Applications Models are More than Conceptual Models. *Proceedings of the Web Engineering Workshop at WWW9*.
- Rumbaugh J. (1995). What is a Method? *Journal of Object Oriented Programming*, 8(6) 10-16,26.
- Sano D. (1996). *Designing Large-Scale Web Sites: A Visual Design Methodology*. Wiley Computer Publishing.
- Sauer S. & Engels G. (1999). Extending UML for Modeling Multimedia applications. *Proceedings of the IEEE Symposium of Visual Languages – VL 99*, IEEE Computer Society.
- Scharl A. (1999). A Conceptual, User-Centric Approach to Modeling Web Information Systems. *Proceedings of 5<sup>th</sup> Australian World Wide Web Conference (AusWeb 99)*. Southern Cross University Press, 33-49.
- Schneider G. & Winters J. (1998). *Applying Use Cases: A Practical Guide*. Addison-Wesley, Object Technology Series.
- Schneiderman B. (1998). *Designing the User Interface: Strategies for effective Human-Computer Interaction*. Addison Wesley.

- Schuhbauer H. (1998). Das Benutzerprofil in einem Freizeitberatungszentrum. *Proceedings of ABIS-98: Workshop on Adaptivity and User Modeling in Interactive Software Systems*, FORWISS Report.
- Schwabe D. & Almeida Pontes R. (1998). OOHDM-WEB: Rapid Prototyping of Applications in the WWW. Technical Report PUC-RioInf.MCC-08/98.
- Schwabe D. & Rossi G. (1998). Developing Hypermedia applications using OOHDM. *Proceedings of Workshop on Hypermedia development Process, Methods and Models, Hypertext '98*.
- Self J. (1988). The Use of Belief Systems for Student Modelling. *Proceedings of the First European Congress on Artificial Intelligence and Training*.
- Self J. (1991). Formal Approaches to Student Modeling. *Technical Report AI-59. Lancaster University, England*.
- Self J. (1996). Deconstructionist Student Models in the Computer-Based Learning of Science. *Proceedings of Computer-Aided Learning and Instruction in Science and Engineering*. LNCS 1108, Springer Verlag, 27-36.
- Selic B. (1999). Using UML in the Real-Time Domain. *Communications of the ACM*, 42 (10), 46-54.
- Shuell T. (1992). Designing Instructional Computing Systems for Meaningful Learning. *Adaptive Learning Environments, Foundations and Frontiers*, M. Jones and P. Winne (Eds.), Springer-Verlag, 19-53.
- Siau K. & Cao Q. (2001). Unified Modeling Language (UML) – A Complexity Analysis. *Journal of Database Management*, 26-34, to appear.
- Sleeman D. & Brown J.S. (1982). *Intelligent Tutoring Systems: Introduction*. Academic Press, London, 1-12.
- SmexWeb: Student Modelling Exercising on the Web (1998).  
<http://pst1.pst.informatik.uni-muenchen.de:8000>
- Sommerville I. (1982). *Software Engineering*. Addison Wesley.
- Spivey J. (1992). *The Z Notation: A Reference Manual*. Series in Computer Science. Prentice Hall International, second edition.
- Stein A., Gulla J. & Thiel U. (1997). Making Sense of User's Mouse Clicks: Abductive Reasoning and Conversational Dialogue Modeling. *User Modeling Proceedings of the Sixth International Conference, UM97*, Jameson A., Paris C. and Tasso C. (Eds.), Springer Verlag Wien, 89-100.
- Strachan L., Anderson J., Sneyby M. & Evans M. (1997). Pragmatic User Modelling in a Commercial Software System. *User Modeling Proceedings of the Sixth International Conference, UM97*, Jameson A., Paris C. and Tasso C. (Eds.), Springer Verlag Wien, 189-200.
- Streitz N. (1990). Hypertext: Ein innovatives Medium zur Kommunikation von Wissen. *Hypertext und Hypermedia*, Gloor P. and Streitz N. (Eds.), Springer-Verlag.

