Vrije Universiteit Brussel

FACULTEIT VAN DE WETENSCHAPPEN
Departement Computerwetenschappen
Web & Information Systems Engineering

Designer Specified Self Re-organizing Websites

Proefschrift ingediend met het oog op het behalen van de graad van Doctor in de Wetenschappen

Sven Casteleyn

Academiejaar 2004 - 2005

Promotor: Prof. Dr. Olga De Troyer

VRIJE UNIVERSITEIT BRUSSEL • SCIENTIA VINCERE TENEBRAS •

Abstract

The days where a web site consisted of a small amount of linked pages lie far behind us. Today, web sites are complex applications, containing large amounts of constantly changing information and functionality. To avoid usability problems typically associated with large ad hoc designed web sites, a systematic, well structured approach is required. Web site design methods address this need by separating the different design concerns involved in web site creation: requirements analysis, data design, navigation design, presentation design and implementation. By focussing on one aspect at a time, the designer is better able to cope with the intricacy of web site design, thereby improving the quality and usability of the resulting web site. However, even with the use of web design methods, the design of web sites is not an easy task. Due to the large variety in targeted visitors, each with different goals and intentions, and the diversity of the offered information and functionality, devising an efficient and easy to grasp site organization and navigation structure proves difficult. Taking into account web site usage information could help the designer to gain a better insight in the browsing behaviour of the users and to provide the best-suited design. However at design time this information is usually not available. Manual analysis of web site access logs is already applied, possibly resulting in manual changes or a complete redesign of the web site. The approach suggested in this dissertation goes one step further: to offer the web designer, during design, the ability to anticipate and react upon web site usage information. The result is adaptive web sites: sites that automatically improve their organization and presentation based on user access data. It is argued in this dissertation that the use of adaptation strategies, which are (design) specifications of how a web site can adaptively change (at runtime) based on user access information (from all users), aids the designer to improve the usability of web sites. Using adaptation strategies, a designer can anticipate runtime browsing behaviour, validate certain design decisions, automatically select between design alternatives and help the web site owner to better achieve business goals. The work is presented in the context of WSDM, an existing web design method. To support adaptive behaviour, an adaptation specification language has been defined, which allows the designer to specify, at design time, the adaptive behaviour that is allowed at runtime. The adaptation specification language is exemplified with three useful adaptation strategies, each illustrating one of the aforementioned benefits. To validate our ideas, a prototype implementation to support WSDM design and implementation, with adaptation support, was performed. The prototype was used to implement a case study, which demonstrates the applicability and effectiveness of designer specified adaptation strategies to automatically re-organize the web site (navigation) structure and organization.

Acknowledgement

First of all, I would like to thank my parents, **Dennis and Lieve Casteleyn**, for their motivation and support throughout my studies, from kindergarten to university. Without their fine education and thorough support, I would probably not have taken up the eagerness or motivation to learn and study. Almost insignificant compared to the aforementioned matters, their practical support while finishing my phd also bought me invaluable time needed to finish in time. *Thank you Dennis and Lieve.* In the same breath, I would like to thank **my sister Ilse** and her husband **Geert Drieghe**, for supporting me on practical issues while working on my phd. And **Tine Drieghe** for bringing new sunshine in my life. *Thank you Ilse, Geert and Tine.* I also owe my grandparents, whose practical support and belief in me indeed granted me the necessary strength and time to carry on. *Thank you Lucien, Maria, Emma and Jerom.*

Second, my greatest gratitude goes out to my promoter, Prof. Dr. **Olga De Troyer**. She deserves credit for so many things it doesn't even seem possible to name them all. She picked me up when I was ready to leave my phd and gave me the exact right motivation and support I needed in order to continue. Without that support at the right time, I would not have made a phd in the first place. Furthermore, she gave me her excellent assistance throughout my work. In my most productive times she gave up some of her own free time to advice me, and granted me much needed patience when I had the most difficult time in my life. *Thank you Olga.*

Next, I would like to thank the members of my jury, Prof. Dr. **Jaime Gómez,** Prof. Dr. **Jean Vanderdonckt**, Prof. Dr. **Theo D'Hondt** and Prof. Dr. Ir. **Geert-Jan Houben** first of all to be in my jury and secondly for providing me with good pointers to further improve the quality of my text. I look forward to cooperate with all of them in the future. *Thank you Jaime, Jean, Theo and Geert-Jan.*

Special thanks go out to **Irene Garrigós**, who came from Alicante to cooperate with me for three months. It proved to be a very fruitful period resulting in valuable research, and more significantly, the right momentum to begin, continue and finish the work on my phd. Her presence here made a lasting impression and put a stamp on my work and life. *Thank you Irene.*

My greatest gratitude also goes out to my friend and colleague **Peter Plessers**, with whom I had an excellent cooperation which stowed both our works forward. I wish him to obtain his phd soon, he most certainly deserves it. *Thank you Peter.*

Furthermore, I want to thank my friend and colleague **Wolfgang De Meuter**, who gave the phrase "Don't worry, we're sending the Wolf" a whole new meaning. He helped me on miscellaneous issues of the Adaptation Specification Language and was there with valuable advice I asked and didn't ask for. He lifted the quality of this phd by allowing me to browse in the vast library of knowledge stored in his mind. *Thank you Wolf.*

I would also like to thank my two thesis students **Tom Vleminckx** and **Stijn Coolbrandt** for their help on the implementation needed to validate my work. *Thank you Tom and Stijn* (where did we hear that before?).

My colleagues **Anne Defaweux** and **Thomas Cleenewerck** deserve a special mention of thanks, respectively for their pointers on statistics and domain specific languages. I would like to thank all my colleagues, and in particular those of WISE, the research lab I worked for: **Wesley Bille, Bram Pellens, Frederic Kleinermann, Raül Romero** and **Abed Mustaha.**

Last, but not least, all my friends deserve their fair amount of gratitude. Some of them just for being my friend, some of them for their extra support which' mentioning here would lead us way too far. Thanks especially to **Philippe Demaecker** (better known as **Philou**) and **Joyce Van Duyvenbode**, and also to **Ellie D'Hondt**, **Freek Brysse**, **Tom Toutenel, Anneke Laurent and Bart Hompes, Hans Van Ingelgom, Geert Dangreau, Werner Van Belle**, **Gerrit Cornelis**, **Iris Plancke, Wim Goedefroy, Tom Tourwe, Ismael Sanz** and so many others I cannot thank them all explicitly.

Finally, I would like to thank all the people I unwittingly forgot to mention here. Many people contributed in some way to this phd and I thank all of them.

Contents

Chater 1: Introduction....13
1.1 Research Context....14
1.2 Problem Statement....16
1.3 Approach....17
1.4 Advantages....19
1.5 Contributions....20
1.5.1 Major Contributions....20
1.6 Outline....21
Chapter 2: Background and Related Work....23
2.1 Hypermedia....24
2.1.1 History....24
2.1.2 Definitions....27
2.1.3 Hypermedia Reference Models....28
2.1.4 The World Wide Web....31
2.2 Adaptive Hypermedia....31
2.2.1 Definitions....32
2.2.2 Adaptive, Adaptable, Dynamic....32
2.2.3 Adapt to what?....33
2.2.4 What to adapt?....34
2.2.5 AHAM....35
2.3 Web Site Design Methods....37
2.3.1 OOHDM / SHDM....37
2.3.2 WebML....39
2.3.3 Hera....42
2.3.4 OO-H....45
2.3.5 UWE....46
2.3.6 Other Web Design Methods....48
2.4 Situating this dissertation in the research field....48
Chapter 3: WSDM Informal Description....51
3.1 Introduction....52
3.2 Mission Statement....54
3.3 Audience Modelling....54
3.3.1 Audience Classification....54
3.3.2 Audience Characterization....56
3.4 Conceptual Design....56
3.4.1 Task & Information Modelling....56
3.4.2 Navigation Design....61
3.5 Implementation Design....63
3.5.1 Site Structure Design....63
3.5.2 Presentation Design....64
3.5.3 Data Source Mapping....69
3.6 Implementation Generation....69
3.6.1 High Level Transformation Mapping....70
3.6.2 Model Integration....70
3.6.3 Implementation Mapping....71
3.6.4 Data Source Mapping....71
3.6.5 Query Execution....71
3.7 Extensions to WSDM....71
Chapter 4: WSDM formalization....73

4.1 Mission statement 74
4.2 Audience Modelling 75
4.3 Conceptual Design 78
4.3.1 Task and Information Modelling 78
4.3.2 Navigation Design 80
4.4 Implementation Design 87
4.4.1 Site Structure Design 87
4.4.2 Presentation design 88
Chapter 5: WSDM Adaptation Model 95
5.1 Web site Overlay Model 96
5.2 Basic Information Storage Operations 99
5.3 Basic Model Transformation Operations 101
5.3.1 Operations on nodes 102
5.3.2 Operations on chunks 103
5.3.3 Operations on links 104
5.3.4 Operations on Pages 105
5.4 Adaptation Specification Language (ASL) 107
5.4.1 General Overview 108
5.4.2 Adaptation Strategies and Policies 109
5.4.3 Rules and Statements 113
5.4.4 Scripts 116
5.4.5 Expressions 117
5.4.6 (Tracking) Variables and Set Variables 118
5.4.7 Set Expressions 119
5.4.8 Native Operations and Functions 122
5.4.9 Remaining BNF production rules 124
Chapter 6: Adaptation Strategies 125
6.1 Promotion and demotion 126
6.1.1 Promotion 126
6.1.2 Demotion 130
6.2 Re-ordering sequential information (gathering) 134
6.2.1 Principle 134
6.2.2 Example 134
6.2.3 ASL Specification 135
6.3 Validation of Audience Driven Navigation Structure 138
6.3.1 Principle 138
6.3.2 Example 143
6.3.3 ASL Specification 145
Chapter 7 Validation 149
7.1 WSDM Validation 150
7.1.1 WSDM Ontology 150
7.1.2 WSDM Implementation Generation 151
7.2 Adaptation Specification Language Validation 152
7.3 Case Study 154
7.3.1 Subject 155
7.3.2 WSDM design 155
7.3.3 Adaptation Strategies 163
Chapter 8: Conclusion 171
8.1 Summary 172
8.2 Contributions and Achievements 174
8.3 Limitations 174
8.4 Future Work 175
Appendix 1: ASL BNF Specification 181
References 187

List of Figures

Figure 2-1 Overview of the Dexter Reference Model 28
Figure 2-2 Overview of the AHAM Reference Model 35
Figure 2-3 OOHDM/SHDM Overview 38
Figure 2-4 The WebML Development Process 40
Figure 2-5 Hera Overview 43
Figure 2-6 OO-H Overview 46
Figure 2-7 UWE Overview 47
Figure 3-1 WSDM Overview 53
Figure 3-2 Audience Class Hierarchy for the Conference Review Web site 55
Figure 3-3 Task Categories supported in WSDM 58
Figure 3-4 Task Model for the “Assign Papers” Task Using the CTT Notation 58
Figure 3-5 Example Object Chunk “Paper Detail” 59
Figure 3-6 Basic Navigation Model for the Conference Review Web site 62
Figure 3-7 Elaborated Author Track for the Conference Review Web site 63
Figure 3-8 Site Structure Design for the Author Audience Track of the Conference Review Web site 64
Figure 3-9 Example Header-LeftSideBar Template from the Conference Review Web site 67
Figure 3-10 Example Page Presentation Design for the Paper Detail page of the Conference Review Web site .. 69
Figure 3-11 Implementation Generation Overview: Transformation Pipeline 70
Figure 5-1 Schematic Overview of the Adaptation Specification Language 109
Figure 6-1 Promotion of Node 1-2-1 (closer to root Node1) 127
Figure 6-2 Simplified Example Navigation Model for a Telephone Company 128
Figure 6-3 Demotion of Node1-1 (away from root Node1) 130
Figure 6-4 Simplified Example Navigation Model for a University Web site 131
Figure 6-5 Navigation Model for the University Example after Promotion 133
Figure 6-6 Example Information Gathering Process for a Bookstore 135
Figure 6-7 Schematic Representation of a Concrete Node Re-ordering 137
Figure 6-8 Simplified Navigation Model for (part of) the NASA Web site 143
Figure 6-9 Information Access Matrix 144
Figure 7-1 WSDM Implementation Architecture supporting ASL 153
Figure 7-2 Audience Class Hierarchy for the WISE web site 156
Figure 7-3 Object Chunk “ShowCourses” 157
Figure 7-4 Object Chunk "ShowCourseDetail" 158
Figure 7-5 Overall Navigation Model for the WISE Web site 159
Figure 7-6 Detailed Navigation Model for the Researcher Audience Track 160
Figure 7-7 Detailed Navigation Model for the Student Audience Track 161
Figure 7-8 Information Template for the WISE Web site Case Study 162
Figure 7-9 Simplified Page Presentation Design for the Course Page 163
Figure 7-10 Audience Class Matrix for Students 168
Figure 7-11 Audience Class Matrix for Researchers 168

List of Acronyms

ACM:	Association for Computing Machinery
AHAM:	Adaptive Hypermedia Application Model
AHS:	Adaptive Hypermedia System
ASL:	Adaptation Specification Language
ASP:	Active Server Pages
AWT:	Abstract Windowing Toolkit
BNF:	Backus Naur Form
CTT:	Concurrent Task Tree
ELF:	Evolutionary List Files
ER:	Entity Relationship [modelling]
HCI:	Human Computer Interaction
HES:	Hypertext Editing System
IDE	Instructional Design Environment
KMS:	Knowledge Management System
LOT:	Lexical Object Type
NAD:	Navigation Access Diagram
NLOT:	Non Lexical Object Type
NLS:	oN Line System
ORM:	Object Role Modelling
OWL:	Web Ontology Language
OT:	Object Type
PRIDE:	Personalized Retrieval, Indexing and Documentation Evolutionary
RDF:	Resource Description Framework
RDFS:	Resource Description Framework Schema
SMIL:	Synchronized Multimedia Integration Language
SQL:	Structured Query Language
TIME:	Timed Interactive Multimedia Extension
UID:	Unique Identifier
UML:	Unified Modelling Language
W3C:	World Wide Web Consortium
WAP:	Wireless Access Protocol
WebML:	Web Modeling Language
WIS:	Web Information System
WML:	Wireless Markup Language
WSDM:	Web Site Design Method
WWW:	World Wide Web
XML:	eXtensible Markup Language
XPath:	XML Path Language
XSLT:	eXtensible Stylesheet Language Transformations

Chapter 1: Introduction

"The mistakes are all waiting to be made"
(Savielly Grigorievitch Tartakower, chess grandmaster, on the opening position)

1.1 Research Context

The World Wide Web (WWW) was conceived in 1991 at CERN, as a research project to allow information sharing and exchange among different research groups. Originally aimed at the physics community, it quickly spread to other research disciplines, and subsequently conquered the world by storm. Today, the worldwide Internet community is estimated to 888 million[1, 2], and the total amount of available web sites rose to 65 million[3, 4]. The World Wide Web has changed the way we communicate, consult, share and distribute information. It penetrated virtually every aspect of our modern lives. It has an impact on the way we work, educate, drive commerce, communicate with the government, etc...

Where the first web sites consisted of a single (static) page, or a few linked pages, current web sites are large volatile applications, offering both static and dynamic information and functionality. Driven by the constant demand for functional advances of a growing population of users and content suppliers, and more powerful hardware becoming readily available, technology behind the WWW has evolved rapidly over the years. HTML transformed from a small mark-up language to support text-based linked pages, to a large and complex mark-up language supporting multimedia elements (e.g. images), forms (e.g. input fields), presentational aspects (e.g. text and background colours, fonts, styles, classes, …)[5], and various other useful and less useful additions (i.e. meta-information, tables, image maps, frames, …). To support programming logic as well as a more flexible way to (dynamically) generate web pages and their content, HTML was complemented with the Common Gateway Interface and scripting languages. The infestation of HTML with more and more purely presentational tags and the problems inherent to mixing content and presentation led the W3C[6] to devise a way to separate content and presentation (again). XML was designed as a flexible general-purpose mark-up language, able to describe data in a structured way; style sheets were used to add presentation styles to an XML document. Soon after, HTML was (gradually) replaced by XHTML, a reformulation of (traditional) HTML using XML technology. Arrival of XML also triggered a variety of assistive technologies to be defined, aimed at aiding the web designer in creating and handling XML documents (e.g. XLink, XPath, XQuery, XSLT, etc).

Recent developments in the WWW field are mostly related to Tim Berners-Lee's vision of the *Semantic Web*: "*an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation*" [Berners-Lee et al, 2001]. To realise this dream, the current WWW was extended to explicitly associate semantics (i.e. in the form of metadata) with the data available on a web site. XML is used as a mark-up language to describe these semantic annotations. Built on XML, the W3C has defined a variety of more specific description languages. Two representative ones are the Resource Description Framework (RDF), designed specifically to represent (meta-data about) resources on the web, and the Ontology Web Language (OWL), built on top of RDF to describe the meaning of terms and the relations between those terms.

[1] http://www.internetworldstats.com/ (access date 20 June 2005)
[2] This is an average increase of 146% compared to 2000.
[3] http://www.netcraft.com/ (access date 20 June 2005)
[4] One million web sites were reported online in April 1997; in February 2000 the 10 million barrier was broken, and today 64 808 485 web sites are reported online.
[5] At the time much to the dislike of academia, who thought the (only) intention of HTML should be to describe the *structure* of a document, not the *presentation*. They were right, as it appeared later, when the W3C attempted once and for all to separate content and presentation with the introduction of XML.
[6] http://www.w3c.org/ (access date 27 June 2005)

In a reaction to the dazzling growth of the WWW, with constantly evolving demands from both web users and developers, ample attention has been paid to technological advances. However, with web sites growing larger, more data and functionality to be managed, technology becoming increasingly complex and more and more aspects involved in web development (i.e. multi-platform support, context awareness, accessibility, localization, …) serious manageability and usability problems emerged.

In an attempt to keep web sites manageable, (poor) solutions were proposed by the industry to support the complexity of creating and maintaining large web sites. Web authoring tools (e.g. Microsoft Frontpage, Macromedia Dreamweaver, …) mostly assist the designer in easily creating (the look and feel of) simple web pages and web sites (using a WYSIWYG approach). However, they fail to support management of large amounts of data and do not provide enough support to effectively maintain a large web site. Content Management Systems (e.g. Zope[7], SparkPlug CMS[8], …) do provide some support for data and work flow management, yet they are primarily focused on the management of content. Web authoring tools as well as Content Management Systems do not provide the web designer any support on *how to design* the web site: which information should be provided and how should it be organized on pages, which navigation structures must be provided and what is the most appropriate presentation style, how do you take into account the expectations of prospective user(s) when designing the web site, etc.

Web sites lacking a systematic underlying design suffer from enormous usability problems [Nielsen, 1992]. Inconsistent or illogical information and site organization makes it hard for visitors to form a mental model of the web site; they experience difficulties in locating the information they are looking for and feel 'lost in hyperspace'. With competitor's web sites only one click away, visitors quickly abandon a site if they don't find the information they are looking for.

The academic community is tackling the abovementioned issues of design, maintenance and usability of large scaled web sites since as early as 1995, when the first web design methods[9], RMM [Isakowitz et al, 1995] and OOHDM [Schwabe et al, 1995], emerged. Steadily, new improved design methods were introduced (e.g. WSDM [De Troyer and Leune, 1998], OO-H [Gómez et al, 2000], UWE [Koch, 2001] , WebML [Ceri et al, 2002], Hera [Frasincar et al, 2002], …). Although differing in approach and techniques, all these methods separate design concerns by distinguishing a data, navigation and implementation design. Some methods also provide tool support and implementation generation (i.e. most notably WebML and OO-H). From the beginning of 2000 there was enough momentum to speak of a Web Engineering discipline: a discipline "*concerned with establishment and use of sound scientific, engineering and management principles and disciplined and systematic approaches to the successful development, deployment and maintenance of high quality Web-based systems and applications.*" [Murugesan et al, 2001].

To better accommodate the individual user, personalization of web sites has been introduced and studied. Extensive research has been performed to support personalization, primarily in the fields of e-learning and user assistance applications. These two particular domains lend themselves very well to personalization because supporting the user to achieve a certain goal (e.g. a particular learning goal; solving a particular problem) is clearly the priority of these systems. More generally, in the context of web sites, personalization is used to perform limited content adaptation (e.g. setting certain domain attribute values, showing/hiding attributes), limited presentation adaptation (e.g. switch between pre-defined layout configurations, altering link appearance, transforming images) and recommendation

[7] http://www.zope.org/ (access date: 20 June 2005)
[8] http://www.13amp.net/index.html (access date: 20 June 2005)
[9] In fact, these methods initially were *hypermedia* design methods, but were focused on the web.

type of personalization (e.g. in an online shop, when buying an item, related items bought by users who bought the same item are recommended). Most web design methods (e.g. OO-H, Hera, UWE, WebML, OOHDM) support some form of personalization. Personalization is done by gathering information about the current user and storing this information in a user model. The information stored in the user model is subsequently used in conditions specified in the design models and/or in rules. Specifying *how* the information to be stored in the user model is obtained is mostly manual (e.g. in the form of questionnaires). Also, specifying *how* the actual personalization is performed is done manually (e.g. in case of recommendations, the designer needs to implement an algorithm to calculate recommendations).

1.2 Problem Statement

The systematic use of web design methods allows simplifying the creation of web sites by providing abstraction from the implementation and by gradually considering each aspect of the design. Some design methods offer an explicit design methodology (e.g. the audience driven approach of WSDM), which may even more improve the consistency and usability of the resulting web site. Most methods also provide some facilities for personalization.

However, no matter the skills and good intentions of the web designer, a design process is an engineering task, and thus inherently requires creativity and may be error-prone. Design errors are nearly unavoidable, and will reduce the web site's usability. Even without making (human) errors, the designer may need to decide upon different design alternatives. Few guidelines are available for this. In addition, for public web sites, it is rather unpredictable who will be the users, and why and how they will use the web site. Therefore, it is difficult for the designer to completely assess correctly how the web site will be used and for what purposes. It is thus, at design time, difficult, if not impossible, for the designer to acknowledge what will be the optimal (overall) site structure and organization. In the same way, it is even difficult to correctly assess the exact information (and functional) requirements of the diversified users. Standard user requirement elicitation techniques (e.g. questionnaires, user interviews) are often more difficult to perform (compared to classical software applications) due to the variety of targeted visitors and their anonymity, a property intrinsic to the web.

Runtime information about the usage of the web site is useful to obtain a better insight in the requirements of the different users and the effectiveness of the web site (navigation) structure and organization. This information can be used not only to accommodate (i.e. personalize for) one particular user, but also to facilitate *all* users by optimizing the global structure and organization of the web site. The nature of the WWW, where all users access the application through a single server (compared to classical software applications, where each user accesses a personal (or local) copy of the application), lends it itself perfectly to gather, at run time, information about *all* users. E.g. based on this information, popular pages, or parts of pages, might be made more directly available (at design time the designer is usually unaware of which pages will be popular at runtime). Runtime information may as well be useful to better realize the business goals of the web site owners. E.g. advertisements can be put on the most frequently visited pages, or the navigation structure can be adapted based on the observed browsing behaviour to assure users come across certain strategic (commercial) information.

Although several tools exist for measuring, reporting and analyzing web traffic (e.g. AWStats[10], WebSideStory® HBX™ Analytics[11], WebTrends[12] and others), still a human interpretation of the

[10] http://awstats.sourceforge.net/ (access date: 20 June 2005)
[11] http://www.websidestory.com/ (access date: 20 June 2005)

resulting information is needed and a manual adaptation of the web site is required. It is clear that this is not the preferred solution. Even with tool support, a design iteration is a costly endeavour, especially when the actual web site is not fully automatically generated (i.e. the implementation needs to be altered as well). Moreover, in practice, people don't bother to adapt the design; they will often perform the adaptations directly at the implementation level. This leaves design and implementation out of synch, and possibly breaks with the design philosophy and principles used during design. With unbridled, ad hoc modifications, site management becomes problematic. Moreover, analysis and interpretation of the user access logs to detect possible problems is a far from trivial task[13] [Ivory and Hearst, 2001].

In conclusion, it is our conjecture that web designers could greatly benefit from anticipation of user access data in their design. This for several reasons: to anticipate runtime browsing behaviour (and act upon it), to detect and correct design flaws, to select between design alternatives and to better realize business goals. By specifying at the design level when and which adaptation will be possible (at runtime), unbridled ad hoc (manual) modifications (after deployment) are unnecessary, and the adaptation that is performed is better manageable and controllable. Furthermore, the adaptation can be done automatically (on the design), keeping web design and implementation consistent. Unfortunately, web site design support for the aforementioned adaptation is currently not available.

It is the aim of this dissertation to allow design time specification of (runtime) adaptive behaviour for web sites, yielding the benefits mentioned above. In this way, we effectively allow the design (and creation) of adaptive web sites: *web sites that automatically improve their organization and presentation based on user access data* [Perkowitz and Etzioni, 1997a]. Note that it is the main objective of this dissertation to improve the web site (design) for *all* users and *not* to personalize for one particular user. Although the presented approach lends itself, with some minor additions, perfectly to support personalization (see Chapter 8, Conclusions), it is not the subject of this dissertation.

1.3 Approach

The approach taken in this dissertation to tackle the formulated problem is to complement the web site design with a mechanism to describe, at design time, which adaptive behaviour is allowed at runtime to optimize the web site's organization and structure in order to improve the usability of the web site.

The mechanism proposed for this consists of a high level, rule-based specification language, called the Adaptation Specification Language, designed to act upon the relevant web site design models (i.e. the conceptual and the implementation design models). This language allows specifying what, how and when adaptive behaviour (at runtime) should be performed. This is done completely separately from the regular design itself. Thereby, possible adaptation issues are effectively separated from the regular design, pursuing a clear separation of design concerns. In addition, it is important to notice that the adaptations are specified in terms of design concepts. So, in fact, the design of the web site becomes adaptive. How the actual adaptations are performed on the implementation level is explained in Chapter 7 (Validation). In this way, the design and the implementations stay consistent.

With the Adaptation Specification Language, the designer has to his disposal the means to specify *when* certain adaptation should be applied (i.e. the *adaptation policy*) and *which* adaptation should be performed (i.e. the *adaptation strategy*). To specify an adaptation policy, the Adaptation Specification

[12] http://www.webtrends.com/ (access date: 20 June 2005)

[13] I.e. data mining is a research field that tackles the problem in identifying useful information from a huge amount of data.

Language includes a versatile event-specification mechanism and a way to specify when each adaptation strategy needs to be performed (i.e. based on which event). In order to specify an adaptation strategy, the designer is given the means to express exactly which runtime browsing information is required and exactly which adaptive actions (possibly) will be taken (based on the gathered runtime information). Neither information gathering nor the specification of changes is limited to pre-defined form; the designer is given the freedom to devise and implement his own adaptation strategies, but he can also re-use pre-defined strategies.

The design time support for adaptation as described in this dissertation can improve a web site design in various ways, and is applicable for the following purposes:

- Improve usability for *all* users: by utilizing web site usage information (of *all* users) during the web site design, the designer is able to anticipate common user browsing patterns, and (automatically) adjust the web site to better accommodate these users. For example, popular pages might be linked directly from the homepage.
- Validation of design-effectiveness: by using web site usage information of all users, the designer is able to validate its design by detecting if the actual use of the web site corresponds to his design intention.
- Automatic evaluation and selection between alternative design choices: at design time, it is impossible for the designer to know how exactly the web site will be used. It is thus also difficult to choose between different design alternatives. Using web site usage information, the designer is able to evaluate the effectiveness of his design choice, and if necessary, (adaptively) switch to the alternative choice.
- Better realization of business objectives: By observation of browsing behaviour, and specification of adaptive changes according to detected browsing patterns, the web site owner might increase his success in realizing his business objectives. For example, if it is detected that strategically important (business) information is on a page that is visited only few times, the designer might adaptively re-locate that information.
- Evaluation of the (runtime) use of the web site: the designer is able to specify, at design time, which and how design elements need to be tracked at runtime. Using the Adaptation Specification Language, he is furthermore able to communicate this information, at runtime, to the webmaster in a (pre-defined) comprehensible form. In this way, both calculated and webmaster-interpreted evaluation is possible.

The vision described above is realized in the framework of an existing web design method, WSDM (i.e. the **W**eb **S**ite **D**esign **M**ethod). In order to be able to elaborate the proposed vision the following steps were taken:

- **WSDM elaboration**: at the time of starting this work, a presentation design phase was still lacking for WSDM. Without a presentation design, it was not possible to define the layout of a webpage (i.e. positioning and look and feel of page objects) and thus it was unfeasible to (automatically) render an actual webpage/site from a WSDM design. In order to validate any work done on WSDM (i.e. adaptation in this dissertation) it was thus necessary to elaborate the WSDM presentation design. A layered set of modelling concepts for page presentation and position was provided. Primitive presentation concepts, sufficient to specify most common layouts, were complemented with high level presentation concepts. These high level presentation concepts build upon the primitive presentation concepts, but are more intuitive for the designer to use. Using these presentation concepts the designer can specify the layout of webpages; or he can define templates, which are used to define the common part of webpages. Finally, styles are specified using cascading stylesheets[14].

- **Formal foundations:** as informal descriptions unavoidably introduce ambiguity, the work described in this dissertation is based on a formal foundation. Therefore, first WSDM, the web site design method used as a framework for this dissertation, has been formalized. Subsequently, on top of the formal specification of WSDM the necessary models to support adaptation are defined. Finally, basic design transformations are formally described based on the aforementioned formal models.
- **Specification mechanism for design time adaptive behaviour:** based on the formal models, which unambiguously describe both a WSDM design, the adaptation models and the basic design model transformations, the higher level, rule-based Adaptation Specification Language is defined. The Adaptation Specification Language allows to build (or re-use) complex adaptation strategies using flexible runtime information storage and basic design model transformation.
- **Illustration of applicability:** having defined the formal foundations and based upon this formalization the Adaptation Specification Language to support designer specified adaptive behaviour, examples of possible adaptation strategies are given to illustrate the use and strength of the approach. Each strategy shows different applicability possibilities.
- **Validation:** to validate the work described in this dissertation, a prototype implementation of the proposed approach is made, and an experiment has been performed. With the experiment some of the adaptation strategies were tested and evaluated.

1.4 Advantages

The approach outlined provides several advantages compared to the current situation:

- Supporting design time adaptive behaviour in the context of a web site design method allows to specify well-thought through changes, which can be kept consistent with the chosen design philosophy and the intentions of the designer (compared to manual changes, which are more time-consuming, less structured and decrease site manageability)
- The flexibility offered by the Adaptation Specification Language goes far beyond existing (personalization) approaches, which are mostly limited to simple (pre-defined) access counts.
- The specification of possible runtime adaptive behaviour is completely uncoupled from the (regular) design and implementation of the web site. This provides a clean separation of concerns, which is generally accepted to improve quality of design as the designer can consider each design aspect separately.
- As the adaptation is in no way intertwined with the existing design models, the adaptation concerns can also be kept separated in the implementation. By doing so, a constructed web site still functions trouble-free even if adaptation is for some reason not available (e.g. some web server restrictions could prohibit server side execution of adaptation)
- As a further consequence of the clean separation of the regular web site design and the specification of the web site adaptation, the adaptation policies and strategies need not to be specified during the design phase. It is perfectly possible to add (design) adaptation (by an ASL expert) after the web site has already been deployed.

[14] http://www.w3.org/Style/CSS/ (access date: 20 June 2005)

- The Adaptation Specification Language facilitates re-use of existing adaptation strategies. This allows novice designers to benefit from useful adaptation strategies, even if they do not have the skill to devise them themselves.

1.5 Contributions

The contributions of this dissertation can be divided into two groups: major and secondary contributions. Major contributions comprise the research results presented in this dissertation; the secondary contributions concern the research that was needed to refine and elaborate WSDM (see Chapter 3). These research results are not always explicitly mentioned in this dissertation but can be found in publications. We mention them briefly:

- The audience modelling phase was significantly improved by introducing a formal method to derive Audience Classes and the Audience Class Hierarchy. Details can be found in [Casteleyn and De Troyer, 2001].
- The WSDM conceptual design phase was better specified, by introducing the task modelling sub phase. This made it easier to obtain the elementary requirements needed for object chunk modelling, and to specify the internal navigation structure with a navigation track. This work was performed in cooperation with my promoter Prof. Dr. Olga De Troyer. Details can be found in [De Troyer and Casteleyn, 2003b].
- Link types were introduced to distinguish between structure, semantic relationship, navigation aid and process logic links. Link types can be exploited in various ways (e.g. enhancing presentation by materializing links with a different purpose in a different way, and allow a deeper separation of concerns, by allowing the designer to focus on one type of link at a time. This research was also performed in cooperation with my promoter Prof. Dr. Olga De Troyer. Details can be found in [Casteleyn and De Troyer, 2002 ; De Troyer and Casteleyn, 2003a].
- WSDM was extended to support accessibility for visually impaired users. This was done by combining WSDM with the Dante approach, which allows semantic annotation of web pages to provide screen readers with extra (semantic) knowledge to better facilitate the audio presentation of web pages. A mapping between the WSDM design concepts and the Wafa Ontology[15] concepts was described, and subsequently used to extend the WSDM implementation generation process to include semantic annotations to support accessibility for visually impaired users. This research was performed in cooperation with Peter Plessers, my promoter Prof. Dr. Olga De Troyer and the School of Computer Science research group of Manchester University (UK). Details can be found in [Plessers et al, 2005].

1.5.1 Major Contributions

The major scientific contributions presented in this dissertation are the following:

- A mechanism that allows designers anticipating certain runtime behaviour at design time to incorporate possible design adjustments/corrections to improve the quality of the web site.
- Some adaptation strategies, which illustrate how a web site design can benefit from designer specified adaptive behaviour, are described. Adaptation is shown to complement, enhance (i.e. by taking into account web site usage information, e.g. provide navigation shortcuts to frequently visited pages) and validate (i.e. web site usage information may reveal design flaws) traditional web site design.

[15] The Wafa ontology is part of the Dante approach.

- A formalization of WSDM is provided. The formal specification defines the design models of WSDM more rigorously and eliminates ambiguity. It has been used as the basis for the WSDM Adaptation Model, but will also be useful for further work on WSDM.
- A formal definition of one of the distinguishing features of WSDM, the audience driven navigation structure, is given. This had not yet been done so far.
- Basic model transformation operations are defined upon relevant WSDM design models. These operations allow transforming the design models. These operations may also be useful in other contexts (see Chapter 8, Conclusions, for an elaboration of these operations in the context of web site evolution).
- A formal model and mechanism to gather, at runtime, information about the usage of the web site in terms of conceptual modelling concepts is defined (the WSDM Adaptation Model).
- A high level, rule-based adaptation specification language (ASL) based on the formal specification of WSDM and the WSDM Adaptation Model, is defined. Using ASL, the adaptive behaviour that is allowed at runtime can be described at design time in terms of design concepts, by specifying arbitrary adaptation policies and strategies. In addition, ASL allows defining more high-level model transformations (e.g. promotion of a node) on top of the basic model transformation operations.
- The Implementation Design of WSDM was developed to an extent that allows today's most common style and layouts to be modelled
- A prototype implementation of the Implementation Generation phase was developed. It was needed for the prototype implementation of our approach but it is also usable for a regular WSDM design (without adaptation specifications). In addition, it proves that the information collected during the different design phases of WDM is sufficient to generate the web site automatically.

1.6 Outline

The rest of the dissertation is structured as follows.

Chapter 2 describes background work and related work. A general introduction to hypertext and hypermedia is given, and the most widely used hypermedia reference model, DEXTER, is reviewed. Due to its particular importance, the limitations are difference between web systems and hypermedia systems are discussed. The different sections in this chapter can be read independently; readers familiar with some or all of the topics can skip parts or even the entire chapter.

Chapter 3 describes in an informal way, the Web Site Design Method (WSDM), which is used as a framework in this dissertation. A general overview of the method is given, and subsequently, each design phase (i.e. Mission Statement, Audience Modelling, Conceptual Design, Implementation Design and Implementation Generation) is described in detail. For each phase, the different models and the techniques to derive these models are described. Examples illustrate each model. Non technical readers are recommended to read this chapter.

Chapter 4 gives the formal specification of WSDM. For each design phase of WSDM, formal definitions for the different models are given. Although this chapter can be skipped without compromising a global understanding of the rest of the dissertation, some knowledge of this chapter is needed for the Adaptation Model, presented in the next chapter.

Chapter 5 describes the WSDM Adaptation Model, build on top of the WSDM formal specification of the previous chapter, in a formal way. First, the Web site Overlay Model is introduced, which formally describes how (runtime) information needed to specify adaptive behaviour is captured. Subsequently, basic information storage operations are defined to manipulate the Web site Overlay Model, and basic model transformation operations are supplied to manipulate WSDM design models. Finally, the Adaptation Specification Language (ASL), a high level rule based specification language, is defined and explained. A general overview of the language is given, and subsequently the different ASL language constructs are discussed in detail. Reading this chapter, more specifically the Adaptation Specification Language section, is essential for the understanding of this dissertation.

Chapter 6 presents three example Adaptation Strategies: specifications, based on web site usage information, of *how* the structure and organization of a web site needs to be (adaptively) changed. The discussed strategies are promotion/demotion, re-ordering sequential information (gathering) and audience driven navigation validation. For each Adaptation Strategy, the general principle is explained, an illustrating example is given, and the specification using the Adaptation Specification Language is provided.

Chapter 7 describes the validation performed for the work described in this dissertation. The implementation of the WSDM design models, using semantic web technology, is discussed. Subsequently, the implementation of the code generation from a WSDM design is presented. Next, the implementation of the Adaptation Specification Language and an implementation architecture to support adaptation support for WSDM designed web sites as explained in this dissertation is discussed. Finally, an experiment is presented, in which two of the Adaptation Strategies described in the previous chapter, are evaluated. Results of the experiment are discussed.

Finally, *chapter 8* reflects on the results of this dissertation. A summary is provided, limitations and boundaries are states and contributions and achievements are outlined. Finally, contemplating the research done, possible extensions and future work are discussed.

Chapter 2: Background and Related Work

"If I have seen further [than others], it is by standing on the shoulders of giants"
(Isaac Newton)

The previous chapter introduced this dissertation. It explained the context, addressed problem(s), and outlined the proposed solution and its benefits. In this chapter, the necessary background knowledge to be able to place the work of this dissertation in a broader perspective and related work are presented. First, a short overview of hypermedia is presented; definitions are stated and reference models reviewed. Subsequently, an introduction to the adaptive hypermedia field is given, again stating definitions and reference models. Finally, we focus on the World Wide Web and more particular on web design methods. A few representative web design methods are reviewed in detail, and support for adaptation (in any form) is highlighted. Finally, we situate our work within the field of web engineering and adaptive hypermedia.

The remainder of this chapter is structured as follows. First, section 2.1 gives an introduction to Hypermedia. Some historical background is provided, definitions are stated and best known reference models are discussed. Section 2.2 describes Adaptive Hypermedia. Again, definitions and well known reference models are stated, and complemented with some extra explanation on the adaptive aspect of adaptive hypermedia. Section 2.3 goes in deeper detail into Web Design Methods. Five representative web design methods are discussed, and their capabilities for adaptivity are pointed out. Finally, section 2.4 situates the work described in this dissertation with in the research field.

2.1 Hypermedia

Hypermedia systems, of which the World Wide Web is now undoubtedly the best known and most widespread example, fundamentally changed the way we publish and access information. An overview of the history and definitions of hypermedia is given in this section.

2.1.1 History

It is generally acknowledged that the roots of hypermedia systems can be traced back to immediately after World War II, when Vannevar Bush, director of the Office of Scientific Research and Development of the United States, published his visionary paper "As We May Think" [Bush, 1945].

In that paper, Bush foresees that, with the dazzling speed at which advances in different fields of science are made, traditional ways of transmitting and reviewing results of existing research will become inadequate as the amount of research results takes vast dimensions. The biggest challenge ahead, according to Bush, will be to maintain and most importantly to *efficiently consult* the vast and ever growing amount of research available. He realized the shortcomings of linear indexing systems, and observes that the human mind operates by *association*.

Based on these observations and confident in future technological advances to realize his ideas, Bush describes a device called ***memex***[16]. A memex is a device in which an individual can store all his/her books, records, communication, etc. and which is mechanized in such a way it can be consulted rapidly and efficiently. Next to the usual (alphabetical) indexing scheme, a memex user can add notes and comments to any piece of text stored in the memex, and, most essential, the user can add at any arbitrary location *a pointer* to another piece of text. When the item with the pointer is accessed, the *linked* item can be retrieved immediately. In this way, users can create *trails* through the text, effectively allowing non-linear access to the information stored in the memex.

[16] Interestingly, Bush describes his memex in terms of projected future photography- and mechanic developments. Computer development was only in the very early stages in 1945. The ENIAC, generally conceived as the first electronic computer, was completed in 1946 and was only capable of basic arithmetic computations. It was nowhere near the versatile, powerful computer which would later trigger the widespread use of hypermedia systems.

In 1965, Nelson first used[17] the term *hypertext* and described a system called the ***PRIDE*** (Personalized Retrieval, Indexing and Documentation Evolutionary) System in [Nelson, 1965], an article clearly inspired by the ideas of Bush. PRIDE 's purpose is to support an ELF (Evolutionary List Files) based file structure, which would allow the idea of *hypertext*. The core of ELF consists of *zippered lists*. An ELF file thus consists of *entries, lists and links*. An entry is an elementary unit of information of any granularity, a list is an ordered set of entries designated by the user, and a link is a connector made by the user between two entries in two different lists. Lists are *zippered* when there exist link(s) between their respective elements. In the same period, Nelson had been working at ***Xanadu***, a system that would support versioning of documents and non-linear associations between (pieces of) text. The idea was later extended to include copyright management, but it was never totally implemented. However, Xanadu would influence future developments in hypermedia systems.

In 1968, Doug Engelbart publicly presented many revolutionary advances of interactive computing (e.g. the *mouse* was first shown), while giving a demo of ***NLS*** (oN Line System) at the Fall Joint Computer Conference in San Francisco (movie files of the demo are available at http://sloan.stanford.edu/mousesite/). NLS was part of the Augment project, started in 1962 and aimed at developing computer tools to augment the human capabilities and productivity. NLS supports stored texts, with cross referencing and figures with 'hot spots' (clickable area's) which could be used for various purposes. It was the first demonstration of a working hypertext/hypermedia system.

It was around the same time that Andries van Dam first developed ***HES*** (Hypertext Editing System, 1967) and later introduced it's successor ***FRESS***[18] (File Retrieval and Editing System, 1968) at Brown University, the first hypertext system to be fairly widely used (at his university) for an extended period of time. FRESS implemented (multiple) windows and vector graphics, along with (bi)directional links, hierarchical structuring, typed links and an undo feature. [van Dam, 1988]. Both HES and FRESS ran on an IBM mainframe, and HES, once completed, was sold to the Houston Manned Spacecraft Center, and was used to generate documentation for the Apollo missions.

Started in 1972 and fully functional in 1977, the ***ZOG***[19] hypermedia system, envisioned and developed by Donald McCracken and Robert Akscyn, consisted of (text-only) frames representing parts of a database, connected through (bi-directional) links. Each frame consisted of a title, a description, selections (which realized the links to other frames) and a line of ZOG commands. ZOG later evolved to ***KMS*** (Knowledge Management System), which became a commercial product in 1983. KMS was optimized to show the next frame to which the user navigates within 0.5 seconds[20]. Other distinctive features of KLM include the 'home frame', accessible from anywhere in the hypermedia system, and the ability to track back to previous nodes.

In 1978, the first true hypermedia system[21] was developed by Andrew Lippman at MIT: the ***Aspen Movie Map***. It offered the user a virtual walk through Aspen (USA), using prerecorded systematically taken images in 4 directions (north, east, south, west) that were linked together accordingly. The images were stored using videodiscs, and the user was confronted with two separate screens, one

[17] According to *Interesting Times,* the Ted Nelson News letter (nr 3 October 1994), it first appeared in the Vassar College *Miscellany News* article "Professor Nelson Talk Analyzes P.R.I.D.E." (February 3, 1965).
[18] After meeting with Engelbart and having seen NLS.
[19] The word ZOG is not an acronym, and has no particular meaning.
[20] It was possible to increase this response time, but tests showed users had difficulty noticing whether the screen had changed or not (at a frame display of 0.05 seconds).
[21] Hyper*media*, as opposed to hyper*text*.

presenting the current position, the other presenting a general overview. The user was able to go in any direction, or pointing to a location on the overview map and immediately go there. As some buildings were photographed from the inside as well, the user could enter some of them.

After having been pioneered in the late sixties and seventies, research in hypermedia increased significantly in the eighties, introducing new systems and features, a lot of them becoming commercially available.

The ***Symbolics Document Examiner*** project started in 1982 by Janet Walker [Walker, 1987] and was from the start aimed to serve as a documentation platform for Symbolics workstations). From 1985 it became commercially available, and was used extensively by Symbolics users. The main idea was to have one node for each piece of information considered relevant for the user. For structuring the information space, a simple book metaphor was used. The user could access the table of content, and jump to chapters and sections. Furthermore, the bookmark principle was introduced.

Hyperties was a research project started by Ben Shneiderman [Shneiderman, 1987] at the university of Maryland in 1983. Hyperties deploys plain text screens, much like KMS, but provided links to documents (possible consisting of multiple pages) instead of to one single frame. Instead of being transported to the target of a link, Hyperties shows a short definition on the bottom of the screen, with some explanation on what to expect when traversing the link. The design of Hyperties was kept simple, resulting in a somewhat restricted hypermedia (e.g. same text from will always be used to point to the same resource, etc). However, as the original targeted users were non technical people (e.g. librarians, museum curators, historians, etc.) working on simple, text-based computers, this sufficed. Hyperties was commercially available from 1987

Notecard (1985), initiated by Frank Halasz [Halasz, 1987], is probably the best known of historical hypermedia systems. It was developed on the Xerox D-machines, specialized Lisp machines. The InterLisp programming environment on these Xerox machines allowed easier implementation of Notecard, and the strong integration of the list interpreter in Notecard allows the user to programmatically customize any aspects of the hypermedia system. The main design concept in Notecards is a *nodecard*, which can be connected to each other by *links.* The user can have one or more notecard open on the screen at any time, each of which is a scrollable window (as opposed to fixed frames). The simplest nodecard consist of plain text or graphics, but there exists various specialized notecards exist for different purposes. The user is able to define his/her own specialized cards, making it an extremely flexible hypermedia system. Two important special cards in Notecard are the browser card, containing a structural overview of nodecards and links, and the filebox, used to hierarchically nest nodecards. Notecard distinguishes different link types, which can be chosen ad hoc by the designer. Later, the Instructional Design Environment (IDE), a courseware application, was built on top of Notecards, providing a new interface fitted for courseware developers.

The ***Intermedia*** hypertext system was, as HES and FRESS, developed at Brown University for educational use starting from 1985 [Yankelovich et al, 1988]. Links in Intermedia were bidirectional, and connected anchors rather than nodes. This made it possible to jump immediately to a specific label at a specific location in a document. Overview diagrams could be either automatically generated, or be user defined. Furthermore, users can define their own set of links, annotating the existing hypertext. The Intermedia project was terminated in the early nineties.

The ***Guide*** project was initiated at University of Kent in 1982 by Peter Brown [Brown, 1987]. It was commercialized, and became widely popular when it was released both for the Macintosh platform (in 1986) and soon after for the Personal Computer. Guide is, as Intermedia and Notecard, based on scrolling text windows. Links are represented as strings and possibly point to a location within the same file. Guide distinguishes between three kinds of links: replacements (expansion of text at the current location, e.g. stretchtext), pop-ups (similar to pop-ups commonly known now in the WWW, yet only being displayed while pressing the link) and jumps (links causing to jump to another location in the hypertext).

In the late eighties (1987), Apple made ***Hypercard,*** designed by Bill Atkinson, widely available by distributing it for free with Macintosh computers. It included an easy to learn programming language, HyperTalk, allowing fast prototyping of graphical user interfaces. The basic concepts of Hypercard are *card*, similar to frames in the earlier systems, and *stacks* (collections of cards). Cards can be composed of different layers, allowing the designer to re-use design elements (e.g. backgrounds) for different cards. The main navigation feature are buttons, to which the designer can attach a programming statement. In the simplest case, this is a straightforward goto-statement, implementing a jump to another location. However, the destination could also be computed by a more complex program computing the target. More-over, other actions (e.g. hiding/showing text) could be attached to a button. Not only the actual pressing of buttons can activate an action; some other events can cause activation as well (e.g. cursor going over a region, inactivity of the user, etc). This flexibility, its wide availability and the existence of an extensive library of graphical building blocks for user interfaces, made Hypercard extremely popular during the beginning of the nineties.

In the same year as Hypercard introduced hypertext systems to the general public, the first ACM Hypertext conference, Hypertext'87, was organized in North Carolina. By then, interest in hypermedia systems had grown and research effort in the field was increased. Many other hypertext and hypermedia systems were introduced from that time, too many to discuss here. However, the most important one, the World Wide Web, would have an enormous impact, changing the way people work and live. Due to its particular importance for this dissertation, the WWW is discussed in a separate section (see section 2.1.4).[22]

2.1.2 Definitions

As discussed in the previous section, hypermedia systems were first envisioned by Vannevar Bush in 1945. From 1965, functioning hypermedia systems emerged and became more widespread ever since. With the arrival of the World Wide Web, hypermedia systems became known to the general public. But, what exactly is a hypermedia system?

In [Nielsen, 1995], Nielsen explains the notion of hypermedia by making the comparison with classical media. In classical media (e.g. a book, a video, …), the order of presentation of content is sequential. In hypermedia systems, the order of presentation is non-linear: the order of access to the different information pieces of the system is non-sequential and determined by the user. To accommodate this freedom, the content is somehow structured in information units, which can be referred to from anywhere in the hypermedium. Each unit of information is called a *node*, pointers among nodes are called *links*. The granularity of the nodes nor the existence of links is fixed in advance. In fact, hypermedia systems often allow introduction of new nodes and links in the system by the user. Where in a hypertext system, the subject of the system is text-based, hypermedia support multimedia contents (i.e. text, audio, video, …).

[22] The author acknowledges Nielsen, who's book [Nielsen, 1995] provided a lot of useful information contained in this section.

A definition of hypertext, in terms of nodes and links, is given by Bieber in [Bieber, 2000]: "*Hypertext is both the concept of interrelating information elements (linking pieces of information) and the name used to describe a collection or web of interrelated or linked nodes.*"

Shneiderman [Shneiderman and Kearsley, 1989] gives a more technology oriented definition of hypertext, which also puts emphasis on the (user-) act of *browsing* (i.e. *following* a link): "*a database that has active cross-references and allows the reader to "jump" to other parts of the database as desired*".

McDaid [McDaid, 1991] differentiates hypermedia and hypertext in his definition, and stresses the ability to re-arrange information by the user: "*Hypermedia is Theodore Nelson's term for computer-mediated storage and retrieval of information in a non-sequential fashion. An extension of Nelson's earlier coinage, "hypertext" (for non-sequential writing), hypermedia implies linking and navigation through material stored in many media: text, graphics, sound, music, video, etc. But the ability to move through textual information and images is only half the system: a true hypermedia environment also includes tools that enable readers to rearrange the material.*"

To deepen the understanding of hypermedia and the terminology used, the next section discusses the most important hypermedia reference models.