- Thomas, C., and Fischer, G. (1996). Using Agents to Improve the Usability and Usefulness of the World Wide Web. In *Proceedings of the 5<sup>th</sup> International Conference on User Modeling*, 5-12.
- Thomson J., Greer J. & Cooke J. (1998). Algorithmically Detectable Design Patterns for Hypermedia Collections. *Proceedings of Workshop on Hypermedia development Process, Methods and Models, Hypertext '98*.
- Thüring M., Hannemann J. & Haake J. (1995). Hypermedia and Cognition: Designing for Comprehension. *Communications of the ACM*, 38 (8), 57-66.
- Tiller T. (1998). Eine Hypertext-Struktur für abgeschlossene adaptive Systeme. *Diplomarbeit, Ludwig-Maximilians-Universität München*.
- Timm U. & Rosewitz M. (1998). Benutzermodellierung in der Elektronischen Produktberatung – Konzept und prototypische Realisierung in einer On-line Umgebung. *Proceedings of ABIS-98: Workshop on Adaptivity and User Modeling in Interactive Software Systems*, FORWISS Report.
- Tochtermann K. (1994). Ein Modell für Hypermedia. Ph.D. Thesis. Universität Dortmund.
- Tochtermann K. and Dittrich G. (1996). The Dortmund Family of Hypermedia Models. *Journal of Universal Computer Science*, 2(1), Springer Verlag.
- UML Version 1.3 (1999). Unified Modeling Language. The Object Management Group. <http://www.omg.org>
- van Ossenbruggen J. and Eliëns A. (1995). The Dexter Hypertext Reference Model in Object-Z. <http://www.cs.vu.nl/~dejavu/papers/dexter-full.ps.gz>
- Vassileva J. (1990) A Classification and Synthesis of Student Modeling Techniques in Intelligent Computer-assisted Instruction. *Proceedings of ICCAL '90 Computer Assisted Learning*. Norrie D., Six H.-W.(Eds.). LNCS 438, Springer Verlag, 202-213.
- Vassileva J. (1992). A Three-dimensional Perspective on the Current Trends in Student Modeling. *Proceedings of EW '92- East-west Conference on Emerging Technologies in Education*, 315-320.
- Vassileva J. (1994). A Practical Architecture for User Modeling in a Hypermedia-Based Information System. *Proceedings of the 4<sup>th</sup> International Conference on User Modeling*, 115-120.
- Vassileva J. (1995). Reactive Instructional Planning to Support Interactive Teaching Strategies. *Proceedings of the 7<sup>th</sup> World Conference on AI and Education*, AACE, 334-342.
- Vassileva J. (1996). A Task-centered Approach for User Modelling in a Hypermedia Office Documentation System, *User Modeling and User-Adapted Interaction Journal*, 6, 185-223. Kluwer Academic Publishers.
- Vassileva J. (1997). Dynamic Course Generation on the WWW. *Proceedings of Workshop Intelligent Educational Systems on the World Wide Web in AI-ED '97: Eighth World Conference on Artificial Intelligence in Education*.



- Vassileva J. & Wasson B. (1996). Instructional Planning Approaches: from Tutoring Towards Free Learning. *Proceedings of EuroAIED '96*, 1-8.
- Wahrshall S. (1962). Theorem on Boolean Matrices. *J. Assoc. Comput. Mach.* 9, 11-12.
- Warmer J. & Kleppe A. (1999). *The Object Constraint Language: Precise Modeling with UML*. Object Technology Series. Addison-Wesley.
- Wasson B. (1990). *Determining the focus of instruction: Content planning for intelligent tutoring systems*. Doctoral Thesis, Department of Computer Science, University of Saskatchewan.
- Waters J. (2000). Getting Personal on the Web. *Application Development Trends, Communications Publication*, 7 (5), 25-32.
- Weber G. & Specht M. (1997). User Modeling and Adaptive Navigation Support in WWW-Based Tutoring Systems. *Proceedings of the Sixth International Conference of User Modeling, UM'97*, Jameson A., Paris C. and Tasso C. (Eds.), Springer Verlag, 289-300.
- Weidenhaupt K., Pohl K., Jarke M. & Haumer P. (1999). Scenarios in System Development: Current Practice. *IEEE Software*, 2, 34-45.
- Wieringa R., Dubois E. & Huyts S. (1997). Integrating Semi-formal and Formal Requirements. *Proceedings of Conference on Advanced Information Systems Engineering, CAiSE '97*, Olivé A. & Pastor J. (Eds.), 19-32.
- Wilkinson N. (1995), *Using CRC Cards: An Informal Approach to Object-oriented Development*. SIGS Books.
- Wirfs-Brock R., Wilkerson B. & Wiener L. (1993). *Object-Oriented Software Design*. Hauser-Prentice Hall.
- Wirsing M. & Knapp A. (1996). A Formal Approach to Object-Oriented Software Engineering. *Proceedings of International Workshop of Rewriting Logic and Its Applications*, Meseguer J. (Eds.) and *Electronic Notes Theoretical Computer Science, Elsevier*, Vol 4, 321-359, revised version.
- Wu H, Houben G.-J. & De Bra P. (1998). AHAM: A Reference Model to Support Adaptive Hypermedia Authoring. *Proceedings of InfWet 98*.
- Wu H., Houben G.-J., & De Bra, P. (1999). Authoring Support for Adaptive Hypermedia Applications. *Proceedings of the ED-MEDIA Conference, AACE*, 364-369.



## **The Author**

Nora Parcus de Koch was born in 1951 in Buenos Aires, Argentina. She received degrees in Computer Science at the University of Buenos Aires in 1974 and 1985. From 1975 to 1979 she worked at the Central Bank of Argentina as application developer. She was a teaching assistant first and a professor later at the University of Buenos Aires until moving to Germany in 1985. From 1986 to 1994 she worked as a consultant for several German and Argentinean companies. She started with her PhD Thesis during her work as researcher at the chair of Martin Wirsing at the Ludwig-Maximilians-Universität (LMU) München (1995-1998). Currently she works at F.A.S.T. Gesellschaft für angewandte Softwaretechnologie mbH and is involved in a variety of both, national and European projects focusing on Web engineering, visual modeling and development processes. At the same time she is a guest researcher at the LMU. Her publications are published under the name Nora Koch (<http://www.pst.informatik.uni-muenchen.de/~kochn>).