2.1.3 Hypermedia Reference Models

2.1.3.1 The DEXTER Reference Model

Undoubtedly the most frequently used and widely referenced hypertext reference model is the DEXTER reference model [Halasz and Schwartz, 1990 ; Halasz and Schwartz, 1994]. Because of its importance in the field of hypermedia in general, and adaptive hypermedia in particular (as AHAM, see [De Bra et al, 1999b], is based on DEXTER), we will discuss the model in detail. The DEXTER model was published in 1990, and the result of extensive discussions among representatives of then existing hypermedia systems. It was an attempt to identify the common features and the difference among these systems, to arrive to a standard hypertext terminology and formal model of the important abstractions in hypertext systems. As all existing hypertext systems needed to be expressible in Dexter, the resulting model allows more than any single hypermedia system[23]. It does, however, provide a good basis for comparing hypermedia systems and a shared understanding of terminology.

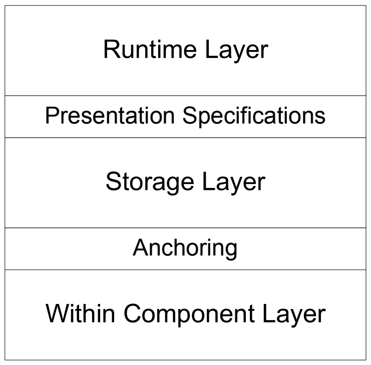


Figure 2-1 Overview of the Dexter Reference Model

[23] This is correct even to date.

Figure 2.1 shows a general overview of the Dexter model. It consists of three main layers, the *Runtime, Storage and Within Component Layer*, and 2 interface layers, *Anchoring and Presentation Specifications*. The Storage and Run Time Layer will be described in the following subsections; the 2 interface layers are elaborated on where appropriate. The Within Component Layer is purposely not described in the Dexter reference model, to allow any kind of information with appropriate referencing mechanism to be included. For a formal specification of Dexter, see [Halasz and Schwartz, 1990].

Storage Layer

The storage layer describes the structure of a hypertext as a finite set of components, each with a resolver and an accessor function.

The main entity is *the component*, which is (globally) uniquely identifiable and accessible through the accessor function. A component is either an *atom*, a *link*, or a composite entity. Atoms are primitives in the storage layer, their substructure is the concern of the Within Component Layer. Links are relations between other components, specified by means of 2 'endpoints specifications' each referring to (part of) a component. Composite components form a hierarchy, which is restricted to a directed acyclic graph.

In order to facilitate addressing of components that are not statically present in the system (i.e. the result of a query), Dexter uses *resorver functions* and *component specifications*. The resolver function resolves a component specification into a unique identifier (UID). In its simplest form, the component specification is the UID itself.

To preserve the separation of concerns between the Storage Layer (components) and the Within Component Layer, *anchors* are used to created links between *parts of* components. An anchor consists of an *anchor id* and an *anchor value*. The *anchor id* is unique within the scope of the component. A component UID and an anchor id thus globally uniquely identify a part of a component. The *anchor value* is a primitive which specifies a location, and is subject of interpretation of the application handling the components.

Endpoint of links can now be described using *specifiers*. A *specifier* consists of a *component specification*, an *anchor id*, a *direction* field (FROM, TO, BIDIRECT, NONE) and a *presentation specification* (a primitive value, part of the interface between the storage layer and the runtime layer). Links are now described as a sequence of two or more specifiers.

The model facilitates arbitrary attribute/value pairs to be assigned to each component, which allows for example adding keywords or types to components. Furthermore, for each component the anchors that index it and a presentation specification are stored. The actual component is referred to as the *base component.*

Finally, basic accessor and constructor operations are described for adding/deleting components, retrieving a component given UID or specifier and retrieving set set of links that refer to an anchor given the anchor and its containing component.

Runtime Layer

The runtime layer is concerned with the presentation of the components to the user.

The main concept in the Runtime Layer is the *instantiation* of a component, which is uniquely identifiable by a IID within a session (one components can have more instantiations). An instantiation of a component automatically results in the instantiation of its anchors. The instantiated anchors are called *link markers*. As for components in the Storage Layer, instantiations are actually divided into *base instantiations* together with link markers and mapping functions to anchors for these markers.

A *session* keeps track of the moment-to-moment mapping between components and their instantiations (as the model allows opening multiple instances of a component, possibly editing and changing them). A session thus contains the hypertext being accessed, a mapping from UID's to their corresponding components, a history, a runtime resolver function, an instantiator function and a realizer function.

The *instantiator* function maps a (component) UID and a presentation specification into an instantiation of the component. Note that the component itself already contained a presentation specification describing the basic notions of how it should be displayed. It is the responsibility of the instantiator function to combine information contained in both presentation specifications and return an instantiation of the component.

The Runtime Layer also provides some operations. The *present component* operation takes as input a component specifier together with a session and presentation specification. Subsequently, it calls the *instantiator* function to output an actual instantiation. The *follow link* operation takes as input a IID of an instantiation together with a link marker contained in that instantiation. Subsequently, the component(s) "pointing to" the particular link marker are presented.

The instantiator also has an inverse function called the *realizer*, which returns a new, updated component (as components can be edited/changed).

2.1.3.2 Other reference models

A number of other reference models exist, but did not have the impact the Dexter reference model had. We state only shortly the most important ones.

Trellis [Furuta and Stotts, 1989 ; Stotts and Furuta, 1989] models a hypertext on the basis of Petri Nets, a generalization of directed (labeled) graphs. Advantages of this approach include the possibility of describing concurrent browsing, synchronization of (concurrent) browsing sessions, guided browsing (e.g. 'guided tours', see 2.2.4) and restrictive browsing. Furthermore, it supports some constraint checks, such as reachability. Although Petri Nets are particularly useful for describing concurrent browsing and its properties, their increased complexity compared to directed graphs make trellis a less used reference model.

The Amsterdam Hypermedia Model [Hardman et al, 1994] is a hypermedia reference model that extends the Dexter Reference Model with the notions of time, high-level presentation and (link) context. In addition to the functionality described by Dexter, the timing constructs allow detailed synchronization of (behaviour among) document components. The high-level presentation allows easier and more fine-grained creation and modification of presentations, and the link context allows detailed specification of the source context (the part of the hypermedia affected by initiating a link) and destination context (i.e. the part of hypertext that is displayed on arrival of a link).

The Tower Model [De Bra et al, 1992] is an object oriented data model for hyperdocuments build around the concept 'tower'. A tower is object is an object that has associated a set of labels, with for each label a mapping to a set of objects. The labels correspond to a different level of abstraction of the hyperdocument (e.g. presentation, linking, …) and depend on the kind of object (i.e. nodes, links, anchors, grouping objects, ..) the tower represents. The kind and amount of labels of a tower is arbitrary, i.e. there is no fixed set of abstraction levels as in the Dexter reference model (although labels corresponding to the levels of abstraction of Dexter will often be used).

2.1.4 The World Wide Web

Although the World Wide Web is undoubtedly the world's best-known application of a hypermedia system, it offers only restricted functionality of what a full hypermedia system could offer (see [Cailiau and Ashman, 1999]). The term "hypermedia" is commonly used to denote a collection of information elements interconnected by links, which can be browsed by the user to access (in some way) relevant information concerning the current link. The World Wide Web offers the following features which make it worthy of the name "Hypermedia":

- **Interconnectivity:** interconnectivity between sources via an embedded link in the source document to the target document
- **Heterogeneous links:** interconnectivity not only between text fragments, but also between other elements (e.g. image linking to text fragment); this is a feature of a hypermedia system, in contrast to a hypertext system, which only provides connectivity between text fragments.
- **Non linearity:** there is no a priori set order of traversal, the choice of navigation path lies completely with the user

Shortcoming of the World Wide Web regarded as a hypermedia system include:

- **Limited content-navigation separation:** in classical web sites the data and the navigation through that data are entangled and represented in one source. Even when underlying databases or other data sources are used, the client receives database content, document data and navigation all merged into one (html) document. Links are not considered first class objects; they are embedded in the data source.
- **No bi-directional links:** links are currently one-directional: they are embedded in the source, and make a connection from the source to a target. The target is not aware of the existence of the link, the designer of the target link is not aware of the consequences when he manages and changes his web pages. Apart from the uni- or bi-directionality, there is also no possibility to define more complex link structures, such as links with multiple endpoints.
- **No link typing:** links can only link (a fragment of) the source to another source (URL). There is no possibility to qualify the link with any additional information (e.g. semantic information, type information).
- **No link constraints:** as a result of the lack of separate link management, and the lack of typing possibility, it is also difficult to specify constraints for certain links. Constraints could add to the ability for web site designers to customize the link being presented to the visitors
- **No customizability**: although users do have control over the order of traversal, they do not have a lot of control over the hyperlink structures themselves: they cannot add or modify existing links, nor manipulate their presentation or its effect on frames or windows.

Web design methods, in contrast to hypermedia reference models, take into account the particularities of the World Wide Web. Such methods are further discussed in section 2.3.

2.2 Adaptive Hypermedia

Hypermedia revolutionized the way information is accessed by humans. The idea of presenting the information to the users in a non-sequential way, allowing the user himself to decide the next piece of information to consult, offered great user-freedom in accessing the global information space. Where (traditional) hypermedia systems abide by a somewhat static structure and linking of information[24],

[24] Indeed, in general, hypermedia systems allow user-introduced information and linking, yet the information, yet the structuring and linking of information is static in the sense that it is presented to all users in the same way.

adaptive hypermedia envision to adapt the information space (and the links between the pieces of information) in order to better accommodate the user while consulting the hypermedia. This section reviews adaptive hypermedia, by elaborating their definition and reference model in detail.

2.2.1 Definitions

Adaptive hypermedia and hypertext stem from research on user-adaptive systems. The goal of this research is to adapt the hypermedia to reflect the goals, knowledge and preferences of the user. In this way, adaptive hypermedia aim to better assist the user in his browsing session. How this is done exactly (i.e. what is adapted and how?) will be the subject of the next sub sections. First, we state what is understood, in literature, with adaptive hypermedia.

Brusilovsky [Brusilovsky, 1996] defines adaptive hypermedia as follows: "*by adaptive hypermedia systems we mean all hypertext and hypermedia systems which reflect some features of the user in the user model and apply this model to adapt various visible aspects of the system to the user.*"

This definition highlights three criteria a system must satisfy to be an adaptive hypermedia: 1) it should be a hypertext or hypermedia system, 2) it should have a user model and 3) it should be able to adapt the hypermedia using this user model.

In the AI community, and more specifically in the context of the World Wide Web, Perkowitz and Etzioni formulate a challenge to the AI community to build *adaptive web sites* [Perkowitz and Etzioni, 1997a]: "*web sites that automatically improve their organization and presentation based on user access data*". This definition does not specifically mention the existence of a user model. In fact, it does not, as does Brusilovsky's definition, fixes adaptive hypermedia to systems that adapt to the current user. Instead, Perkowitz and Etzioni distinguish between customization (i.e. adjust the web site for the current user; personalization) and optimization (i.e. improve the web site as a whole, for *all* users).

2.2.2 Adaptive, Adaptable, Dynamic

In popular literature and even in the research domain, some confusion about the terms *adaptation, adaptivity, adaptability, dynamic* exists. Before continuing, we shortly review those three terms. Paul De Bra [De Bra, 1999a] describes the difference as follows (also see [Mérida et al, 2002]):

In *Adaptable Hypermedia* it is the user himself that can adapt certain parts of the hypermedium, by providing some information to the system (in the form of a profile) according to which the 'customized' version of the hypermedium is presented. Typically, presentation preferences (colours, fonts, etc) and user background (qualifications, knowledge, etc) information are provided.

In *Adaptive Hypermedia*, the system monitors and logs the user behaviour, and adapts the presentation of the information nodes accordingly. The user's preferences and knowledge about concepts is updated continuously (if needed), and is mostly derived from accesses to pages. Knowledge for a particular user can also be derived (generalized) from other users.

Also concerning the difference between adaptability and adaptivity, Dieterich [Dieterich et al, 1994] decomposes adaptation in four activities: identification of the need for adaptation, proposal of adaptation mechanisms, decision of which adaptation to conduct, and execution of the decided proposal. Depending whether these steps are performed by the user or the system (or by both), Dieterich respectively speaks of adaptability and adaptivity.

Finally, in *Dynamic Hypermedia* the system also monitors the user's behaviour, but instead of adapting predefined presentation, dynamic systems generate a presentation from atomic information items (often through natural language generation).

2.2.3 Adapt to what?

Adaptive hypermedia systems adapt the appearance of the system to the user according to his user model. So, the important question is: what exactly is contained in this user model, or in other words, according to which information will the system adapt? In [Brusilovsky, 1996], Brusilovsky identifies four features to which the system may adapt:

- **Knowledge**: knowledge about the subject represented in the hypermedium is a common feature based on which to adapt. One of the most popular applications of adaptive hypermedia, educational systems, mainly relies on this kind of information. In these systems, the knowledge of the user about the concepts of the subject domain is most frequently represented by an overlay model. In an overlay model, an estimation of the knowledge of the user for each domain concept is stored. The information contained in the overlay model is updated continuously while the user is using the application.
- **Goals**: the user goals are more related to the context of the user's work, rather then with the user as an individual. Examples of user goals include search goals (in information retrieval systems) and learning goals (in e-learning systems). User goals are highly volatile: the same user might have different goals on different times he accesses the application, and his goals may evolve (even within one browsing session).
- **Background and Experience**: by user experience, the familiarity of the user with the system is denoted; user background refers to related knowledge the user has (before using the system) which can be relevant in some way.
- **Preferences**: when using a hypermedium, the user may for some reason have some preferred nodes or links. These can be absolute (i.e. they are invariant) or relative (they depend on the current node, goal or current context). Unlike the previously mentioned features, preferences cannot be deduced by the system (i.e. preferences need to be explicitly acquired from the user).

Brusilovsky focuses, as mostly done in the adaptive hypermedia community, on user characteristics, somehow stored in a user model, to base adaptation on. However, in the context of personalization for hypermedia systems (and more specifically for the WWW) to improve online customer relationships, Kobsa et al [Kobsa et al, 2001] identify three different categories of data on which adaptation (i.e. personalization in this case) may be based[25]:

- **User data:** user data is information about the personal characteristics of the user. This category conforms quite well to the features identified by Brusilovsky. Examples include demographical data, user knowledge, user skills, capabilities, interests, preferences, goals, plans, …
- **Usage data:** usage data is information about the interaction of the user with the system. Usage data may be directly observed, it may be collected by the system or it may be deduced by analysing the applications access logs. The granularity of usage data depends on the hypermedia system used (e.g. html-based systems can only detect page visits and link activations; they cannot, as some more dedicated systems can, record scrolling, record in-page navigation, …)

[25] This is acknowledged by Brusilovsky in [Brusilovsky, 2001]

- **Environment data:** environment data relates to the environment of the user. This includes his software (e.g. browser versions, available plug-ins, …), hardware (e.g. bandwidth, display device, …) and locale (e.g. users current location, characteristics of the current location)

2.2.4 What to adapt?

In his elaboration on what to adapt, Brusilovsky [Brusilovsky, 1996; Brusilovsky, 2001] distinguishes between *content-level adaptation* and *link-level adaptation*. He calls these two classes of adaptation respectively *adaptive presentation* and *adaptive navigation support*. For both classes, Brusilovsky [Brusilovsky, 1996; Brusilovsky, 2001] and Debra et al [De Bra et al, 1999a] describes the following techniques:

Adaptive Presentation:

- **Adaptive text presentation:** the textual presentation is adapted. Several variants of adaptive text presentation exist: page variants (different variants of each page exist), fragment variants (different variants of fragments of pages exist), frame based techniques (pages are assembled from small information items), inserting/removing fragments, stretchtext (allows the user to activate in-place text expansion), altering fragments, sorting fragments and dimming fragments (text is dimmed to indicate a lesser importance).
- **Adaptive multimedia presentation:** the media of the hypermedia is adapted. This can be either by transforming the actual media (e.g. reducing image quality) or by selecting among alternative media to represent the same conceptual information (e.g. an video or a textual description).

Adaptive Navigation:

- **Link hiding:** the link is still present, but its presentation is altered so that is appears as normal text to the user. In this way, the user is not overwhelmed by links that are (at the moment) irrelevant for him.
- **Link removing:** the link is removed. This technique can only be used when removal of the link does not invalidate the global navigation (i.e. reach ability of information is not compromised) and the link is not essential for understanding the surrounding information.
- **Link disabling:** the link is still visible to the user, but the associated link functionality is disabled (i.e. the user cannot *follow* the link).
- **Sorting:** (a list of) links are sorted usually according to the relevance for the user. Link sorting can also be applied based on other parameters: link usage, link label (e.g. alphabetically), …
- **Annotation:** links are annotated according to relevance for the user. Different colours may be used or any other visual cue (e.g. the traffic light metaphor is highly popular).
- **Direct guidance:** in direct guidance, the system determines the “next best page” for suggesting it to the user. Direct guidance most often takes the form of a next button being presented to the user.
- **Hypertext map adaptation:** some hypermedia systems provide the user with local or global maps of the hypermedium. Map adaptation techniques manipulate the presentation of such maps (e.g. granularity, presentation, …).
- **Link generation:** (automatically) generated links are added to the (pre-authored) hypermedia system. A well known example of link generation is the suggestion of other items to buy in e-commerce applications, based on previous buys of the user.

In [Scharl, 2001], Arno Scharl makes a similar distinction between adaptive presentation and navigation. In addition, he identifies *meta-level adaptation*, where the aim of adaptation is situated on the level of the system as a whole (e.g. provide overviews, helping the user in verifying his location, ..). Examples of meta-level adaptation include trails and guided tours.

For an in-depth discussion of methods and techniques in adaptive hypermedia we refer to [Brusilovsky, 1996; Brusilovsky, 2001; Scharl, 2001; De Bra et al, 1999a].

2.2.5 AHAM

The *Adaptive Hypermedia Application Model* (AHAM) is a Dexter based reference model for adaptive hypermedia which is the most widely adopted and referenced. It was developed by De Bra, Houben en Wu [De Bra et al, 1999b]. The AHAM reference model further specifies the Dexter Storage Model to allow the nodes and links to be adapted, according to the (individual) user. Therefore, it introduces the *Domain Model, the User Model and the Teaching Model* in the (Dexter) reference model, see figure 2.2. These models are described in the following subsections.

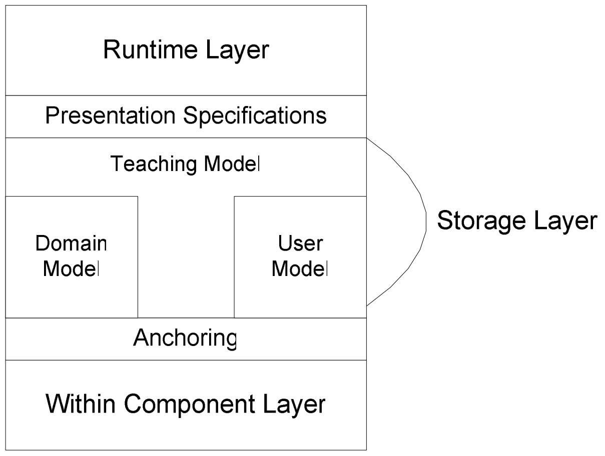


Figure 2-2 Overview of the AHAM Reference Model

Domain Model

The central notions in AHAM are *concept components (short: concept)* and *concept relationships*. Together, they form the Domain Model.

A ***concept*** is an abstract representation of an information item of the application domain. It consists of a *globally unique identifier* (uid) and *component information*, which contains a set of attribute/value pairs, a sequence of anchors and a presentation specification.

Concepts can be either *atomic* or *composite. Atomic concepts* are *fragments* of information, and are primitives to the model. Their internal anchoring is the concern of the within-component-layer (not specified by AHAM). *Composite concepts* have an attribute called 'children' which contains a sequence of concepts, and possibly a constructor attribute, describing structure within the concept.

Two kinds of composite objects are described in AHAM: *abstract composite objects*, containing only other composite objects as children*, and page (composite) objects*, containing only atomic concepts as children. In AHAM, each atomic concept is required to be a subcomponent of at least one page concept.

Concept relationships are the AHAM version of (Dexter) links. Although concept relationships will most commonly imply navigation, other types of relationships are possible. Concept relationships consist of a sequence of *components*, indirectly specified as *specifiers*. Specifiers are *resolved* by the *adaptive hypermedia engine* to unique identifiers. In the case of composite components, the resolver function adaptively traverses the concepts, until atomic nodes are encountered and presented using a presentation specification.

Concept relationships are components, consisting of a *globally unique identifier* (uid), a sequence of one or more *specifiers* and *component information*, which contains a set of attribute/value pairs, a sequence of anchors and a presentation specification.

User Model

The goal of the user model is to store and maintain knowledge about the user, in order to lead him to relevant information and keep him away from irrelevant information.

In AHAM, the user model is specified as an overlay model on the domain model: *each concept component* can be assigned an arbitrary number of *attribute/value pairs*. Examples of such attributes used in many Adaptive Hypermedia Systems (AHS) are "knowledge value", indicating how much knowledge a user has about a certain concept, and "read", indicating whether the user has read something (fragment, page, set of pages). Although intent of the attributes is to store the knowledge of the user about the domain model, the attribute/value pairs can be (ab)used to represent domain independent values, for example, the age of the user.

For each user, the AHS then stores a table, consisting of all concept components and their attribute and values for that particular user. On the basis of (the values in) this table, the AHS will attempt to determine which concept/pages the user is ready to see, and which ones he is not ready for. The logic for determining this is specified in the form of *pedagogical rules*, which are part of the teaching model, discussed in the next subsection.

Teaching Model

The teaching model is the "glue" between the domain model and the user model. It consists of a set of *pedagogical rules* which help determine how information is to be presented. Two kinds of pedagogical rules exist: *generic* and *specific* pedagogical rules.

A *generic pedagogical rule* consists of three sub-elements:

- a *triggered rule,* which is the actual rule
- a *phase,* indicating if the rule is to be executed before and during generation of the page (pre) or after generation of the page (post)
- a boolean *propagate* field, indicating whether the rule can trigger other rules.

A *specific pedagogical rule* contains one extra sub-element compared to the generic variant, namely the *set of concept components* that are used in the rule. Specific rules are generally applied to use and set attributes of specific concept components, while generic rules are not specific for a particular set of concept components.

Rules can be predefined in the AHS, or they can be added by the user. AHAM dictates that such author-define rules take precedence over predefined rules. Furthermore, specific rules have precedence over generic rules. Specific rules can thus be used to implement 'exceptions' of certain generic rules. The syntax of pedagogical rules is not specified in AHAM.

2.3 Web Site Design Methods

With the exponential growth of the World Wide Web, both in amount of users[26] as in amount of web sites online[27], and more and more different content providers (i.e. individuals, companies, organization, etc.) publishing more and more information, problems with ad hoc designed web sites became apparent. The most important problem is undoubtedly the problematic usability of web sites. In the beginning of the WWW, when web sites consisted of only a small amount of linked pages, the necessity for a well-considered, structured design was less important: the complexity of building a web site was well manageable by web designers and the structure of the resulting web site was easy to grasp for its visitors. With the aging of the WWW, this situation changed: current web sites are complex, professional applications, containing both static and dynamic, rapidly changing information and functionality. Without a systematic and transparent underlying design, designers have difficulty in coping with the complexity of web design and the resulting web sites become unusable: users fail to grasp the site organization, i.e. fail to build a mental model of the web site and consequently have trouble locating the information they are looking for. They feel “lost in hyperspace”.

To tackle this problem, the research community proposed *web design methods*. The first methods appeared in 1995 (i.e. RMM and OOHDM), and over the years, new improved design methods were introduced (e.g. WSDM, WebML, UWE, OO-H, Hera, …). These design methods allow the web designer to consider the different aspects of a web design (i.e. requirements, data, navigation, presentation, implementation) separately, thereby allowing him to better cope with the complexity of creating large web sites. Some of these methods also offer a design methodology, which guides the designer through the construction of the different design models. Design methods and methodologies assure a more logical and consistent site organization, and thus improve usability.

To further accommodate the (individual) user, most methods include some form of personalization: a mechanism to specify how the web site should adapt to a visitor (user). For this purpose, similar techniques are used as discussed in the previous section (Adaptive Hypermedia): a user model is maintained to store relevant information about the user and the appearance of the web site is adapted according to this user model.

In the remainder of this section, five representative web design models are discussed. For each method a general overview is provided, and their possibilities to model adaptive behaviour (mostly aimed at personalization) is discussed. Other web design methods exist (i.e. RMM [Isakowitz et al, 1995], OOWS [Pastor et al, 2003], OntoWeaver [Lei et al, 2005], OntoWebber [Jin et al, 2002], XWMF [Klapsing and Neumann, 2000], …), but they do not offer any (adaptation) functionality or techniques that are not provided by one of the five methods discussed below[28].

2.3.1 OOHDM / SHDM

The **O**bject **O**riented **H**ypermedia **D**esign **M**ethod (OOHDM) [Schwabe and Rossi, 1995; Schwabe and Rossi 1998] was one of the first web site design methods and, as its name suggests, is built according to the Object Orientation paradigm. Its successor, the **S**emantic **H**ypermedia **D**esign **M**ethod (SHDM) [Schwabe and Moura, 2003; Schwabe et al, 2004] differs from OOHDM in its use of semantic web technology (e.g. RDF(S) and OWL) to represent its various models. Both methods are

[26] See http://www.netcraft.com/ (access date: 20 June 2005)
[27] See http://www.internetworldstats.com/stats.htm (access date: 20 June 2005)
[28] Some methods have a different focus (i.e. OntoWeaver, OntoWebber, XWMF focus on semantic web development), other have had few or no recent research activity.

conceptually the same (i.e. they consist of the same design phases), we'll describe the latest version, SHDM in the remainder of this section.

The SHDM method is a model-driven approach to design web applications in five phases: Requirements Gathering, Conceptual Design, Navigation Design, Abstract Interface Design and Implementation (see figure 2.3).

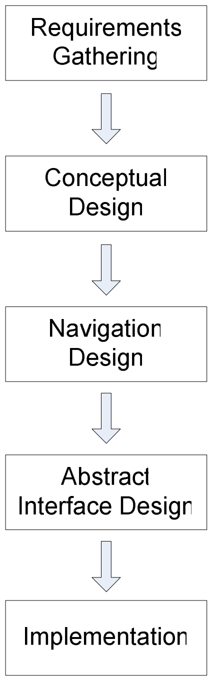


Figure 2-3 OOHDM/SHDM Overview

OOHDM/SHDM starts the *information gathering* phase by identifying the roles the user plays and tasks the web application should support. Subsequently, for each user (role), scenarios that describe the tasks of the user are described (informally). Scenarios describing the same tasks are grouped, and for each unique scenario, a use case is defined, expressing the interaction with the user. The designer therefore identifies which items are shown to the user without taking into account the internal aspects of the implementation. For each use case, OOHDM/SHDM introduces a user interaction diagram, which graphically represents the interaction between the user and the application. Finally, to validate each use case, the designer interacts with the user to obtain feedback and, if necessary, adjust the use case and the corresponding user interaction diagram [Güell et al, 2000].

In the next design phase, the *conceptual design*, the domain of the application is described. For this purpose, OOHDM uses typical object oriented modelling techniques (i.e. classes, aggregation, inheritance, …); SHDM uses semantic web technology (i.e. RDF and OWL).

During *navigation design*, the navigation structure of the web application is described. Two models are produced during this phase: the navigation class model and the navigation context model. The navigational class model defines what information can be reached; the navigational context schema defines how this information can be accessed. Navigation class models are views[29] on the application domain, where the navigational classes represent views of conceptual classes. A navigation class

[29] In SHDM, the views are defined using the RDF Query Language (RQL).

model is built taking the user tasks into account, and shows which information in the conceptual model can be reached by the user, and what are the relationships (the links) between the navigational classes. The navigational context model describes how navigation is organised in different contexts. A navigational context schema is built out of navigational context objects and access structures. Access structures are indexes (a collection of links) that allow the user to reach navigation objects within some context. Context objects refer to navigational classes and define how the user can browse the information in the current. In addition, the full definition of a context is given by a Context Definition Card (CDC).

The *abstract interface design* focuses on making navigation objects functional perceptible to the user. The abstract interface design only provides the support to model the information exchange between the user and the web site. Since this information exchange is driven by the tasks being supported, it is less sensitive to hardware and software specific aspects. In SHDM, the abstract interface design is specified using the Abstract Widget Ontology. At a more concrete level, the concrete interface design specifies the actual look & feel and layout of the web site. SHDM leaves this task to the graphic designer since it is almost totally dependent on the particular hardware and software runtime environment.

During the *implementation* phase, the interface objects are mapped to implementation objects, and the actual web site implementation code is outputted.

In the context of personalization [Rossi et al, 2001], OOHDM discusses *link personalization* and *content personalization.* Link personalization concerns the selection of links that are more relevant to the user (changing the navigation space); content personalization occurs when a page presents different information to different users. To model relevant information about the user, OOHDM adopts the *user* class in its conceptual model. Attributes of the user class can subsequently be used in the Navigation Class Model to adjust the information that is shown to the user (e.g. offer a price reduction to certain users). Link recommendations can be realized by creating a link in the Navigation Class Model, with as target a query on the conceptual model using a designated *recommendation* method (of the user class). The approach assumes the corresponding method (i.e. *recommendation*) is implemented. Note that OOHDM also allows, by use of its Navigation Context Model, to personalize the way nodes from the Navigation Class Model are shown to the user (e.g. different attributes may be shown to different users).

2.3.2 WebML

The **W**eb **M**odelling **L**anguage **(WebML)** is “a visual language for specifying the content of a web application and the organization and structure of such content in a hypertext” [Ceri et al, 2000; Ceri et al, 2002]. It is a model-driven design method which contains the following steps:

- **Requirements Analysis:** a description of the essential information about the application domain (e.g., business rules).
- **The Data Design:** this design phase produces the data model by making use of business rules and the requirements specifications.
- **The Hypertext Design:** this design phase produces the hypertext model which serves as a navigation model on top of the data model.
- **Implementation:** the actual implementation of all models.
- **Testing and evaluation:** test, evaluate and repeat each step if necessary.
- **Deployment**
- **Maintenance and Evolution**

Figure 2.4 schematically shows the different design phases of the WebML development process and denotes the dependencies between the different phases:

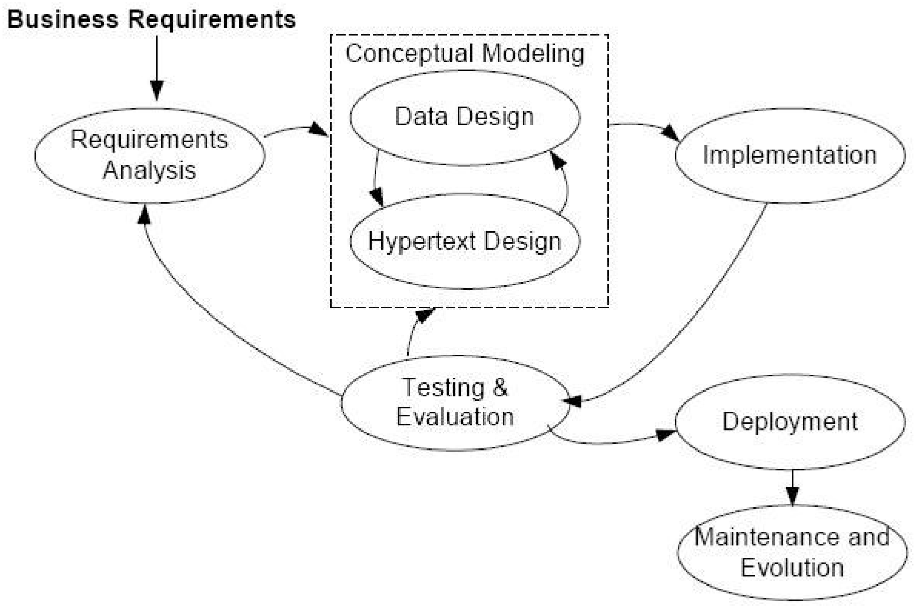


Figure 2-4 The WebML Development Process

Let's shortly discuss the relevant design phases:

WebML does not prescribe any specific format for *requirements analysis*. WebML designers are left to use (informal) tables, describing in an informal way the user requirements, or fall back on existing requirement analysis techniques.

The WebML conceptual modelling phase consists of two sub-phases: the *data design* and the *hypertext design*. The data design, expressed as an ER model [Chen, 1975], is concerned with the modelling of (user-) required information. The second conceptual modelling phase, the *hypertext design*, is concerned with the construction of a coherent navigation model for the web site. For this purpose, WebML uses site four key ingredients:

- **Content Units**: the elementary components of publishing. Content units either publish information extracted from a data source (i.e. each publishing unit is connected to a particular entity of the data model) or display a (web-)form with which content can be entered. WebML provides content units to represent a single data object (i.e. data units), a set of objects (i.e. multidata, index, multichoice, hierarchical and scroller units) or to enter information (i.e. data entry units).
- **Page**: the actual containers of information delivered to the user (e.g. a web page). Pages are typically composed of content units, and are themselves grouped in area's (see next). Pages can be nested if needed.
- **Area's**: a grouping mechanism for pages within a site view (see next), aimed to represent of group of pages logically belonging together (according to the business purpose). Area's can be nested if needed.
- **Site views**: the largest grouping mechanism, composed of area's and designed to address the specific needs of one particular user or a group of users with similar needs.

To model the navigation (i.e. the browsing in the resulting hypertext), WebML allows links to be placed between content units (placed in the same page or in different pages) and between pages. WebML distinguishes between contextual links (i.e. links that carry information) and non-contextual links.

To support operations on data, WebML prescribes *operations*. In contrast to content units, operations do not display information and are thus placed outside pages (in the hypertext model. WebML distinguishes some operations for content management (i.e. create, delete, modify, connect, disconnect), access control (i.e. login, logout) and communication (i.e. sendmail). Operations are either successful, after which an ok-link is followed, or unsuccessful resulting in a ko-link to be followed.

WebML applies standard approaches for presentation design, i.e. how the information is shown to the user (look and feel) and where it is placed on a page (layout). The layout of a page is organized with a page template skeleton which only contains the minimal HTML mark-up needed to define the layout grid of the page and the position of the various units in such a grid. The template skeleton is transformed with XSLT presentation rules. There are two kinds of such rules:

- Page Rules: match the outermost part of the skeletons layout (e.g., a table tag).
- Unit Rules: match the class of units (for instance, index units) and produce the markup for their presentation.

Note that the HTML produced by the XSLT rules exploits the CSS styles associated to the corresponding units, so that the XSLT rules are only concerned with layout and not with graphic properties. WebML comes with a tool called WebRatio Site Development Studio (WebRatio). This tool provides a GUI for the WebML design process and is able to organize the layout and presentation of a page.

WebML describes a personalization approach [Ceri et al, 1999a] consisting of four steps: profile modelling, delivery modelling, profile capturing and delivery execution. The Torii environment [Ceri et al, 1999b] supports these phases. Consistent with the data design, user profiles, which represent information related to users and groups, are modelled using ER. For each user the standard profile includes identification information, login, history, trace information (visited pages and time of visit) and group membership is stored. Profiles can be extended to fit a particular application domain. User groups group the individual users that are somehow related in groups (e.g. children) and are associated to site views. Using the delivery specification language Torii-DL, WebML allows using user profile data to personalize the information presented to the user. Event-Condition-Action (ECA) rules [Dayal, 1988] are used to capture the profile information. The considered events are *clickstream* (e.g. start and end of a session, page access) and *data change* events (resulting from data modifications operations). Conditions are predicates or queries over the database. Using ECA rules, users may be classified in user groups (based on profile information), data actions (e.g. adding/removing a currently selected item to a pre-defined profile attribute) can be performed, or information can be pushed to the user (e.g. informing the user of the availability of new items).

To personalize according to context (i.e. the interaction environment), WebML extends its application data schema (i.e. data design) with a *context model*, a representation of the meta-data needed to perform context-based adaptivity [Ceri et al, 2003]. Context models contain *context entities* (e.g. user device, location, …), which are associated to each user. Through *groups* (see previous paragraph) different site views are associated to groups of users. Using the context model, the application can, at runtime, switch between these different site views according to the adaptivity rules that are defined for the context.

The adaptivity rules primarily intervene on the level of the hypertext model. WebML tags pages that possibly need to be adapted according to context-attributes with a “C-label” (i.e. “context-aware”). These pages are augmented with a *refresh mechanism*, which automatically generates refresh events at given time intervals. This allows the application to react on context-changed occurring while the user is consulting a page. Also, some adaptivity actions are associated with the marked pages. These are either *context management* or *hypertext adaptivity* actions. WebML proposed three units to accommodate these actions:

- **Units for accessing fresh context data:** WebML assumes the context information is somehow available on the client machine and made available through URL parameters. Two new get units, URLParameter (provides access to an URL parameter) and get data unit (retrieves data to be used as input for another unit) are described.
- **Units for condition evaluation**: two units for condition evaluation are added: if and switch. Both units have incoming links possibly carrying parameters, which are subsequently used to evaluate an expression (i.e. condition). In case of an if unit, an ok-link is followed if the boolean expression evaluates to true and an alternative ok-link if it evaluates to false. Switch units provide a set of ok links, one for each switch-alternative. In both cases, a ko-link is followed when evaluation failed.
- **Units for site view switching:** a new unit, the ChangeSiteView unit, is responsible for switching site view (if context changes require this).

In a recent poster presentation on the WWW conference [Facca et al, 2005], WebML describes *reactive* web sites: web sites that allow automatic restructuring of application components according to the user profile. A Web Behaviour Model is specified as a timed state-transition automata for representing classes of user behaviour. For example, a behaviour can be specified, where the user visits a certain page A and within a certain time-limit browses to a page B. This behaviour can subsequently be used as a condition in an Event-Condition-Action rule. The event triggering the rule is a page request; the action part of the rule consists of any adaptive behaviour as described above. This work is in a preliminary stage, yet shows some similarities with the work presented in this dissertation (see section 2.4).

2.3.3 Hera

The Hera Methodology [Frasincar et al, 2002] [Vdovjak et al, 2003] is a model driven methodology for designing and developing Web Information Systems (WISs) using Semantic Web technologies. It is developed from a database perspective; the conceptual modelling (similar to the ER modelling) and the querying of the different Hera models are important issues in the proposed methodology.

The Hera methodology has two main layers: *the data collection layer*, and the *presentation generation layer*. The first layer helps in making available data coming from different, possibly heterogeneous data sources. The second layer is responsible for producing Web presentations tailored to the user and its browsing platform. Based on the principle of separation of concerns, these two main layers are further decomposed in more specific (sub)layers.

The data collection layer distinguishes two (sub)layers:

- **The conceptual layer:** a shared layer among the two main Hera layers as it acts as an interface between them.
- **The integration layer:** provides appropriate articulations (mappings) from the different data sources to the shared concepts identified in the conceptual layer.

The Hera presentation generation layer distinguishes two other (sub)layers:

- **The application layer:** defines the abstract hypermedia (navigation) view on the data in terms of an application model.
- **The presentation layer:** defines the presentational aspects of the WIS, needed to generate the actual implementation on a chosen implementation platform (e.g. HTML, WML, …)

There is also an orthogonal layer, the adaptation layer, that captures adaptation issues in all the above layers. For its particular importance to this dissertation, we go into closer detail on this layer at the end of this subsection, after explaining the Hera models.

Distinguishing these different layers, Hera differentiates at design level between the semantical aspects, the navigational aspects, and the interface aspects of a WIS. In each layer a design activity is responsible for producing an appropriate model that stores all design aspects specific to this phase.

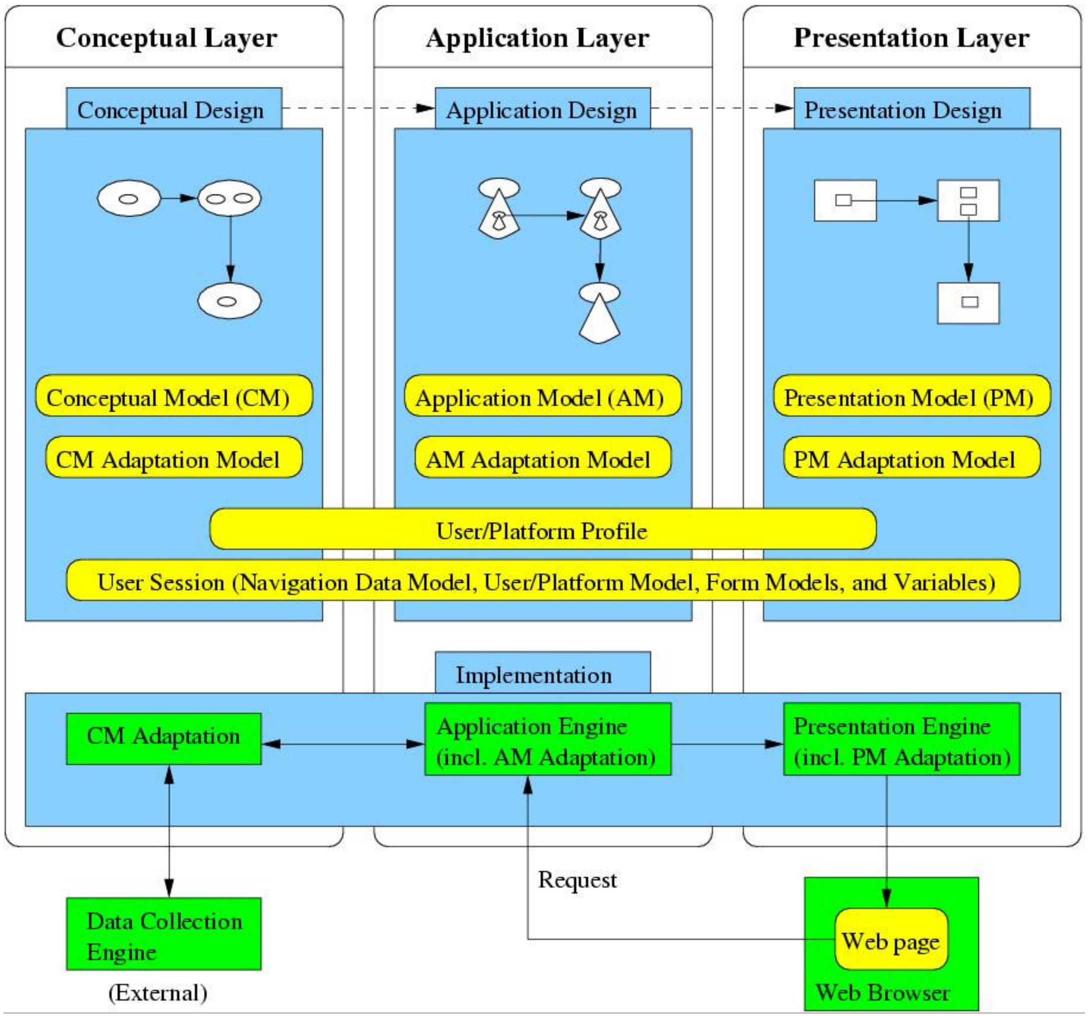


Figure 2-5 Hera Overview

There are four basic models within these layers to capture the information modelled:

- **Conceptual Model (CM):** defines the concepts specific to the application domain
- **Integration Model (IM):** the IM depicts appropriate articulations between these concepts and the concepts specific to the data sources that need to be integrated. The designer's background

knowledge regarding the data sources is captured in the decorations (e.g., availability of a certain data source) associated to articulations.
- **Application model (AM):** describes the navigational structure of the application. Basic element in this model is a slice, a meaningful presentation unit that groups attributes from possibly different CM concepts. Slices can be recursively defined.
- **Presentation model (PM):** The PM basic element is the region, an abstraction of a rectangular area of the user's browsing device to be used for presenting the information contained in a certain slice. Regions have associated with them layouts (e.g., a table layout or a vertical/horizontal layout) and a certain style (e.g., font size, font colour, link colour etc.). Regions can be recursively defined.

In addition to the basic models there are three other models:
- **User/platform profile:** stores the static user preferences and platform capabilities (fixed before the browsing starts)
- **User/platform model:** captures the dynamic user preferences and platform capabilities. The user model changes during user browsing. This model is also called the *session model.*
- **Adaptation model:** specifies how one can tailor the three basic models using the data coming from the user/platform profile and/or the user/platform model.

An overview showing the Hera design models is shown in figure 2.5.
Adaptation in Hera is done in the adaptation layer, a layer that spawns all other layer. This is done using by inclusion conditions attached to model elements. Conditions based on the user/platform profile attributes define the so called adaptability (presentation is fixed before user browsing) while the conditions based on the user/platform model define the so called adaptivity (presentation changes during user browsing) of the system [Frasincar and Houben, 2002] [Vdovjak et al, 2003].

Adaptation at conceptual model filters certain concepts or attributes from the conceptual model [Frasincar et al, 2004]. For example, a presentation on a WAP phone which is not image capable will not include all image attributes from the conceptual model.

Adaptation at application model suppresses certain slices and the links that point to them [Frasincar et al, 2004]. For example, an expert user has more information available in a presentation than a beginner.

Adaptation at presentation model chooses the appropriate layout based on the user's display [Fiala et al, 2004]. Also an appropriate style is chosen, for example, a user with a poor vision will have bigger fonts than a user with good vision. With respect to adaptivity one can dynamically change the presentation (using the same mechanisms as above) at runtime. For example, if the bandwidth of the network dramatically drops the images can be removed from the presentation. Also the user can be presented with information depending on his browsing/interaction history (e.g. forms, link following) [Houben et al, 2004] [Vdovjak et al, 2003].

Information specified in the adaptation layers is taken into account when generating the actual implementation for a WIS. The Hera methodology proposes a sequence of data transformations that leads to a Web presentation. Based on the static part of the adaptation model the four basic models are tailored to the particular user/platform situation before the presentation is generated. Next, there is a sequence of pipeline transformations (the output of one transformation is the input of the subsequent transformation) to generate different model instances.

First, based on the integration model and user query, a conceptual model instance is created using the available data sources. The conceptual model instance is then translated (based on the application model) into an application model instance and the application model instance is translated (based on the presentation model) into a presentation model instance. In the last transformation step the presentation model instance is converted to a format interpretable by the user browser (HTML, HTML+TIME, SMIL or WML). If the presentation is static (e.g., without forms or not considering the user browsing behaviour) the full Web presentation is generated at once. If the presentation is dynamic (e.g., with forms or considering the user browsing behavior) only one Web page is generated at a time and the Hera pipeline transformations are triggered for
each Web page request.

All basic models are represented in RDFS and basic model instances in RDF.

2.3.4 OO-H

OO-H (Object Oriented Hypermedia) [Gómez et al, 2000; Gómez et al, 2001] is a user-driven methodology based on the object-oriented paradigm and partially based on standards (XML, UML, OCL...). This approach allows the designer to develop web-based interfaces and its connection with previously existing application logic modules providing him with the semantics and notation necessary.

In OO-H four views are defined as can be seen in figure 1. The design process departs from the *domain view*, where the domain model is defined, provided by the UML use case diagram which captures the system requirements, and a class diagram that captures the domain information structure (and expresses the final data model). From there the *navigation view* is defined: different navigation diagrams (NAD, Navigation Access Diagram) instances are modelled for each user type allowed to navigate through the system. Each NAD instance reflects the information, services and navigation paths for the associated user's navigation requirements fulfilment. Once the NAD has been constructed, we can generate (if desired), following a set of mapping steps, a default web interface (default presentation model) that will constitute the OO-H *presentation view*. This automatic generation feature allows the designer to shorten the time necessary to develop application prototypes. However, final implementations usually require a much higher level of sophistication, both from the visual and the usability point of view. In order to help the designer to refine this structure while maintaining its quality, the Pattern Catalog contains a set of constructs proven to be effective solutions to so far identified problems inside web environments. Once the presentation model has been refined, a web application front-end, either static or dynamic, can be generated for the desired environment (HTML, WML, ASP's, PHP's...).

To define navigation and visualization constraints, OO-H uses the object constraint language [Warmer and Kleppe, 1998], a subset of the standard UML that allows software developers to write constraints over object models augmenting the model precision. OO-H associates such constraints to the navigation model by means of filters defined upon links.

The fourth view in OO-H is the *personalization view* allowing the designer to better tailor the site to the particularities and needs of the individual user [Garrigós et al, 2003 ; Garrigós et al, 2005]. This view affects to the rest of the views as it can be seen in figure 1. It is supported by a personalization framework that is part of the model. The framework can be instantiated by the web designer, and connected to any OO-H based web site to empower it with personalization support for (individual) users or user groups. This framework can be divided in two parts:

1. The user model which is part of the domain model and provides the designer with the means of gathering and storing all (updatable) information needed to personalize the site. It will

contain information related to user characteristics, context, requirements, as well as information related to the browsing behaviour of the user.

2. The personalization model that provides the designer with the means of specifying the personalization policy for the different users or group of users (adapting structure, content, layout and/or presentation) by means of Event-Condition-Action [Dayal, 1988] rules. These rules are specified using an effective language defined in OO-H called PRML (Personalization Rules Modelling Language). The purpose of this language is to help the web designers to define all the rules required to implement a personalization strategy.

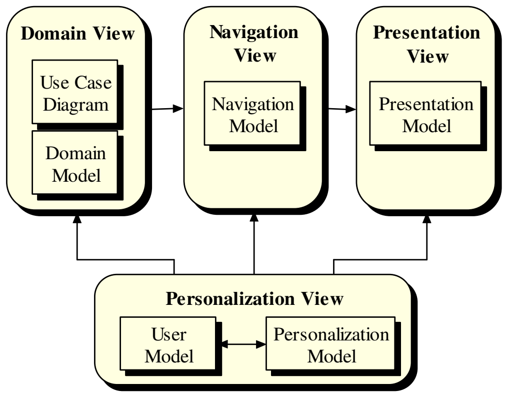


Figure 2-6 OO-H Overview

For satisfying a personalization requirement it is needed to define where and what information is acquired (stored in the user model at runtime) to obtain the required knowledge to personalize (acquisition rule) and define the personalization in terms of the effects this personalization causes in the system (personalization rules). There are three kinds of personalization rules: navigation rules (to alter navigation), content rules (to add/remove/adapt content) and presentation rules (to modify presentation). There are also profile rules to classify the users using the acquired information. These rules allow to define a group of users with similar features and to attach acquisition or personalization rules to those groups.

For integrating this personalization framework into the OO-H approach the dynamic execution has been changed adding a rule engine to interpret the personalization rules at execution time. Rules are defined in the navigation and presentation diagrams and then those diagrams (and rules) are compiled into an XML specification. The XML specification which contains the rules (with variable data) has to be interpreted by the rule engine, generating the variable interface logic that will contain the commands to be executed by the web server. The advantage of separating these rules from the rest of the application is that we can modify them in an independent way.

2.3.5 UWE

The UWE approach (Koch, 2001; Koch et al., 2001) supports the systematic development of web applications focusing on the specification of adaptive (personalized) applications. UWE is an object oriented approach based in the principles of the Unified Software Development Process [Jacobson et al., 1999]. The authoring process is iterative and incremental, and covers the whole development cycle of a web application. This authoring process consists of a notation and a method. The notation is the Unified Modelling Language. UWE also proposes a UML profile that is used along all the authoring process. This profile includes stereotypes defined for the modelling of navigation and presentation aspects of Web applications. An extension of ArgoUML known as ArgoUWE gives support to the approach. The fundamentals of this approach are a standard notation (UML through all the models), the precise definition of the method and the specification of constraints (with the OCL language [Warmer and Kleppe, 1998]) to increase the precision of the models.

The method consists of six analysis and design models. Figure 2.7 shows the (six) UML packages related by trace dependencies that represent the models produced by the UWE design process.

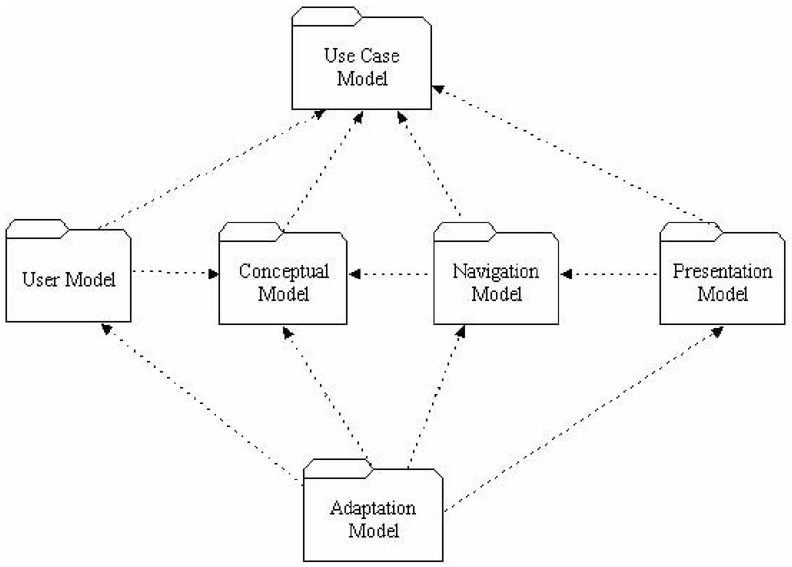


Figure 2-7 UWE Overview

The functional requirements are expressed as a UML use case diagram. It describes the users (actors) of the application and the functionality that has to be fulfilled for each of them. The conceptual model is built taking into account the functional requirements captured with the use cases. It is expressed as a domain model represented as a UML class diagram.

During the navigational design, the navigation model is built. This model is based on the conceptual model and it is the origin of two other models: the navigation space model, which specifies the objects that can be visited by navigation, and the navigation structure model, which specifies how to reach the objects defined in the navigation space model (using guided tours, indexes, etc). The presentation model defines both static and dynamic aspects of the presentation. This model is derived from the navigation structure model. The static aspects are modelled by a presentation structure model and an abstract user interface model. Moreover, a presentation flow model can be defined to model dynamic aspects of the presentation and optionally object lifecycle models (using respectively UML sequence diagrams and state diagrams).

UWE also provides a reference model for adaptive hypermedia systems (Munich reference model), based on the Dexter reference model. The purpose of this model is to identify the main features of adaptive hypermedia and personalized Web applications, as a prior step to the definition of appropriate modelling techniques. It is an object-oriented reference model, visually represented by UML and formally specified in OCL. It supports adaptation, adaptivity and even pro-activity features.

UWE stresses on personalization features, like the definition of a user model and an adaptation model, or a set of adaptive navigation features, which depend on preferences, knowledge or tasks that the user must execute. The static user model is represented as a class diagram, expressing the view the system has of the user. It includes the user attributes (and their values) relevant to the adaptive application. These attributes can be domain dependent (will have a value for each domain component) or domain independent attributes. State transitions of the objects (i.e. instantiation of the user model) can be expressed by rules of the adaptation model.

The adaptation model consists of containers for user behaviour and a set of rules. A rule is modelled by a condition, an action and an executor function. Rules are triggered by user behaviour (i.e. browsing, user input or user inactivity) or by other rules and they are based on the information of the user and domain models (as well as the user interaction activities). Depending on whether the rule is applicable to all instances of a domain class or just to a specific instance, two types of rules are distinguished: global or generic rules and local or specific rules. Rules are also classified depending on their objectives into construction rules, acquisition rules and adaptation rules. The construction rules find the appropriate concept on the basis of relationships. The Acquisition rules objective is to fill the user model gathering the (user) needed information. The adaptation rules specify conditions under which to adapt the content, navigation and presentation of a web application.

The adaptive content selects different information depending on the current state of the user model. As aforementioned, the actions of these rules can also update the user model (i.e. acquisition rules) taking into account the observations of the user behaviour.

The adaptive navigation's aim is to prevent users to follow irrelevant navigation paths. It consists of changing the navigation structure of the web applications, e.g. changing the link appearance, the link target or the number of links presented to users, as well as sorting them. In the navigation model, adaptive navigation can be performed (depending on the relevance for the user) by direct guidance, sorted links, removed links, annotated links and passive navigation. To specify their adaptive behaviour, associations showing navigability and access primitives (i.e. menus, guided tours, indexes) are annotated with properties based on the current state of the user model. For example, the property "{removed}" for an index or a menu denotes the links were removed according to the adaptation rules (which consider the current values of the attributes in the user model).

The adaptive presentation shows different layouts or different font size, image size… etc depending on the conditions specified in the adaptation rules.

2.3.6 Other Web Design Methods

Besides OOHDM/SHDM, WebML, Hera, OO-H and UWE, which have been explained in the previous sub sections, there is a range of other design methods available. Most notably, the emergence of the semantic web has caused so called semantic web design methods to arise. They differ from (classical) web design methods in their use of semantic web technology. However, from a method point of view, they are quite similar to the aforementioned methods. Therefore, we do not discuss each of them in detail here. A more detailed overview of existing design methods can be found in [Murugesan et al, 2001] and [Frasincar, 2005].

2.4 Situating this dissertation in the research field

In adaptive hypermedia, emphasis primarily lies on accommodating the individual user when accessing the hypermedia (i.e. personalization) [Brusilovsky, 2002]. In his early definition (see section 2.2.1, Adaptive Hypermedia - Definitions), Brusilovsky explicitly mentions the *user model* and the (individual) user [Brusilovsky, 1996], thereby implying that adaptive hypermedia (always) adapt to the individual user. The work in this dissertation does not include a user model (i.e. it does not model individual users) nor does it adapt the hypertext (i.e. the web site) to the individual user. Instead, access data[30] from *all* visitors is gathered and used to improve the web site as a whole, for *all* users (i.e. called *optimization* according to [Perkowitz and Etzioni, 1997a], as opposed to *customization*).

As described in section 2.3 (Web Site Design Models), in the context of adaptation, existing web design methods focus mainly on personalization, either by adapting content (e.g. offering price reductions to some users by adapting price), navigation (e.g. offering recommendations) or presentation (e.g. presentation adaptation according to browsing device)[31]. All methods described model the user in some way, and tailor their adaptation towards the current user (i.e. personalization). To model the actual adaptation, some methods use conditions to specify design alternatives according to user model attributes, others embed the adaptation in queries over the data model, others tag the design models with meta-information in order to adapt it, and some use rules to specify (part of) the adaptive behaviour. Modelling *when* adaptation should be performed differs from method to method. In some methods, execution of adaptation (rules) is hard-coded (e.g. when a page is loaded, the appropriate adaptation is triggered); other methods allow more fine-grained specification of timing for adaptive behaviour (e.g. general event-based rule triggering).

To point out the differences of our approach compared to existing adaptation methods in (other) web design methods, we first state what is *not* the aim of the work presented in this dissertation:

- **No personalization:** as already mentioned, in our approach there is no user model; adaptation is not geared to the individual user, but rather to improve the web site for *all* users. For this purpose, a Web site Overlay Model, a model that stores information about the (runtime) usage of the different (instantiations of) design elements is provided.
- **No content adaptation:** the focus of this dissertation is *not* on the adaptation of *content*. More specific, it is not our aim to add/delete/hide/alter attributes or concepts of the (conceptual) data model[32].
- **No presentation adaptation:** the focus of this dissertation is *not* on the adaptation of *presentation*. More specific, it is not our aim to alter pure presentational aspects of the web site (e.g. colours, fonts, layout, …).
- **No adaptation according to context:** the focus of this dissertation is *not* on the adaptation according to *context* (e.g. browsing device, location, …).

Instead, the work presented in this dissertation focuses on:

- **Design Adaptation:** adaptation in our approach is intended to (destructively) change the (static) structure and navigation of the web site, based on web site usage information, at the design level and this for all users.
- **Adaptation for all users:** in contrast to existing adaptation methods, which are mostly tailored to accommodate the individual user in browsing the web site (personalization), our approach aims to adapt the web site to accommodate *all* (or most) users.
- **Monitoring (pure) web site usage:** in contrast to existing methods, where an attempt is made to relate runtime events to conclusions regarding the knowledge or features of the individual user (e.g. reading a page results in updating a knowledge value about the domain concepts represented on that page), our approach monitors "pure" runtime usage (i.e. accesses to elements originating from the web site design, such as pages, nodes, object chunks, …), without trying to draw conclusions related to the knowledge of the user.

[30] Identified by [Kobsa et al, 2001] to be relevant for adaptive hypermedia, and adopted by Brusilovsky in [Brusilovsky, 2001]
[31] Note that different methods might have different names for the same kind of adaptation. For example, OOHDM calls recommendations *link adaptation*, while OO-H would call this *content-adaptation*.
[32] In WSDM, this is done using Object Chunks (see Chapter 3).

Instead, the collected usage data is used to draw conclusions on the effectiveness of the web site structure and navigation organization.

- **Fine-grained web site usage information gathering:** an important feature we would like to highlight is the ability, for the designer, to specify exactly which information needs to be gathered. The access information is also not stored in a generic format (e.g. like in a web site access log). Instead, it is specified by the designer in a usable form which information need to be stored, having the intended adaptation in mind. Therefore, some programming logic can be expressed when specifying information gathering rules (e.g. track the accesses to a certain page *done by a certain group of users*). We thus do not simply track all information (e.g. like a web site access log), because in that case , we are left with the difficult task of mining relevant access patterns and information from this access log.
- **Timing of adaptation:** our approach includes a generic mechanism to specify *when* adaptation should be performed. This is not based, as in traditional approaches, on some features of the user or the context (e.g. a change of a value in the user model, detection of a certain browsing device). Instead, it is based on time events (e.g. expiring of a time interval) or usage events (e.g. after a certain amount of sessions). Timing of adaptation can furthermore be recurrent (e.g. each month).
- **Automatic and non-automatic adaptation:** next to automatic adaptation of the web site design models, we also allow non-automatic adaptation, by alerting the web designer of (possible) detected problems

This dissertation responds to some of the AI-challenges set by Perkowitz and Etzioni in their paper "Adaptive Web Sites: an AI Challenge" [Perkowitz and Etzioni, 1997a]:

- **Meta-information:** by integrating this work in the Web Site Design Method (WSDM), meta-information about the design, the meaning and use of different page object, and the design intention of the designer, is available and can be used when specifying adaptive behaviour.
- **Specify adaptive behaviour during design:** by providing a high level language to (1) specify how the user interaction with the web site should be tracked, and (2) based on which usage patterns the site should be adapted, we provide a way to specify adaptive behaviour (to be performed at run time) at design time.
- **Interaction with web design(er):** adaptive behaviour is defined in terms of the design models, from which the actual implementation is generated. Adaptive behaviour can be performed automatically, by adjusting the design models, but it is also possible that changes are only suggested to the designer, which can accept or reject changes, and only re-generate the site if necessary.
- **Flexibility in transformation of site structure and navigation:** adaptive behaviour is specified using a high-level adaptation specification language, build on elementary design model transformation. This allows a flexible way to specify both simple as well as complex adaptation.

The approach as it is described in this dissertation can be identified as adaptivity according to both De Bra and Dieterich's definition (see section 2.2.2 – Adaptive, Adaptable, Dynamic). The identification of the need and the execution of adaptation are performed by the system. Step two (proposal of adaptation mechanism) is not applicable (only one kind of adaptation is considered, namely adaptation of the website structure) and step three (decision of which adaptation to conduct) is performed by the designer (i.e. not the user). See also Chapter 8 for a further discussion on this latter point.

In conclusion, the work presented in this dissertation can be described as an extension to a web engineering approach and aims to support design time specified adaptive behaviour for optimization of web sites (i.e. adapt the web site for *all* users) based on web site usage information.

Chapter 3: WSDM Informal Description

" Thus, the task is, not so much to see what no one has yet seen; but to think what nobody has yet thought, about that which everybody sees."
(Erwin Schrödinger)

The previous chapter discussed the relevant background and related work for this dissertation. The history of hypermedia and the World Wide Web was reviewed; an overview of adaptive behaviour in hypermedia systems and the best known reference models were given and finally, the most important Web Design Methods were discussed. In this chapter, the design method used as a framework for this dissertation, the **W**eb **S**ite **D**esign **M**ethod (WSDM), is introduced in an informal way. A formal description will be given in the next chapter.

The remainder of this chapter is structured as follows. First, a short introduction to WSDM is given in section 3.1. Next, the different phases of WSDM are discussed: section 3.2 presents the Mission Statement Phase, section 3.3 the Audience Modelling Phase, section 3.4 the Conceptual Modelling, section 3.5 the Implementation Design and section 3.6 the Implementation Generation phase. Finally, extensions to WSDM as described in this chapter are presented in section 3.7.

3.1 Introduction

The Web Site Design Method (WSDM) was introduced by De Troyer and Leune in 1998 [De Troyer and Leune, 1998]. It was the first so called *audience driven design method*, i.e. the design process takes as starting point the identification of the different target audiences and classifies them into a hierarchy according to their requirements. From this identification, the main navigation structure for the web site is derived. Concretely, for the visitors this results in a number of links on the homepage, each representing a different navigation path for a different kind of visitor (called audience track).[33]

Other important features of WSDM are:

- *Explicit conceptual design phase:* WSDM allows the designer to model the information and functionality needed in the web site before considering any presentation or implementation details. Also the conceptual navigation structure can be specified without the need to specify implementation details. For example, at the conceptual design phase, there is no need to specify the exact labels (names) of links or to indicate which information should be clustered onto a single page. By allowing the designer to focus on one aspect at a time, s/he can better handle the complexity of (large) web sites and arrive to a more complete and coherent web design. Furthermore, an explicit conceptual design allows to specify multiple presentations without the need to re-model the actual information/functionality or basic navigation structure.

- *Explicit implementation design:* WSDM separates between an (explicit) implementation design, and the actual implementation. The implementation design allows the designer to group information and functionality into pages, and to specify the layout of the pages. From these models, in combination with the information gathered during the conceptual design phase, a basic implementation can automatically be generated. Furthermore, the implementation can be generated in different formats (e.g. HTML, XHTML, PHP, WML, etc), without any extra effort from the designer. In this way, the designer is effectively shielded from the details of the chosen implementation platform.

[33] Until then, most web sites were designed using a data driven (i.e. the available data drives the design process) or an organizational driven (i.e. the internal structure of the organization of the web site drives the design process) approach.

- *WSDM is a methodology*: besides the different models and techniques provided to perform the different phases of a web site modelling activity, WSDM also provides the designers with a method on how to derive these models and how to use the techniques. This helps the designer to come to a consistent web site, with an easy to grasp navigation structure and containing all necessary information and functionality. This, on its turn, helps the visitors to build a coherent mental model of the web site, thus enhancing usability.

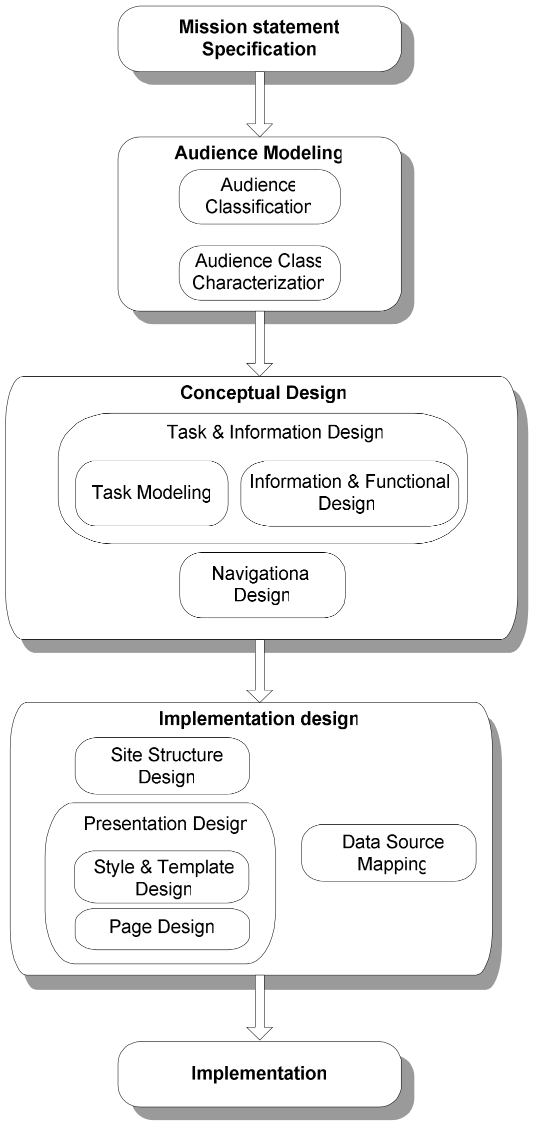


Figure 3-1 WSDM Overview

Figure 3.1 shows an overview of the different phases of WSDM. In the next sections, the different phases of WSDM are explained one by one. To further clarify each phase, an illustrative example will be given in each section. The examples are taken from a larger case study, solved with WSDM in the context of the IWWOST 2001 workshop [De Troyer and Casteleyn, 2001]. The case study comprises the design of a conference review web site, aimed at supporting the full submission, review, notification and announcement process of a conference.

3.2 Mission Statement

In the first phase of WSDM, the *mission statement* for the web site is expressed. In the mission statement, *the purpose*, *the subject* and the *targeted users* of the web site are specified. The mission statement establishes the borders of the design process. In the rest of the method it will allow to decide what information or functionality to include or not, how to structure it and how to present it. In addition, the mission statement can be used during validation to check if the web site has achieved the formulated purpose. By giving due consideration to the target users, it is assured that no potential target visitors are forgotten. The mission statement is formulated in natural language.

The mission statement for the conference review web site is given as follows:

"It is the purpose of the web site to support the overall selection process (submission by authors, evaluation and selection by the Program Committee) of papers for a conference".

3.3 Audience Modelling

During the Audience Modelling phase, the Mission Statement is taken as a starting point to analyze the different types of visitors, their requirements and characteristics. In the next two sub sections, we will elaborate on the two sub phases of the Audience Modelling phase: Audience Classification and Audience Characterization.

3.3.1 Audience Classification

In the first sub phase, the *audience classification*, the mission statement is taken as a starting point to identify and classify the different groups of expected visitors into so-called audience classes. For each type of visitor, their requirements are formulated (in natural language). WSDM distinguishes between three types of requirements: information requirements (requirement of the user of purely informational nature), functional requirements (requirements that define the functionality required to enable users to accomplish their tasks) and business requirements (requirement, typically of commercial nature, coming from the individual, organization or company for which the web site is built). As a classification criterion, the (information, functional) requirements of the different users are used: users with the same information and functional requirements are grouped into so called Audience Classes.

As an example from the conference review web site, consider the Audience Class "Author", with the following information, functional and business requirements:

Author
Functional and Information requirements:

- *Submit paper info and upload file*
- *Change submission until submission deadline*
- *Pre-register Co-author*
- *Information about the author's submissions*

Business requirements:

- *Show announcements for other conferences, related to the current conference, in which the author might be interested*

Subset relations between the set of requirements of the different audience classes give rise to a partial ordering relationship. Audience classes with the same and additional (information and functional) requirements (compared to another audience class) are called audience subclasses of the former Audience Class. Audience classes can thus be ordered in a tree-like hierarchy, the *Audience Class Hierarchy*. The root of this hierarchy is always the *visitor* audience class. The requirements of the visitor audience class represent the requirements shared among all users (possibly an empty set). More specialized audiences classes, having additional requirements compared to their audience parent class, can be found deeper in the tree (e.g. the leafs are most specialized audience classes, having additional requirements compared to all their ancestors).

Notation wise, Audience Classes are denoted using a schematic notation of a user (similar to UML use case notation); the Audience Sub Class relation notation is denoted similar to the inheritance relation in UML. As an example, the Audience Class Hierarchy for the conference review web site is given in figure 3.2[34].

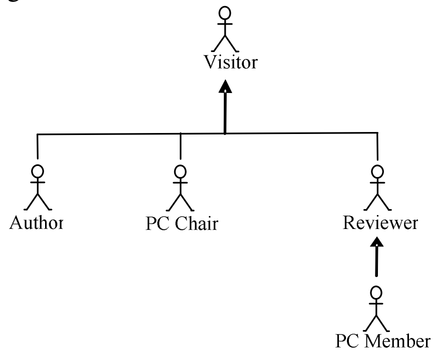


Figure 3-2 Audience Class Hierarchy for the Conference Review Web site

WSDM provides two alternative methods to derive the Audience Class Hierarchy for a web site. The simplest method consists of constructing an activity diagram to schematize the activities of the organization that are related to the purpose and subject of the web site. Activities are represented by ellipses; people involved in activities by rectangles. Next, the activities are further decomposed into elementary activities. The activities are decomposed in order to refine in each decomposition step the people involved. For each group of users identified in this way, the requirements can be formulated. By studying the subset relations between the requirements of the different groups, the Audience Class Hierarchy can be derived.

The second, more formal technique is based on the construction of an Audience Class Matrix. The rows and columns of this matrix represent the different (elementary) requirements of the different types of users. The matrix can be populated by the designer with Boolean values, by answering simply yes/no questions concerning the tasks the different types of users can perform. Using a partial order relation between the columns of the matrix, Audience Classes and Audience Subclass relations, can be identified, and thus the Audience Class Hierarchy is automatically derived. Details on the algorithm can be found in [Casteleyn and De Troyer, 2001].

[34] Note that, for the sake of simplicity of the proposed examples, authorization issues involved for the conference review web site are not taken into consideration. See section 3.7 (Extension to WSDM) for a further discussion on this issue.

3.3.2 Audience Characterization

In the second sub phase, the *audience characterization*, for each Audience Class identified in the Audience Classification phase, the relevant characteristics and possible navigation and usability requirements are summed-up. Example of characteristics include “average age > 50”, “minors” (i.e. age < 18), “colour-blind”, “Dutch speaking”, etc. Examples of navigation and usability requirements include “all information should be available in maximum 5 clicks”, “a link to the homepage should always be available”, “for each date input field, there should be an indication of the input format”, etc. Characteristics, navigation and usability requirements are specified using natural language. This information can later be taken into account when deciding how much information is put on a page (e.g. site structure design, see section 3.5.1), how to present information/functionality, or when creating the layout of the web site (e.g. for children, flashy colours and animations can be used; for colour blind people, colours should not be used to convey information).

3.4 Conceptual Design

During the *conceptual design* phase, we analyze in detail the requirements of the visitors, model their required information and functionality (task & information modelling), and decide how the available data/functionality will be structured and how the (different) visitors will be able to navigate through the site (navigation modelling). In the next two sub sections, we will provide more details on these two sub phases.

3.4.1 Task & Information Modelling

The intention of the Task and Information Modelling phase is to model in detail the different tasks each audience class needs to perform (Task Modelling), and to formally describe the data/functionality that is needed to fulfil those tasks (Information and Functional Modelling). These two sub phases, which may be performed in parallel, are described in the next sub sections.

3.4.1.1 Task Modelling

The goal of the Task Modelling phase is to analyze and model in detail the different tasks each Audience Class needs to be able to carry out. Each requirement will be supported by means of a user task. For each such task, a task analysis specifying exactly which steps are necessary to support the particular task is performed. Each task is decomposed into elementary tasks. During Information & Functional Modelling (see next sub section), for each elementary task, a data model describing the data/functionality needed to perform this task, will be constructed.

In WSDM, the task modelling is performed using a slightly modified version [De Troyer and Casteleyn, 2003b] of the Concurrent Task Tree (CTT) notation [Paterno et al, 1997 ; Paterno, 1999]. The CTT notation stems from the Human Computer Interaction (HCI) research field. In addition to hierarchical ordering and conditioning of tasks and subtasks, as in classical task analysis (e.g. Hierarchical Task Analysis (HTA), see [Dix et al, 1998]), the CTT notation allows to specify temporal relation between (sub)tasks. This additional expressiveness can be exploited when deriving parts of the navigation model (further explained in section 3.4.2, Navigation Design).

The temporal relationships specified in CTT are the following (WSDM changes the meaning of some relationships slightly; when this occurs, it is mentioned explicitly below):

- ***Order independent*** (T1 |=| T2): the tasks can be performed in any order.

- ***Choice*** (T1 [] T2): one of the tasks can be chosen and only the chosen task can be performed.
- ***Concurrent*** (T1||| T2): the tasks can be executed concurrently, for example browsing a file and printing a file
- ***Concurrent with information exchange*** (T1|[]| T2): the tasks can be executed concurrently but they have to synchronize in order to exchange information, for example editing a file and scrolling its content. They need to exchange information because it is possible to edit information that is visible in the scrolling.
- ***Deactivation*** (T1 [> T2): the first task is deactivated once the second task is started, for example once an item is deleted from the shopping cart it cannot be edited anymore.
- ***Enabling*** (T1>> T2): the second task is enabled when the first one terminates, for example consulting the database is possible after logging in.
- ***Enabling with information exchange*** (T1 []>> T2): the second task is enabled when the first one terminates but in addition some information is provided by the first task to the second task, for example checking a password is possible after the password was entered and the value of the password should be provided for checking it.
- ***Suspend-resume*** (T1 |> T2): indicates that T1 can be interrupted to perform T2 and when T2 is terminated, T1 can be reactivated from the state reached before the interruption
- ***Iteration*** (T*): the task can be performed repetitively. Note that in CTT the meaning is different. There, the meaning is that the action is performed repetitively: When the action terminates, it is restarted automatically until the task is deactivated by another task. The interpretation here is that the task can be repeated several times.
- ***Finite iteration*** (T(n)): if the task has to be repeated a fixed number of times (number known in advance).
- ***Optional*** ([T]): indicates that the performance of the task is optional
- ***Recursion***: occurs when the sub tree that models a task contains the task itself. This means that performing the task can be a recursive activity.

We found it useful to extend this list with an extra operator to express a transaction:

- ***Transaction*** (-> T <-): the task must be executed as a transaction. This means that if the task or in case of a complex task one of the tasks in the task's sub tree is not completed successfully, the task will not be successful and all activities should be rolled back.

The symbolic notation for these temporal relations, shown between brackets in the enumeration above, is used between two sibling tasks.

CTT distinguishes between four categories of tasks:

- ***User tasks:*** these are tasks that are performed by the user without using the software system (e.g. thinking how to solve certain problem, making some sketches on paper, etc). Although useful in classical HCI analysis, we **do not** consider these tasks in WSDM, as they do not directly influence the design of a web site.

- ***Application tasks:*** these are tasks that are performed by the application. Application tasks can supply information to the user (e.g. validate a login).

- ***Interaction tasks:*** these are tasks that are performed by the user, by interacting with the system (e.g. filling in a form, pushing a button, etc.)

- ***Abstract tasks:*** these are tasks that model a complex activity, and require further decomposition (e.g. perform a payment)

The graphical notation for the different categories of tasks is given in figure 3.3:

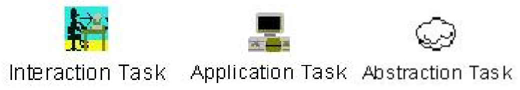


Figure 3-3 Task Categories supported in WSDM

WSDM imposes one other small change to the original CTT notation:

- CTT prescribes that if the children of a task are of different categories than the parent task, that parent task must be an abstract task. WSDM does not follow this rule. Instead, the category of the task is used to explicitly indicate who will be in charge of performing the task. E.g. for an interaction task, the user will be in charge; for an application task the application will be in charge. In this way, we can indicate for example who will initiate a subtask, or who will make a choice between possible subtasks

As an example in the conference review web site, the task model for the task "Assign papers to PCMembers" for the PC Chair Audience Class is given in figure 3.4. From the task model, we see that the task of assigning a paper to a reviewer starts with a choice for the PC Chair between three subtasks: assigning a paper to a PC Member, inspecting PC Members or inspecting Papers. When inspecting PC Members, the PC Chair is first presented with an overview of all PC Members, from which he can subsequently select a PC Member to see his/her details (e.g. which papers are already reviewed to this PC Member, what are his interests). Inspecting papers is done in a similar way.

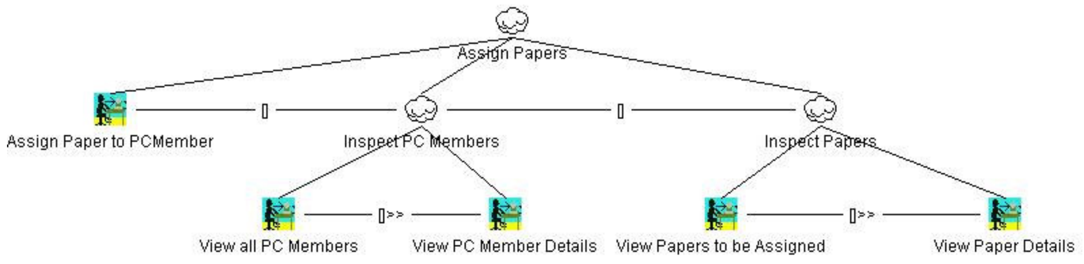


Figure 3-4 Task Model for the "Assign Papers" Task Using the CTT Notation

For clarity, we list all adaptations made to CTT for WSDM:

1. User tasks are not considered
2. Iteration temporal relationships do not require termination of another task
3. Transactions, a set of tasks that have the *all or nothing* property
4. Children of a task need not to be of the same category as the parent task; the type of the task is used to indicate who is in charge

3.4.1.2 Information & Functional Modelling

The goal of the Information & Functional Modelling phase is to formally describe the data and functionality that is required by the different Audience Classes. More specifically, for every elementary task (representing an elementary requirement) that was obtained during the Task Modelling phase, a small conceptual data model is constructed, which exactly describes the information/functionality needed to fulfil this task. Such a conceptual data model is called an *Object Chunk*. WSDM uses a slightly modified version of **O**bject **R**ole **M**odelling (ORM) [Halpin, 2001] to model object chunks. We call these models chunks because they can be seen as parts of a larger model (i.e. the Business Information Model). For requirements dealing with non-structured information, WSDM also provides 'TEXT' or other multi media types of object chunks.

Object **R**ole **M**odelling (ORM) views the world as objects playing roles. The basic building blocks of ORM are Object Types (OT) (graphically represented as circles) and (usually binary) relationships composed of roles (graphically represented as a rectangle composed into boxes, each box connects with a line to the corresponding OT). Identifiers (represented by lines on roles) are used to indicate one-to-one, one-to-many or many-to-many relationships. A mandatory role is indicated by a dot. Object Types are divided into Lexical Object Types (graphically represented by a dashed circle) and Non Lexical Object Types (graphically represented by a solid circle). In contrast to ORM, WSDM types the Lexical Object Types with one of seven primitive (multi media) types: email, resource, image, audio, video, string and integer. These primitive types are used in the presentation design to uniformly treat the different types of data and to fully automatically generate the actual implementation from the different design models.

Because WSDM needs to be able to refer to instances of Object Types, the ORM notation is extended in the following way. To refer to object instances we use referents. E.g. these referents can be used to indicate a transfer of information among Object Chunks (in the implemented web site, this will correspond with the passing of parameters from one page to another page). A typical example is a page showing basic personal information of a person, where it is possible to navigate to the homepage of that person.)

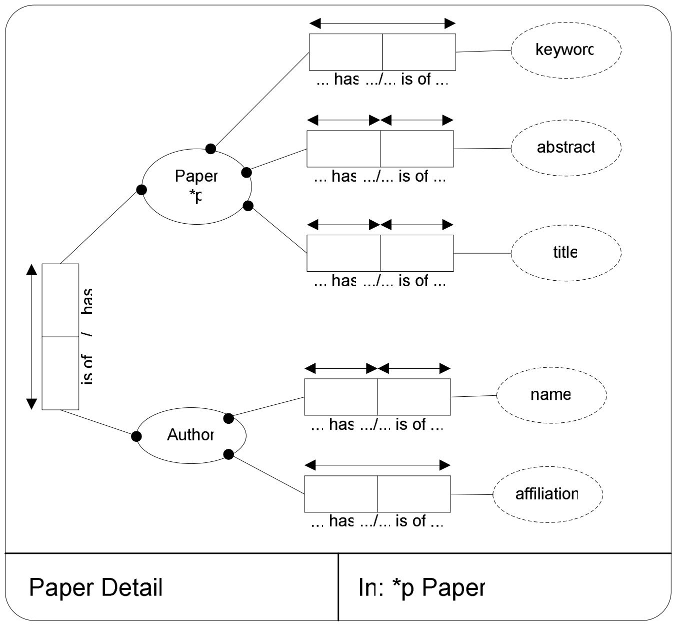

keyword
... has .../... is of ...
abstract
... has .../... is of ...
Paper
*p
title
... has .../... is of ...
is of / has
name
... has .../... is of ...
Author
affiliation
... has .../... is of ...
Paper Detail
In: *p Paper

Figure 3-5 Example Object Chunk "Paper Detail"

A referent is placed inside the circle depicting its Object Type (e.g. '*p'). Sets are represented by the traditional set brackets '{' and '}', e.g. {*p} placed in the Object Type represents a set of instances of this Object Type. The traditional set operators ('∪', '∩', '/') can be used on sets. To represent a relationship between instances, we can place the referents either in the Object Types (if no confusion is possible) or in the boxed of the roles if there are several relationships between the particular Object Types.

As an example Object Chunk from the conference review web site, consider the object Chunk "Paper Detail", relevant for the PC Chair when assigning papers to PC Members (see figure 3.5). The Object Chunk specifies what exactly should be displayed when the PC Chair chooses to view the details of a particular paper (hence the incoming referent *p, representing a unique Paper): the abstract, keywords for the paper, title, name and affiliation of the authors of the paper.

To model (basic) interaction with users (i.e. functionality), WSDM adds the following notation to ORM:

- **!** and **!!**: indicates that the user must select an instance of the particular Object Type. ! indicates the selection of one instance, !! indicates the selection of multiple instances. E.g. !{*p} would be used to denote a selection of a single instance from a set {*p} of instances.
- **?** and **??**: indicates that the user must interactively provide an (arbitrary) value. ? denotes the user can enter one value, ?? denotes the user may enter multiple values. Values obtained in this way can be assigned to referents, e.g. *p = ?.
- *p **->:** indicates that the value of referent *p can be changed by the user.
- **NEW** *p**:** indicates that a new instance of the Object Type to which the referent *p is refereeing is created.
- **DELETE** *p**:** indicates that the current instance of the Object Type to which the referent *p is refereeing is deleted.

Some other built-in functions and variables can be used, e.g. 'EMAIL', 'CurrentDate', etc. They can be used (textually) where required.

For clarity, we list all adaptations made to ORM for Object Chunk modelling in WSDM:

1. **Primitive types:** Lexical Object Types are typed with the primitive (multi media) types email, resource, image, audio, video, string and integer
2. **Referents:** To point to Object Type instances, referents are used. Referents are used to express functionality, to transfer information among Object Chunks, or to specify interaction with the user (see following).
3. **User Interaction:** to support basic user interaction, WSDM adds notation for instance selections, and user input.

As a method to specify the Object Chunks, the WSDM designer can rely on the Task Models to distinguish which Object Chunks need to be modelled (i.e. one Object Chunk for each elementary task). For the actual modelling of Object Chunks, the designer can use the method described in [Wintraecken, 1990] to derive relevant Object Types and Relations among them from a piece of text. Simply put, each adjective used in the informal description is a potential Object Type, while the verbs used are potential relations between the Object Types. The informal description of the requirements, specified during Audience Modelling, serve as input for this process.

3.4.2 Navigation Design

The goal of the Navigation Design is to define the conceptual structure of the web site and to model how the different audience classes will be able to walk through (i.e. navigate) (the information and functionality presented on) the site. The central navigation entities are nodes, which have zero or more object chunks connected to it, and links between these nodes. A node is a conceptual navigation entity, referring to information/functionality (by means of the object chunks) that logically belongs together. There are two specialized nodes: the root node, which is the root of the navigation hierarchy, and external nodes, representing external resources.

Links provides connections between nodes. They can be conditioned, i.e. the existence of the link depends on the evaluation of the associated condition (e.g. constrain the link for certain users, time frames or output devices), or transfers certain information, i.e. by using referents (see section 3.4.1.2, Information & Functional Modelling). WSDM distinguishes four different types of links:

- ***Structural links****:* these links provide the actual structure of the information and functionality being offered on the site. Well-known hypertext organizational structures are linear, hierarchical, pure web and grid (dual linear structure).
- ***Semantic links****:* these links represent semantic relationships that exist (in the universe of discourse) between the concepts represented by the nodes involved. As an example of a semantic link, consider a university web site where a visitor is able to browse from a lecturer to the courses s/he teaches, because there exists a “is taught by” relation between courses and lecturers.
- ***Navigation aid links****:* these links are put on top of the existing structural link structure, and are aimed to better facilitate navigation for the visitor. In principal, the visitor is able to navigate the web site using only the structural links and his browsers navigation support, but the navigation aid links provide him with useful shortcuts and aids. Well known example of navigation aid links are home and landmark links.
- ***Process logic links****:* these links connect two or more nodes to express part of a workflow or an invocation of an (external) functionality. An obvious example is the step-by-step payment process in most e-commerce sites, where the nodes representing these sub-steps are connected using process logic links.

A detailed description of this purpose based link categorization, its benefits and each different link type can be found in [Casteleyn and De Troyer 2002, De Troyer and Casteleyn 2003a].

Note that the navigation model only provides the *conceptual* structure of the web site. The actual pages are specified during Site Structure Design in the Implementation Design phase (see section 3.5.1). Different site structures can consequently be defined upon the same conceptual model, possibly targeting different devices, contexts or platforms. Another interesting observation is that links are defined between nodes. The actual anchors for the links are not provided during Conceptual Design (e.g. provide a link on the *name* of a person, on his *picture* or on both). These will only be specified during Implementation Design (see section 3.5). This allows the designer to focus exclusively on the intricate task of creating a transparent and coherent site structure, without being burdened with the details of implementation.

WSDM provides a clear method to derive the basic navigation model for a web site. Conform the audience driven philosophy, WSDM aims to create different navigation tracks for the different audience classes, containing all and only the information and functionality relevant for the particular audience class. These navigation tracks are also called *audience tracks*. The basic navigation structure reflecting these aims is obtained by a one-to-one mapping of the audience class hierarchy to a navigation model, where each audience class is mapped on a node, and each subset relation between audience classes is mapped to a (structural) link. The Audience Matrix algorithm[35] can be used to derive this navigation structure automatically. More details can be found in [Casteleyn and De Troyer, 2001].

The basic navigation model for the Conference Review web site, derived from Audience Class Hierarchy given in figure 3.2, can be found in figure 3.6. In this figure, double rectangles represent the root of an audience track (i.e. a node) and arrows represent links. As already mentioned, in this illustrative example we have abstracted from the authorization issues involved to correctly identify authors, reviewers and PCMembers and PCChairs. For a further elaboration on this issue, see section 3.7 (Extensions to WSDM).

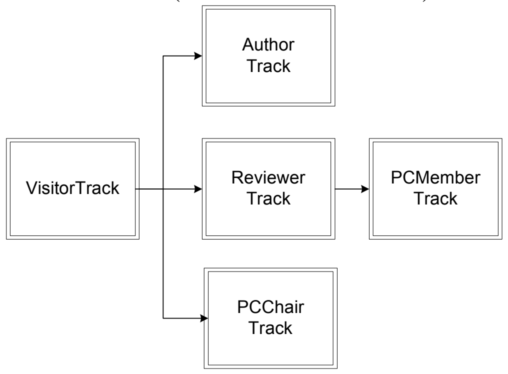


Figure 3-6 Basic Navigation Model for the Conference Review Web site

To define the internal structure within an audience track, the task models of the audience class are considered. As each task model resulted from a requirement of a particular audience class, considering the task models for modelling the audience track will result in the correct information and functionality offered in the corresponding track. To define the structure between the internal nodes, the temporal relations between the different tasks in the task models can be used. WSDM does not provide a fully automatic algorithm to derive this internal structure, but a semi-automatic approach can be found in [De Troyer and Casteleyn, 2003b].

As an example from the Conference Review web site, consider the elaborated Author track in figure 3.7[36]. Nodes are represented by rectangles, (root of) audience tracks by double rectangles, object chunks by rounded rectangles, and links by arrows between nodes. As can be seen from the figure, when entering the Author Audience Track, the author is presented with a choice to register a new paper, or to view his submissions. Registering a paper must be followed by adding co-author(s) and/or submitting paper info and file (after which the author can see his submissions). When inspecting submissions, authors can view detailed info, and update their submission if desired. Note that the Navigation Model does not yet specify actual

[35] Discussed in section 3.3.1, Audience Classification.

[36] Elaborated to fulfil the requirements stated in the IWWOST 2001 workshop (see [De Troyer and Casteleyn, 2001]).

(web) pages; deciding which node(s) come on which page is done during implementation design (see next section).

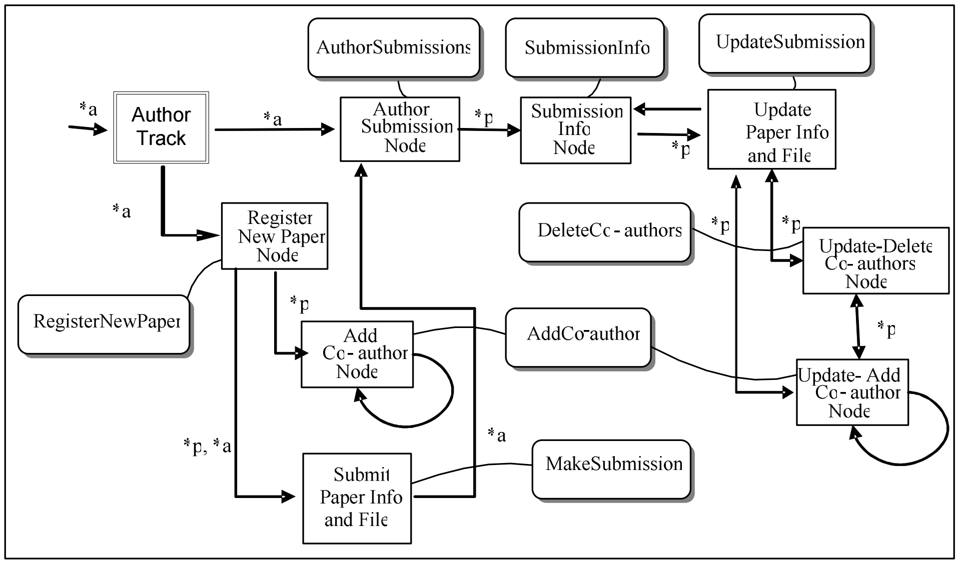


Figure 3-7 Elaborated Author Track for the Conference Review Web site

The links conveyed in this figure are structural links, as they define the (internal) structure of the Author Track. Multiple Navigation Models can be used when multiple link types are required, each representing the links of one particular type. Alternatively, the different link types can be textually annotated in one figure.

3.5 Implementation Design

The goal of the implementation design is to enrich the conceptual design models, in an abstract way, with the necessary details needed for the actual implementation. The implementation design consists of three sub phases: the Site Structure Design, the Presentation Design and the Data Source Mapping. From the models specified during Conceptual and Implementation Design, an implementation in the chosen implementation platform, can be generated automatically. The remainder of this section describes the different sub phases of the implementation design.

3.5.1 Site Structure Design

The goal of the Site Structure Design is to decide which conceptual nodes defined in the navigation design will be grouped into pages. The characteristics, usability and navigation requirements of the different audience classes are taken into account when deciding which information/functionality to group on a page. E.g. if a particular audience class has a characteristic "age > 60" specified, the designer might want to take this into account and limit the amount of nodes grouped on a page. Different Site Structures can be defined for one single design, each supporting e.g. a different device, context or platform.

Pages specified during the Site Structure Design are abstract pages (also called "page types"). Each abstract page will possibly give rise to a set of concrete pages (page instances) when the actual implementation is generated. E.g. a Submission Info Page will give rise to several page instances, one for each submission.

A Site Structure Model thus specifies for each node of the navigation model the page on which it is placed, and the links between the pages. Note that the links between pages are already implicitly present through the links defined between the conceptual nodes. We do not need to specify them again. Links between nodes grouped on the same page will disappear in the implementation or may become in-page links.

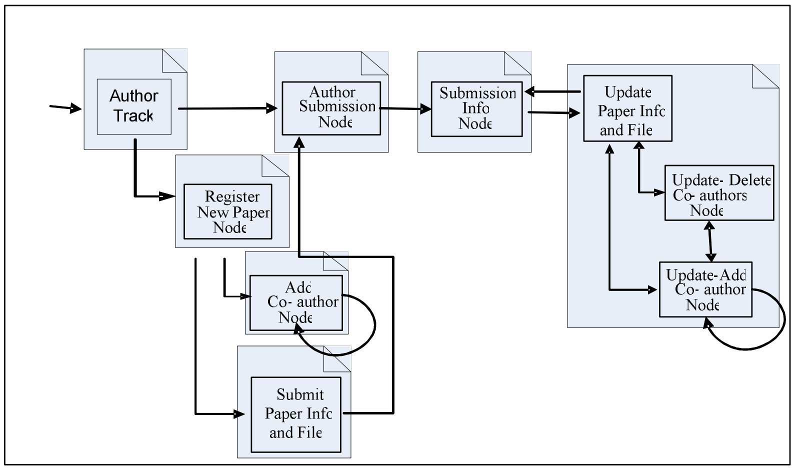

Author Track
Author Submission Node
Submission Info Node
Update Paper Info and File
Register New Paper Node
Update- Delete Co- authors Node
Add Co- author Node
Update-Add Co- author Node
Submit Paper Info and File

Figure 3-8 Site Structure Design for the Author Audience Track of the Conference Review Web site

The Site Structure Design is schematically presented by adding abstract pages to the Navigation Model, grouping the desired nodes on each abstract page. The graphical notation for an abstract page is a rectangle, where the top right corner is folded. As an example of a Site Structure Design, consider the Audience Track for the author Audience Class of the conference review web site, given in figure 3.7 of section 3.4.2. A possible Site Structure Design is given in figure 3.8.

3.5.2 Presentation Design

The goal of the Presentation Design is to describe the layout (i.e. positioning & style) of all pages of the web site. While describing the layout, the designer also has to decide on which particular primitive presentation concept (see next sub section) links, defined during navigation design, should be placed. The presentation design is divided in two sub-phases: the Style & Template Design, aimed at designing page templates for the web site, and the Page Design, aimed at ordering and presenting the actual information on these pages (using the created templates).

Both the Style & Template and Page Design are based on a (minimal) set of primitive presentation concepts, generic enough to specify the most common layout of web pages. Due to their particular importance in the WSDM Presentation Design phase, these primitive presentation concepts will be explained in the next sub section (3.5.2.1). Subsequently, we resume the informal description of the Presentation Design sub phases, by describing the Style & Template Design phase in sub section 3.5.2.2 and the Page Design in sub section 3.5.2.3.

Primitive Presentation Concepts

WSDM provides a layered set of presentation concepts, which can be used for the presentation design. The base of this set is a minimal set of primitive presentation concepts, that is expressive enough to represents the most common layouts. On top of these primitive presentation concepts, more intuitive higher level concepts are defined (see next sub section). The set of primitive presentation concepts is important for the implementation generation: as a WSDM design targets platform- and implementation language independence, the Implementation Generation process (see section 3.6, Implementation Generation) is able to generate different formats. For the implementation generation, to support a new platform, it suffices to add code generation for this minimal set of primitive presentation modelling concepts.

The set of primitive presentation modelling can be divided into three groups:

- Multi Media Elements: the primitive data, in a specific format, used on the webpage
- Basic Positioning Concepts: used to position data on a web page
- Interaction Concepts: representing form elements and action controls

The first group of the primitive presentation concepts consists of the ***multi media elements***. These primitives modelling concepts represent the primitive data supported by WSDM (i.e. the *actual* data). The multi media types most commonly found on web pages, are supported: integer, string, email, image, video, audio and resource. Where appropriate, some additional properties can be specified for in order to display the multi media correctly (e.g. height and width for an image). Note that (the type of) these multi media elements correspond to the types introduced for Lexical Object Types in Object Chunks. Indeed, ultimately, the instances of Lexical Object Types are multi media elements, and will be represented on the pages as (multi media) data.

The second group of primitive presentation modelling concepts provides primitives for positioning elements on a web page. For this purpose, WSDM uses the ***positioning concept*** *(positioning) grid*. A positioning grid (short: grid) contains one or more rows, and each row contains one or more cells. Absolute and relative height/width can be specified for grids, rows and cells. A cell contains either primitive data (i.e. a multi media element), a reference to primitive data (specified by means of a reference to a Lexical Object Type defined in an object chunk), or another (nested) grid. In this way, it is possible to position any set of elements at will (note how this mechanism is very similar to Java's AWT positioning method, which also allows arbitrary positioning of GUI elements). Furthermore, each cell can be augmented with a reference to a navigation link. Deciding the exact source (i.e. the content of a cell) of a link, defined in the navigation design phase during conceptual modelling, is thus done during implementation design. We highlight that a grid is used to specify how the information and functionality specified by means of an object chunk is laid out on a web page i.e. the rows and cells of a grid allow positioning the Lexical Object Types of the object chunks.

The third group of the primitive presentation concepts are the ***interaction concepts***, which allows modelling form elements and action controls. Currently[37], WSDM makes a distinction between *select controls* (i.e. controls to make a selection out of choices), *input controls* (i.e. controls to enter a certain value or values) and *action controls* (i.e. controls to perform an action). Types of select controls are radio button, check box, list box and drop down box. Input controls are text box and 'secret' text box (typically used for entering passwords). Currently, the only action control supported is the push button. It should be noted that, at this moment, not all possible interaction concepts are supported. More concepts may be added in the future.

[37] This is work in progress.

Let's now discuss the two sub phases of the Presentation Design: Style & Template Design, and Page Design.

Style & Template Design

The goal of the Style & Template Design is to design the different kinds (i.e. templates) of pages that will be used in the web site. These templates can subsequently be used in the next sub phase, the Page Design, when for each (abstract) page the specific positioning and presentation for the nodes and chunks present on that page will be specified. Typically, a web site requires different kinds of pages each represented by a specific template, e.g. a homepage template, a title-page template (also called a "hub"-page), leaf page templates, etc. In the audience driven approach, (possibly) different templates may be created for different audience tracks. The characteristics, navigation and usability requirements of the audience classes are taken into account when designing the site templates.

Templates are specified using the primitive presentation concepts introduced in the previous sub section. Grids are used to position the multi media elements that make up the template. A template specifies the presentation elements that will appear at each page that will use this template. In addition, each template should contain at least one editable region. Editable regions denote the variable areas of the template. These are areas that should be filled during Page Design (see next sub section). An editable region can be placed anywhere in a grid (i.e. in any cell). Every area in the template that does not contain an editable region will be locked (i.e. cannot be changed) once the template is used during page design.

WSDM provides a set of pre-defined templates. These represent page lay-outs commonly found on the Web. These templates are:

- ***Header-Template***: a header template containing a header field
  - *Header-LeftSideBar-Template:* a header-leftsidebar template contains a header and a left sidebar. A specialization of this template is the *Header-LeftSideBar-Footer-Template,* which also contains a footer.
  - *Header-RightSideBar-Template:* a header-leftsidebar template contains a header and a left sidebar. A specialization of this template is the *Header-LeftSideBar-Footer-Template,* which also contains a footer.
  - *Header-Footer-Template:* a header-footer template contains, next to a header, also a footer

As an example of a template, consider the Header-LeftSideBar-Template from the conference review web site shown in figure 3.9. Here, the header and the left side bar both contain a multi media element (i.e. an image), and the main area below the header is an editable area, left to be filled when this template is used during page design (see next section). Note that indeed, this template can and has been modelled using only primitive presentation concepts (i.e. grids for positioning, multi media elements for content).

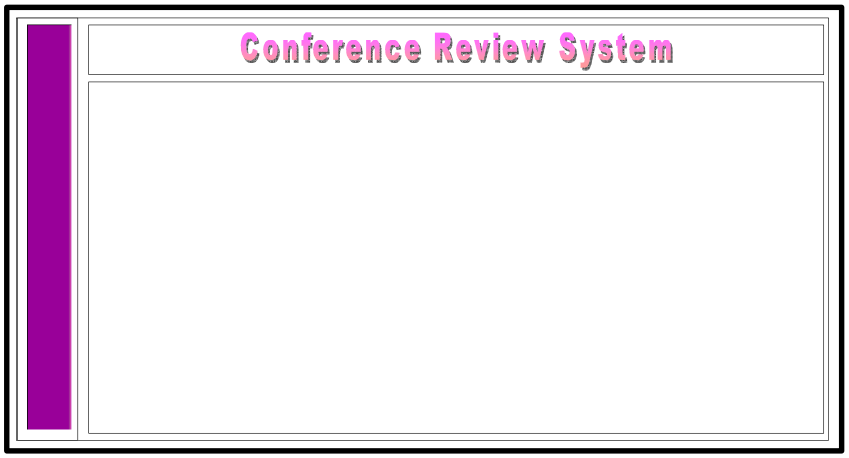


Figure 3-9 Example Header-LeftSideBar Template from the Conference Review Web site

To specify the style of a web site, WSDM currently relies on Cascading Stylesheets[38]. This allows style specification for one particular element (e.g. a grid, a specific cell of a grid) or a set of elements (e.g. all string multi media elements), and is expressible enough to describe most styles commonly found in web sites.

Page Design

The goal of the Page Design is to describe how the data and functionality, specified by means of object chunks created during the Information & Functional Modelling phase (see section 3.4.1.2) will be positioned and laid out on a page. For each page, the designer chooses a template, created during the Style & Template Design phase (see section 3.5.2.2, Style & Template Design), and specify how the lexical object types of the object chunks associated with the nodes grouped on the page will be positioned in the editable regions of that template. The characteristics, navigation and usability requirements of the audience classes are taken into account when designing the different pages.

Laying out the information modelled by means of the object chunks on pages is done using the primitive presentation modelling concepts described in section 3.5.2.1. For each object chunk connected to a node of a page, a positioning grid is constructed. Each Lexical Object Type of an object chunk is assigned to a particular cell of the positioning grid. If functionality denotations were used in the object chunk, interaction concepts can be used in the grid to represent them. If needed, multi media elements can be added to provide additional presentational information (e.g. labels, presentational graphics, etc).

For each link defined in the navigation model, the designer needs to specify the anchor. He does this by associating each link with a cell of a positioning grid. The element contained in that cell will consequently function as the link anchor. For each node on a page, each link defined in the navigation model that has as a source this node, must be associated with a cell of a positioning grid in the Page Design. Note how this versatile linking mechanism does not differentiate between the type of anchor (e.g. a text element, an image, a table, etc): a link is uniformly specified on a cell of a grid, no matter what is the content.

[38] http://www.w3.org/Style/CSS/ (access date 20 June 2005)

As for templates, WSDM has a set of pre-defined higher-level presentation concepts, which can be specified in terms of primitive presentation modelling concepts. These high-level presentation concepts correspond to well-known (web-based) GUI components and are more intuitive for a designer to use than the primitive presentation modelling concepts. We give a short overview of these higher level presentation concepts here:

- ***List:*** a list captures the widely used notion of a set of *elements*, separated by *separators* and bounded by a beginning and ending boundary. The elements in the list can be any other (high level or primitive) presentation concept. Lists can be oriented horizontally or vertically. WSDM provides two specialized lists that are commonly used in web pages: ordered bulleted lists (i.e. numbered lists, lettered lists, etc.) and non ordered bulleted lists.
- ***Table:*** a table corresponds to the classical notion of an html-table with rows and columns. Tables have the usual properties height and width, and may contain headings or footings, and possibly a caption.
- ***Menu:*** Menus provide the designer with a way to present the user with a set of navigation choices. Menus contain menu-items and are represented by lists. Due to the flexibility of lists, any kind of menu (i.e. bulleted list of navigation item, classical drop down menu, etc) can be represented.
- ***TableOfContent:*** tables of contents are commonly found at the top of a webpage, and often offer in-page navigation to different sections of a webpage. Similar as menus, tables of contents are represented by lists, and can thus materialize in any form.
- ***BreadCrumTrail:*** a breadcrumb trail captures the metaphor of keeping track where one comes from when arriving to a certain page. Breadcrum trails may be navigable.
- ***Section:*** typically, a webpage contains several *sections*. Next to the main content, a section provides support for a section title and for footnotes. Height and width can also be specified. One specialized section is a *summary*, often find on web pages to provide a short summary concerning another section or a page.
- ***Banner:*** banners typically appear on top, or on the left or right side of a page. A banner typically contains company information and/or logo, advertisements, a menu, etc. This information can be represented by any high-level or primitive presentation concept.
- ***Copyright:*** a copyright statement is mostly found on the bottom of a page, but in principle can appear anywhere. It contains the copyright symbol and a representation of the copy right holder (represented by an image or a string).
- ***Graphic:*** WSDM supports four particular kinds of graphics, namely *advertisements, figures, icons and logos.* All these graphics are represented by images, and thus height and width can be specified. A figure can furthermore have a caption.

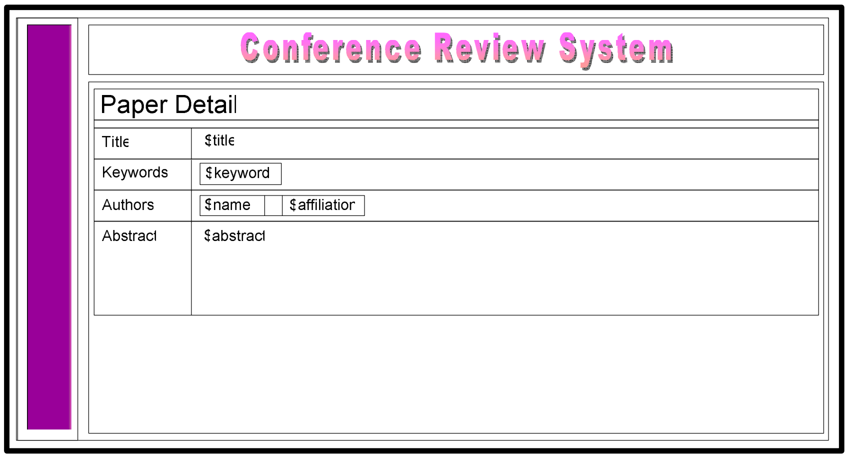


Figure 3-10 Example Page Presentation Design for the Paper Detail page of the Conference Review Web site

As an example of a Page Presentation Design for the conference review web site, consider the page for the PC Chair to view the details of a paper. This page consists of one single node, which has the "Paper Detail" object chunk connected (see figure 3.10). References to (lexical) object types of the object chunk are denoted using a $ before the name of the object type. Note that no high level presentation concepts are used in this example[39].

3.5.3 Data Source Mapping

The purpose of the data source mapping phase is to define the link between the object chunks, and the actual data. In this way, existing data sources (e.g. a relational database) can be re-used, by specifying where the actual data, conceptually represented in WSDM by object chunks, is located in the data source.

As WSDM uses ORM to model object chunks, the mapping rules described in [Halpin, 2001], to derive a relational database schema from an ORM model, can be used. Although other data source could be used as well (e.g. XML files, OWL instance files, text files, …), WSDM currently does not provide support for them.

3.6 Implementation Generation

The different design models created with WSDM: Object Chunks, the Navigation Model and the Implementation Design Models, can be used to generate and the actual implementation for a chosen platform, and in the chosen implementation language. This transformation can be performed fully automatically.

In a prototype, we have implemented this transformation as a pipeline. An overview of this transformation pipeline is shown in figure 3.11.

In the next sub sections, each of the transformations is described in detail.

[39] In the next chapter, only primitive presentation concepts will be formalized; hence we refrain from using high level concepts here. This furthermore shows that primitive presentation concepts alone are enough to model an arbitrary page presentation.

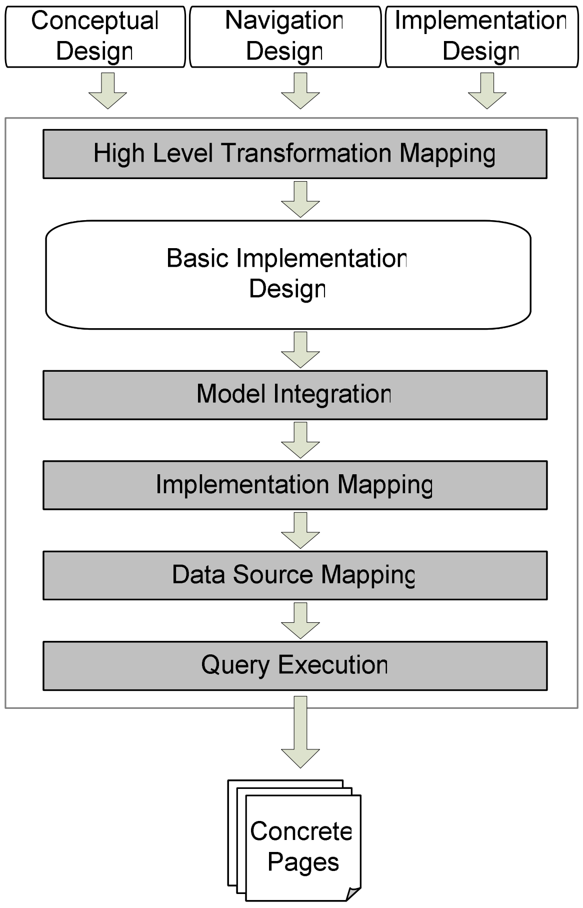


Figure 3-11 Implementation Generation Overview: Transformation Pipeline

3.6.1 High Level Transformation Mapping

The goal of the high level transformation mapping is to transform all high level presentation concepts to primitive presentation modelling concepts. More specifically, page and template design concepts are translated into primitive presentation modelling concepts (i.e. core positioning concepts, multi media elements and interaction elements). Thus, after the high level transformation mapping, the resulting model contains only primitive presentation modelling concepts: multimedia concepts (i.e. audio, video, integer, string, image, resource and video), positioning concepts (i.e. grids, rows and cells) and interaction concepts (i.e. select, input and action controls).

An important advantage of this first transformation is that all following transformation are significantly simplified, as they only need to be able to handle the primitive presentation modelling concepts, instead of all concepts.

3.6.2 Model Integration

The goal of the Model Integration Phase is to integrate the information & functional models (i.e. the object chunks), the navigation model, the page structure model, the template design model and the page design model into one (integrated) model. In principle, this step could be omitted, but as it simplifies to a large extend the following transformations, it is considered useful.

3.6.3 Implementation Mapping

During Implementation Mapping, the implementation platform is chosen, and the integrated model of the previous phase is partially transformed into code of the actual implementation platform. Any implementation platform can be chosen (e.g. html, xhtml, wml, etc.), provided that the platform is capable of expressing the minimal set of WSDM primitive presentation modelling concepts.

The implementation mapping transforms the non-variable parts (e.g. tables, hard-coded multi media elements, …) of the model into corresponding elements in the chosen implementation platform. Any reference to data (i.e. references to Lexical Object Type instances of object chunks) is not yet resolved, nor are the links specified in the navigation model.

The implementation transformation consists of two parts: the position transformation, responsible for transforming position concepts, and the multimedia transformation, responsible for transforming the multi media elements. The latter sub-transformation will be re-used in the Query Execution Transformation (see section 3.6.5, Query Execution), to represent the data retrieved from the data source.

3.6.4 Data Source Mapping

During the data source mapping, the references to Non-lexical Object Type instances are resolved and mapped to queries on the corresponding data source[40]. Depending on the format of the data source, a different the data source mapping algorithm will be needed. The aim of the data source mapping is to transform any reference to data, into (executable) queries using the querying formalism that is compatible with the implementation platform chosen. This transformation query can be performed fully automatically, provided that the mapping from the object chunks to the data source (e.g. a relational database) is available. For example, when the data is available in a database, scripts will be generated which specify the necessary queries.

Note that during data source mapping, the actual queries are not yet executed; they are only generated.

3.6.5 Query Execution

The last phase, the Query Execution Phase, executes the queries that were generated in the previous phase. Doing so, the web pages are populated with the actual data. During the query execution phase, the links to (actual) pages are resolved.

When the query execution phase is/can be performed *offline* a static site is generated. When the query execution is/needs to be done *at runtime*, dynamic pages are generated.

3.7 Extensions to WSDM

Next to the basic version of WSDM described in this chapter, some extensions to WSDM have been proposed in the past. A short overview is given in this subsection:

- ***Multiple Audience Class Hierarchies:*** WSDM supports multiple audience class hierarchies (e.g. an authorization hierarchy when authorization issues are involved, or a localization hierarchy, when localization issues are involved). By using different audience hierarchies for different aspects of the design, the designer can focus on one aspect at a time. A formal method to weave the different hierarchies into a single Audience Class Hierarchy is specified in [Casteleyn and De Troyer, 2001].

[40] For this purpose, the data source mapping (see section 3.5.3) is used.

- ***Semantic annotation:*** A crucial aspect in the realization of the Semantic Web is the availability of methods and tools to create, integrate and use semantic information in a, as much as possible, transparent and automatic way. As an answer to this problem, WSDM was extended so that semantic information can be automatically generated. By elevating the semantic annotation process to a conceptual level, a number of problems encountered in current approaches can be solved. Therefore, the annotation process is performed while designing the web site, and not after it is already completely implemented. This has as advantage that the creation of semantic information becomes less labour-intensive and the maintenance of the semantic information can be greatly improved. [De Troyer et al, 2003a ; De Troyer et al, 2003b]

- ***Localization:*** WSDM supports design for localized web sites. In this case, WSDM requires the specification of the different 'localities' (i.e. a particular place, situation or location). For each locality, where necessary, differences in the different models of the WSDM are indicated and specified. More details can be found in [De Troyer and Casteleyn, 2004].

- ***Accessibility:*** WSDM supports accessibility for visually impaired users. For this purpose, the Wafa Ontology is used: an ontology that defines concepts concerning how objects on a web page are presented (their structural properties) and used (the role they fulfil) to complete a successful web travel [Yesilada et al, 2004]. WSDM Presentation Design concepts are (formally) mapped on the concepts of the Wafa Ontology. Furthermore, WSDM describes the transformation process necessary to automatically generate these annotations in the actual implementation. Details can be found in [Plessers et al, 2005].

Chapter 4: WSDM formalization

"Everything should be made as simple as possible, but not simpler"
(Albert Einstein)

"I do not believe in mathematics"
(Albert Einstein)

The previous chapter explained WSDM in an informal way. A distinction between the models and techniques to derive these models was made. In this chapter the different models of WSDM are formally described. As formalism, set theory and first order predicate logic is used.

Specifying WSDM in a formal way (next to the informal description given in the previous chapter) provides the following benefits for this dissertation:

- **Unambiguous:** although the informal description of the previous chapter provides the reader with a good insight on WSDM, unavoidably, this textual description may introduce ambiguity and/or un-clarity. The formalization of the models will unambiguously specify WSDM.
- **Basis for Adaptation Model:** the formal models described in this chapter will be used as a basis to define the Adaptation Model, which will be the core of this PhD dissertation.
- **Implementation independent:** a formal description as given in this chapter provides platform- and implementation independence. Based on this formal description, any implementation can be (and has been) easily built.
- **Audience Driven Design:** as an important side effect, a formal characterization of the *audience driven* navigation model is given in this chapter. This was not previously done in the literature.

To increase readability, we will use the following conventions:

- Where needed, immediately after a definition, a short intuitive explanation is given. This should make it easier for the reader to follow the formalization.
- In definitions, names of sets are denoted **in bold.**
- The web site **W** and the Navigation Model **M** are denoted **in bold**
- Variables used in definitions are denoted *in italics*.

The following sets will be used frequently. Therefore, we define them here:

Let

$\mathbb{S}$	be the set of strings
$\mathbb{N}$	be the set of integers
W	be a web site
M	be the Navigation Model for the website **W**

As in the previous chapter, examples taken from the design of a conference review web site are used to illustrate the formal specification of WSDM. This case study was formulated in the context of the first IWWOST workshop (2001), and was solved using WSDM in [De Troyer and Casteleyn, 2001].

The remainder of this chapter is structured as follows. The phases of WSDM are described in the following sections: section 4.1 formally described the Mission Statement, section 4.2 the Audience Modelling, section 4.3 the Conceptual Design and finally section 4.4 the Implementation Design. For each phase and for each sub-phase, the different design models used in that phase are formally defined.

4.1 Mission statement

We start with defining the mission statement:

Definition [Mission Statement]:

The *Mission Statement* **M** for a web site **W** is a triple (purpose, subject, **targetAudiences**), where purpose $\in \mathbb{S}$, subject $\in \mathbb{S}$ and **targetAudiences** $\subset \mathbb{S}$.

End Definition [Mission Statement]

The Mission Statement is an informal description of the purpose and subject of the web site, and the targeted audience(s) for which the site will provide support. Purpose and subject are given as strings; TargetAudiences as a set of strings.

For the conference review web site, the formal mission statement looks as follows:

("*to support the overall selection process (submission by authors, evaluation and selection by the Program Committee)*", "the conference, *papers, authors and program committee*", {"*authors*", "*program committee members*", "*conference chair*"})

4.2 Audience Modelling

From the mission statement, the targeted audiences can be derived and elaborated. In the audience modelling phase, these targeted audiences are better analyzed, and classified. In the following definitions, we suppose the (targeted) visitors of a web site can be uniquely identified, and can be summed up. Their requirements and characteristics will be given as strings:

Suppose

T	is the set of targeted visitors of the web site **W**
R	$\subseteq \mathbb{S}$ is the set of all (information, functional, usability and business) requirements for **T**
C	$\subseteq \mathbb{S}$ be the set of all characteristics for **T**

Then we use the following notational conventions to refer to information, functional, usability and business requirements, and the characteristics of T:

$\mathbf{R}_{[i]}$	$\subseteq$ **R**, be the set of all information & functional requirement for **T**
$\mathbf{R}_{[u]}$	$\subseteq$ **R**, be the set of all usability requirements for **T**
$\mathbf{R}_{[b]}$	$\subseteq$ **R**, be the set of all business requirements for **T**

To refer to the requirements or characteristics of an arbitrary subset $\mathbf{X} \subseteq \mathbf{T}$, we respectively write $\mathbf{R}_\mathbf{X} \subseteq \mathbf{R}$ and $\mathbf{C}_\mathbf{X} \subseteq \mathbf{C}$. If **X** only contains one instance t (i.e. $\mathbf{X} = \{t\}$), we can use the shortcut notation $\mathbf{R}_\mathbf{t}$ instead of $\mathbf{R}_{\{\mathbf{t}\}}$. Both this notation, and the notation to refer to information, functional, usability and business requirements can be combined. I.e. to refer to the set of information and functional requirements of **X**, we write $\boldsymbol{R}_{X[i]} \subseteq \boldsymbol{R}_{[i]}$. Using these notations, an Audience Class is defined as follows:

Definition [Audience Class]:
An *Audience Class* **A** for a web site **W** is a subset of all targeted visitors **T** ($\mathbf{A} \subseteq \mathbf{T}$) of the web site **W**, with which a four-tuple ($\mathbf{R}_{A[i]}$, $\mathbf{R}_{A[u]}$, $\mathbf{R}_{A[b]}$, $\mathbf{C}_A$) is associated, where:

- $\forall\, a, b \in \mathbf{A}$: $\mathbf{R}_{a[i]} = \mathbf{R}_{b[i]} = \mathbf{R}_{A[i]}$

The four-tuple is called the audience class specification.

End definition [Audience Class]

The definition states that an audience class is a subset of the total set of visitors, for which all visitors in that subset have the same information and functional requirements.

As a shortcut notation, we can combine i, u and b between **[ ]**, denoting the union of the specified types of requirements. For example, $\mathbf{R}_{\mathbf{A[i,\,u]}} = \mathbf{R}_{\mathbf{A[i]}} \cup \mathbf{R}_{\mathbf{A[u]}}$. As an example from the conference review web site, consider the Audience Class "Author". Its audience class specification looks as follows:

({'*submit paper info and upload file*'**,** '*change submission until submission deadline*'**,** '*pre-register co-author*'**,** '*information about the authors submissions*'},
{},
{'*show announcements for other conferences the author might be interested in'*},
{'*able to communicate in English*', '*knowledgeable with the technical jargon of the conference*'})

Definition [Complete set of Audience Classes]
A *set of audience classes* {A₁, …, Aₘ} for a web site **W** *is complete* if and only if:

- $\bigcup_{1 \le j \le m} A_j = \mathbf{T}$

End definition [Complete set of Audience Classes]

Intuitively, the union of all Audience Classes of a complete set of Audience Classes is the full set of targeted visitors.

The following corollary follows immediately from the definition of complete set of Audience Classes:

Corollary [Each visitor belongs to at least one Audience Class]
Let
AC = { $A_1, \ldots, A_m$} be a complete set of Audience Classes.

Then:

- $\forall\, t \in \mathbf{T}\ \ \exists\, A_j \in \mathbf{AC}$ with audience class specification $(R_{Aj[i]}, R_{Aj[u]}, R_{Aj[b]}, C_{Aj})$ where $R_{t[i]} = R_{Aj[i]}$

End Corollary [Each visitor belongs to at least one Audience Class]

Intuitively, in a complete set of Audience Classes each individual visitor belongs to an Audience Class. In the remainder of this section, we will always work with a complete set of Audience Classes and thus, we can be sure that each targeted visitor belongs to an Audience Class.

Audience Classes may have a common set of requirements, which is a subset of their total set of requirements (i.e. each Audience Class possibly also has other requirements). To capture this, Audience Sub Classes are introduced:

Definition [Audience Sub Class]:
An Audience Class **B** with audience class specification $(R_{B[i]}, R_{B[u]}, R_{B[b]}, C_B)$ is an *Audience Subclass* of an Audience Class **A** with audience class specification $(R_{A[i]}, R_{A[u]}, R_{A[b]}, C_A)$, if and only if, $\mathbf{R}_{\mathbf{A[i]}} \subset \mathbf{R}_{\mathbf{B[i]}}$. Formally, we notate this: B < A.

End definition [Audience Sub Class]

Intuitively, and Audience Class is an Audience Subclass of another Audience Class if it has all the information and functional requirements of this last Audience Class and (some) additional requirements. Note from the definition that only information and functional requirements are taken into account in the Audience Sub Class relation. Business requirements are in fact requirements "forced" upon the user by the website owner (i.e. they are (unwillingly) assigned to the user); they are thus not considered discriminating from a user point of view. Characteristics are taken into account later in the design process, mainly for presentational issues, and are not considered when defining Audience Sub Class relations.

Terminology wise, if the Audience Class **B** is an Audience Subclass of the Audience Class **A**, we call **A** the Audience Superclass of **B**. From the point of view of their populations, the population of the Audience Subclass is a subset of the population of the Audience Superclass: each member of **B** is also a member of **A**. This is stated by the theorem that follows.

For the conference review web site, the 'PC Member' Audience Class has the same and some additional requirements compared to the 'Reviewer' Audience Class (e.g. a PC Member can introduce extra reviewers, which a Reviewer cannot do). Therefore, there exists an Audience Sub Class relation between the 'Reviewer' and the 'PC Member' Audience Classes, denoted as follows:

- 'PC Member' < 'Reviewer'

Theorem
Let **W** be a web site, with associated audience classes $\mathbf{AC} = \{A_1, \dots, A_m\}$ and $\mathbf{A_i} < \mathbf{A_j}$, with $1 \leq i, j \leq m \wedge i \neq j$, then the following proposition holds:

- $\mathbf{A_i} \subset \mathbf{A_j}$

End Theorem

Proof
From the definition of $\mathbf{A_i} < \mathbf{A_j}$, we know that $\mathbf{R_{Aj[i]}} \subset \mathbf{R_{Ai[i]}}$. Therefore, a member of $\mathbf{A_i}$ has the set of requirements $\mathbf{X} = \mathbf{R_{Ai[i]}} \setminus \mathbf{R_{Aj[i]}}$ extra, compared to $\mathbf{A_i}$. Members of $\mathbf{A_i}$ therefore have an extra condition they have to fulfil (i.e. they also need to have the requirements **X**, next to having the requirements $\mathbf{R_{Aj[i]}}$) to belong to $\mathbf{A_i}$ compared to $\mathbf{A_i}$ and consequently $\mathbf{A_i} \subset \mathbf{A_j}$.

End Proof

Using the Audience Subclass relationships between the Audience Classes, we can construct a hierarchy of Audience Classes. We call this hierarchy the Audience Class Hierarchy. The top of this hierarchy is always the Audience Class "Visitor". The Audience Class Visitor represents all potential users of the web site, including those that accidentally comes to the web site, and have no specific needs. The requirements associated with this class are usually general ones, such as a general statement about the web site, contact information, disclaimer information, …

Definition [Visitor Audience Class]:
Let

$\mathbf{T} = \{t_1, \dots, t_n\}$	be the set of targeted visitors of the web site **W**
$\mathbf{A} = \{A_1, \dots, A_n\}$	be the set of Audience Classes associated with a web site **W**.

We call $\mathbf{A_x}$ with audience class specification $(R_{T[i]}, R_{T[u]}, R_{T[b]}, C_T)$ $(1 \leq x \leq n)$ the *Visitor Audience Class* if and only if:

- $\mathbf{A_x} = \mathbf{T}$ and

- $\forall A_j \in \mathbf{A}$, with $1 \leq j \leq n \wedge j \neq x$: $A_j < \mathbf{A_x}$

End definition [Visitor Audience Class]:

The definition of the Visitor Audience Class states that all other audience classes are Audience Subclasses from Visitor. In other words, all other audience classes (and thus all targeted users) share the requirements of the Visitor audience class. These requirements (possibly an empty set) are thus the requirements all users of the web site have in common. In the Audience Class Hierarchy, the Visitor audience class is always the top-node.

Immediately from the definition of Audience Sub Class, and the definition of the Visitor Audience Class, we obtain the following corollary:

Corollary [Requirements of Visitor are shared for all targeted visitors]
Let **W** be a web site, with associated audience classes $\{A_1, \ldots, A_n\}$ and the Visitor Audience Class V with audience class specification $(R_{V[i]}, R_{V[u]}, R_{V[b]}, C_V)$, and a set of targeted visitors **T**, then:

- $\forall t \in \mathbf{T}$: $R_{V[i]} \subseteq R_{t[i]}$

End Corollary [Requirements of Visitor are shared for all targeted visitors]

As an example from the conference review web site, the visitor audience class has the following audience class specification:
({'view general information about conference'}, { }, { }, {'*able to communicate in English*'})

4.3 Conceptual Design

During the conceptual design phase, the requirements of the visitors are analyzed in detail, and their required information and functionality (task & information modelling) is modelled. Next, it is decided how the available data/functionality will be structured and how the (different) visitors will be able to navigate through the site (navigation design). In the next two sub sections, we will provide more details on these two sub phases.

4.3.1 Task and Information Modelling

During the Task and Information modelling phase, two tasks are performed:

- The designer elaborates, for each Audience Class, the requirements that were formulated during the audience modelling phase. For each requirement a task model is made by decomposing the task into elementary tasks. WSDM uses the Concurrent Task Tree technique to perform this task analysis. The task models obtained in this way are only an intermediate result in WSDM and therefore, no formalization of this technique is given in this chapter. The reader can consult the previous chapter for an informal description of the Concurrent Task Trees technique.

- For each elementary task, a tiny conceptual schema, called an Object Chunk, needs to be created that describes exactly which information/functionality is needed to support the elementary task. The data modelling language used to describe an Object Chunk in WSDM is a slightly modified form of ORM [Halpin, 2001].

We continue the formalization with a formal description of Object Chunks. First, we will formally describe the modified version of ORM that is used to model Object Chunks. This modified version of WSDM used predefined types, which conform to most primitive data types commonly found in web sites. We formalize these types first:

Definition [Primitive Type Names]
We call the set **PT =** {email, resource, image, audio, video, string, integer} the set of *Primitive Type Names*, or shortly Primitive Types.

End Definition [Primitive Type Names]

Next to the standard multimedia types email, image, audio, video, string and integer, there is also the versatile type resource, representing any referable resource. These Primitive Types are used to type the Lexical Object Types in ORM Schema's. We now define an ORM schema.

Definition [ORM Schema]
An *ORM Schema* S is a triple **(OT_S, $ROLE_S$, $STATEMENTS_S$)**, where

OT_S = $LABEL_S \cup NLOT_S$ is the finite set of Object Type, and
$LABEL_S$ $\subseteq \mathbb{S}$ x **PT,** is the finite set of Lexical Object Type (name)s with associated primitive type
$NLOT_S$ $\subseteq \mathbb{S}$**,** the finite set of Non Lexical Object Type (name)s

$ROLE_S$ $\subseteq \mathbb{S}$**,** is the finite set of role (name)s

$STATEMENT_S$ = $RELATION_S \cup$ SUBCLASS, where
$RELATION_S$ $\subseteq$ **$NLOT_S$** x (**$ROLE_S$** x **uniqueness** x **mandatory**) x (**$ROLE_S$** x **uniqueness** x **mandatory**) x **OT_S,** is the set of relations between Non Lexical Object Types and Object Types[41], with uniqueness $\in$ {0,1} and mandatory $\in$ {0,1} denoting the presence (1) or absence (0) of a uniqueness respectively mandatory constraint on each role of a relation, and

$SUBCLASS_S$ $\subseteq$ **$NLOT_S$** x **$NLOT_S$** the set of subclass relations between Non Lexical Object Types, where the first element is the parent and the second the child.

End definition [ORM Schema]

An Object Chunk in WSDM is an ORM Schema:

Definition [Object Chunk]
An *Object Chunk* **O** is an ORM Schema.

End definition [Object Chunk]

As an example of an Object Chunk, consider again the "Paper Detail" Object Chunk from the conference review web site (already informally described in Chapter 3, section 3.4.1.2, Information

[41] E.g. relations between two Non Lexical Object Types, and relations between a Non Lexical Object Type and a Label are allowed; relations between two Labels are not allowed.

and Functional Modelling). The formal specification of this Object Chunk is as follows (note that for reasons of readability we have named the different elements of the triple):

OT = {“Paper”, ”Author”, (“keyword”, string), (“abstract”, string), (“title”, string), (“name”, string), (“affiliation”, string)};
ROLE = {“has”, “is of”};
STATEMENT ={(“Paper”, (“has”, 0, 1), (“is of”, 0, 1), “keyword”),
(“Paper”, (“has”, 1, 1), (“is of”, 1, 1), “abstract”),
(“Paper”, (“has”, 1, 1), (“is of”, 1, 1), “title”),
(“Paper”, (“has”, 0, 1), (“is of”, 0, 1), “Author”)
(“Author”, (“has”, 1, 1), (“is of”, 1, 1), “name”),
(“Author”, (“has”, 1, 1), (“is of”, 0, 1), “affiliation”)}

In WSDM, for each information-, functional- and business requirement formulated, the designer needs to specify an Object Chunk describing the information and/or functionality that will fulfil this requirement. The formalization of this modelling activity is given next:

Definition [Complete Requirement Mapping]
Let

$\mathbf{R}_{[i]}$	be the set of all elementary information & functional requirement requirements for the targeted visitors **T** of a web site **W**
$\mathbf{R}_{[b]}$	be the set of all business requirement requirements for **T**
ORM	be the set of ORM Schema’s

A *Complete Requirement Mapping* is a relation $\mathbf{OC} \subseteq \mathbf{R}_{[i,b]} \times \mathbf{ORM}$ for which the following proposition holds:

- $\forall\, r \in \mathbf{R}_{[i,b]}$: $\exists!\, (r, c) \in$ **OC,** where c is the Object Chunk representing the requirement r.

End postulate [Complete Requirement Mapping]

Intuitively, a Complete Requirement Mapping is a relation, where each information-, functional- and business requirement is associated with an Object Chunk formally describing the information/functionality needed to fulfill the requirement. Note that the activity of modelling Object Chunks is a design activity; the relation **OC** is typically non-automatable and must be specified by the designer.

4.3.2 Navigation Design

The goal of the Navigation Design is to define the conceptual structure of the web site and to model how the members of the different audience classes will be able to navigate through the site. This information is captured in the Navigation Model:

Definition [Navigation Model]
Let

P	be a set of predicates.

A *Navigation Model* **M** for a web site **W** is a four-tuple **(N, H, C, L)** where

N	is a finite non-empty set of nodes
H	be a finite non-empty set of Object Chunks
C	$\subseteq$ **N** x **H**, the set of connections between nodes and Object Chunks

L is a 4-tuple ($\mathbf{L}_{[S]}$, $\mathbf{L}_{[R]}$, $\mathbf{L}_{[P]}$, $\mathbf{L}_{[A]}$) representing a set of links, where

- $\mathbf{L}_{[S]} \subseteq \mathbf{N} \times \mathbf{P} \times \mathbf{N}$, is the set of structural links between nodes
- $\mathbf{L}_{[R]} \subseteq \mathbf{H} \times \mathbf{P} \times \mathbf{H}$, is the set of semantic relationship links between Object Chunks
- $\mathbf{L}_{[P]} \subseteq \mathbf{N} \times \mathbf{P} \times \mathbf{N}$, is the set of process logic links between nodes
- $\mathbf{L}_{[A]} \subseteq \mathbf{N} \times \mathbf{P} \times \mathbf{N}$, is the set of navigation aid links between nodes

End definition [Navigation Model]

We use the same notational convention for links as we did for requirements in section 4.2: the type of link (**S**tructural, semantic **R**elationship, **P**rocess Logic and Navigation **A**id) is subscripted and noted between brackets **[ ]**. As shortcut notation, letters between **[ ]** can be combined to denoted the union of sets of links. For example, $\mathbf{L}_{[S, A]} = \mathbf{L}_{[S]} \cup \mathbf{L}_{[A]}$.

For a link $l = (n_1, p, n_2) \in \mathbf{L}_{[S,P,A]}$, we call n_1 the source of l (noted *source(*l*)*) and n_2 the target of l (noted *target(*l*)*). Similar, for a connection $c = (n_1, c_1) \in \mathbf{C}$ we call n_1 the source of c (noted *source(*c*))* and c_2 the target of c (noted *target(*c*)*). Note that each link has, next to a source and a target, also an associated predicate. The purpose of this predicate is to restrain the existence of the link according to type of user, time frame or output devices (e.g. "audienceClass(currentVisitor) = 'Author'").

As example, consider the basic Navigation Model for the Conference Review web site, as it was given in Chapter 3 figure 3.6. As this example concerns only a part of the complete Navigation Model (e.g. it only contains the basic audience track structure; no elaboration of audience tracks is given), the set H of Object Chunks and the set C of connections between nodes and Object Chunks are empty. The example also only contains structural links.

N ={'Visitor Track', 'Author Track', 'Reviewer Track', 'PCChair Track', 'PCMember Track'};
H ={};
C= {};
$\mathbf{L}_{[S]}$ = {('Visitor Track', '', 'Author Track'), ('Visitor Track', '', 'Reviewer Track'), ('Visitor Track', '', 'PCChair Track'), ('Reviewer Track', '', 'PCMember Track')};
$\mathbf{L}_{[R]}$ ={};
$\mathbf{L}_{[P]}$ ={};
$\mathbf{L}_{[A]}$ ={}

Before we are able to define the core characteristic of WSDM, the audience driven navigation approach, we need some auxiliary definitions:

Definition [path]
Let

M = (**N**, **H**, **C**, **L**)	be a Navigation Model for the web site **W**
n_1, n_2	$\in$ **N** be two nodes

A *path* p from n_1 to n_2 is a sequence of links $(l_1, ..., l_m)$ with $l_i \in \mathbf{L}_{[S,P,A]}$ and $1 \leq i \leq m$ where:

- source $(l_1) = n_1$
- target $(l_m) = n_2$
- $\forall\, k$ with $1 \leq k < m$: $\text{target}(l_k) = \text{source}(l_{k+1})$

End definition [path]

Intuitively, a path between two nodes consists of a sequence of links, where the source of the first link is the start of the path (i.e. the first node), the target of the last link is the end of the path (i.e. the second node), and the target of each link is the source of the following link,

Note that the definition of path only uses structural, process logic and navigation aid links (i.e. semantic links are not considered). As a semantic link constitutes a semantic relation between two Object Chunks, they do not directly define links between nodes. They are thus not considered in the definition of a path between nodes.

Notation wise, we use *nodes(path)* $\subseteq$ **N** to denote the set of nodes involved in a path. More precisely, $nodes((l_1, ..., l_n)) = (\bigcup_{1 \le i \le n} \text{source}(l_i)) \cup \text{target}(l_n)$

Paths can be defined using only one particular type of link. We define:

$\text{path}_{[S]}$ from n_1 to n_2 as a path $(l_1, ..., l_m)$ where $l_i \in L_{[S]}$ for $1 \le i \le m$
$\text{path}_{[P]}$ from n_1 to n_2 as a path $(l_1, ..., l_m)$ where $l_i \in L_{[P]}$ for $1 \le i \le m$
$\text{path}_{[A]}$ from n_1 to n_2 as a path $(l_1, ..., l_m)$ where $l_i \in L_{[A]}$ for $1 \le i \le m$

Definition [loop free path]
Let

M = (**N**, **H**, **C**, **L**)	be a Navigation Model for the web site **W**
n_1, n_2	$\in$ **N** be two nodes

We call a *path* $(l_1, ..., l_m)$ from n_1 to n_2 a *loop free path* if and only if:

- $\forall\, i$ with $1 \le i \le m$, $\forall\, j$ with $1 \le j \le m$, $i \neq j$: $\text{target}(l_i) \Rightarrow \text{target}(l_i) \neq \text{target}(l_j) \wedge \text{target}(l_i) \neq n_2$

End definition [loop free path]

Intuitively, a loop free path is a path in which there are no loops. Similar as for paths, we can define loop free $\text{path}_{[S]}$, $\text{path}_{[P]}$ and $\text{path}_{[A]}$ with respectively only structural links, process logic links and navigation aids link.

Using paths, we can now define when a certain node is *reachable* from another node:

Definition [reachable]
Let

M = (**N**, **H**, **C**, **L**)	be a Navigation Model for the web site **W**
n_1, n_2	$\in$ **N** be two nodes
h_1	$\in$ **H** be an Object Chunk

- We call a node n_2 *reachable* from the node n_1 if and only if:
  $\exists$ path p from n_1 to n_2
- We call a chunk c_1 *reachable* from a node n_1 if and only if:
  $\exists\, n_2 \in N$: $\exists$ path p from n_1 to $n_2 \wedge (n_2, h_1) \in$ **C**

End definition [reachable]

Similar as for paths, we define reachable$_{[S]}$, reachable$_{[P]}$ and reachable$_{[A]}$ as reachable, where respectively path$_{[S]}$, path$_{[P]}$ and path$_{[A]}$ are used in the definition instead of path.

Definition [root]
Let

M = (**N**, **H**, **C**, **L**)	be a Navigation Model for the web site **W**
r	$\in$ **N**

We call *r* the *root* of **W** if and only if:

$\forall n \in N \setminus \{r\}$: *n* is reachable$_{[S]}$ or reachable$_{[P]}$ from r

End definition [root]

Intuitively, the root is *the starting point* of the navigation (e.g. the *homepage*), and evidently, all other nodes are reachable from the root node. Note that in the definition of root, only structural and process logic links are considered (i.e. navigation aid links are not considered). As was explained in Chapter 3, section 3.4.2 (Navigation Design), navigation aid links are not in se necessary to navigate through the web site, they are only meant to ease navigation and to provide shortcuts to some main navigation hubs of the webpage (e.g. the homepage).

Definition [Transitive Closure of a node]
Let

M = (**N**, **H**, **C**, **L**)	be a Navigation Model for the web site **W**
n	$\in$ **N**

We call $n^+ \subseteq \mathbf{N}$ the *transitive closure* of a node $n \in \mathbf{N}$ if and only if:

$\forall x \in n^+$: *x* is reachable from n

$\neg\exists\, y \in (\mathbf{N} \setminus n^+)$: *y* is reachable from n

End definition [Transitive Closure of a node]

Intuitively, the transitive closure of a node is the set of all nodes that can be reached starting from that particular node.

Corollary
Let

M = (**N**, **H**, **C**, **L**)	be a Navigation Model for the web site **W**
r	$\in$ **N** be the root of **W**

- $r^+ = \mathbf{N}$

End Corollary

This corollary follows immediately from the definition of root and the transitive closure.

Definition [Transitive Closure of a path]
Let

M = (**N**, **H**, **C**, **L**)	be a Navigation Model for the web site **W**
p	= $(l_1, ..., l_n)$ be a path

We define the transitive closure $p^+ \subseteq \mathbf{N}$ of the path p as:

$$\left(\bigcup_{1 \le i \le n} (\text{source}(l_i))^+ \right) \cup \text{target}(l_n)^+$$

End definition [Transitive Closure of a path]

Intuitively, the transitive closure of a path is the combined transitive closure of *nodes(p),* i.e. all nodes of the path.

Similar as for the definition of reachable, we can define $n^{+[S]}$, $n^{+[P]}$, $n^{+[A]}$ and $p^{+[S]}$, $p^{+[P]}$, $p^{+[A]}$ where respectively $\text{reachable}_{[S]}$, $\text{reachable}_{[P]}$ and $\text{reachable}_{[A]}$ is used in the definition of Transitive Closure.

Definition [Navigation sub model]

Let

M = (**N**, **H**, **C**, **L**)	be a Navigation Model for the web site **W**
P	be a set of predicates
P’	$\subseteq$ **P**

The Navigation Model **M’** = (**N’**, **H’**, **C’**, **L’**) is a *(Navigational) sub model* of **M** if and only if:

- $\mathbf{N'} \subseteq \mathbf{N} \wedge \mathbf{N'} \neq \emptyset$
- $\mathbf{H'} \subseteq \mathbf{H} \wedge \mathbf{H'} \neq \emptyset$
- $\mathbf{C'} \subseteq \mathbf{C}$
- L’ is a 4-tuple ($\mathbf{L'}_{[S]}$, $\mathbf{L'}_{[R]}$, $\mathbf{L'}_{[P]}$, $\mathbf{L'}_{[A]}$) representing a set of links, where
    - $\mathbf{L'}_{[S]} \subseteq \mathbf{N'} \times \mathbf{P'} \times \mathbf{N'}$ and $\mathbf{L'}_{[S]} \subseteq \mathbf{L}_{[S]}$ is the set of structural links between nodes
    - $\mathbf{L'}_{[R]} \subseteq (\mathbf{H'} \times \mathbf{P'} \times \mathbf{H'}) \cap \mathbf{L}_{[R]}$, is the set of semantic relationship links between Chunks
    - $\mathbf{L'}_{[P]} \subseteq \mathbf{N'} \times \mathbf{P'} \times \mathbf{N'} \cap \mathbf{L}_{[P]}$, is the set of process logic links between nodes
    - $\mathbf{L'}_{[A]} \subseteq \mathbf{N'} \times \mathbf{P'} \times \mathbf{N'} \cap \mathbf{L}_{[A]}$, is the set of navigation aid links between nodes

We use the notation **M’ < M.**

End definition [Navigation sub model]

Intuitively, a navigation sub model can be seen as part of a navigation model where only some nodes, the links among them and the chunks connected to them, are considered.

Definition [Navigation Track]

Let

M = (**N**, **H**, **C**, **L**)	be a Navigation Model for the web site **W**
n	$\in$ **N**

A *Navigation Track* **M’** = (**N’**, **H’**, **C’**, **L’**) with root n is a sub model of **M** where:

- $\mathbf{N'} = n^{+(S)} \cup n^{+(P)}$
- $\mathbf{H'} = \{h \in \mathbf{H} \mid \exists\, c \in \mathbf{C}: \text{source}(c) \in \mathbf{N'}\}$
- $\mathbf{C'} = \{c \in \mathbf{C} \mid \text{source}(c) \in \mathbf{N'}\}$
- L’ is a 4-tuple ($\mathbf{L'}_{[S]}$, $\mathbf{L'}_{[R]}$, $\mathbf{L'}_{[P]}$, $\mathbf{L'}_{[A]}$) representing a set of links, where
    - $\mathbf{L'}_{S} = \{l \in \mathbf{L}_{[S]} \mid \text{source}(l) \in \mathbf{N'} \wedge \text{target}(l) \in \mathbf{N'}\}$
    - $\mathbf{L'}_{[R]} = \{l \in \mathbf{L}_{[R]} \mid \text{source}(l) \in \mathbf{H'} \wedge \text{target}(l) \in \mathbf{H'}\}$

- $\mathbf{L'}_{[P]} = \{l \in \mathbf{L}_{[P]} \mid \text{source}(l) \in \mathbf{N'} \wedge \text{target}(l) \in \mathbf{N'}\}$
- $\mathbf{L'}_{[A]} = \{l \in \mathbf{L}_{[A]} \mid \text{source}(l) \in \mathbf{N'} \wedge \text{target}(l) \in \mathbf{N'}\}$

End definition [Navigation Track]

Intuitively, a Navigation Track can be seen as all possible paths a user can follow starting from a particular node (i.e. the root), and thus all information that is reachable from that node.

Definition [Audience Track]
Let

$\mathbf{M} = (\mathbf{N}, \mathbf{H}, \mathbf{C}, \mathbf{L})$	be a Navigation Model for the web site **W**
T	be the set of targeted visitors for the web site **W**
A	$\subseteq \mathbf{T}$ an audience class with associated audience class specification $(\mathbf{R}_{A[i]}, \mathbf{R}_{A[u]}, \mathbf{R}_{A[b]}, \mathbf{C}_A)$
n	$\in \mathbf{N}$
OC	be the mapping of elementary requirements on Object Chunks

An *Audience Track* **AT** for the Audience Class **A** is a Navigation Track $\mathbf{M'} = (\mathbf{N'}, \mathbf{H'}, \mathbf{C'}, \mathbf{L'})$ originating from a node $n \in \mathbf{N}$ (i.e. the root) for which:

- $\forall\, r \in \mathbf{R}_{A[i,b]}$: OC(r) is reachable from n
- $\neg(\exists\, r \in \bigcup_{Ai \in \mathbf{A} \wedge Ai \neq A \wedge \neg(Ai < A)} (R_{(Ai)[i,b]}) \setminus \mathbf{R}_{A[i,b]} \wedge \neg(\mathbf{A_i} < \mathbf{A})$: OC($r$) is reachable from n)

End definition [Audience Track]

We note $\mathbf{AT}_A$ to denote the Navigation Model **M'** associated with audience class A.

Intuitively, an audience track is the navigation track specifically designed for a certain audience class. Along this navigation path, the members of this audience class find all the information and functionality that is relevant for them[42], and no other information besides the information and functionality of possible Audience Sub Classes.

Definition [Audience Based Navigation Design]
Let

$\mathbf{M} = (\mathbf{N}, \mathbf{H}, \mathbf{C}, \mathbf{L})$	be a Navigation Model for the web site **W**
$\mathbf{A} = \{A_1, \ldots, A_n\}$	be the set of Audience Classes associated with the web site **W**

We call **M** *audience based* if and only if:

- $\forall\, A_i \in \mathbf{A} : \exists$ Audience Track $\mathbf{AT}_{Ai} = (\mathbf{N}_{Ai}, \mathbf{H}_{Ai}, \mathbf{C}_{Ai}, \mathbf{L}_{Ai})$for A_i
- $\forall\, i$ with $1 \le i \le n \wedge \forall\, j$ with $1 \le j \le n \wedge i \neq j$:
  - $\text{root}(A_i) \neq \text{root}(A_j)$
  - if $\neg(\mathbf{AT}_{Ai} < \mathbf{AT}_{Aj})$
    $\mathbf{N}_{Ai} \cap \mathbf{N}_{Aj} = \varnothing$

    if $(\mathbf{A_i} < \mathbf{A_j})$
    $\mathbf{AT}_{Ai} < \mathbf{AT}_{Aj}$

[42] To be more correct, the information and functionality the designer assessed for this audience class. (i.e. the object chunks resulting from the requirements assessed for this audience class are reachable)

End definition [Audience Based Navigation Design]

An audience based navigation design is a way of organizing the structure of the web site, so that each audience class identified for the site, has its own audience track. Tracks of Audience Subclasses are nested (e.g. are Audience Submodels), and all distinct audience classes have audience tracks that do not overlap (i.e. all the nodes are disjoint). Note that they can contain the same information if they have the same chunk(s) connected to some of their nodes, but the nodes themselves can never be shared (e.g. navigation within a track cannot cause a user to unwillingly 'jump' to another track).

As an example of an Audience Track, consider the elaborated Author Audience Track from the Conference Review which was already schematically described in figure 3.7, Chapter 3, section 3.4.2 (Navigation Design).

N = {'AuthorTrack', 'AuthorSubmissionNode', 'SubmissionInfoNode', 'UpdatePaperInfoandFile', 'RegisterNewPaperNode', 'AddCo-authorNode', 'SubmitPaperInfoandFile', 'Update-deleteCo-authorsNode', 'Update-AddCo-authorNode'};

H = {'AuthorsSubmissions', 'SubmissionInfo', 'UpdateSubmission', 'RegisterNewPaper', 'AddCo-author', 'MakeSubmission', 'DeleteCo-authors'};

C = {('RegisterNewPaperNode', 'RegisterNewPaper'), ('AddCo-authorNode', 'AddCo-author'), ('Update-AddCo-authorNode', 'AddCo-author'), ('SubmitPaperInfoandFile', 'MakeSubmission'), ('Update-deleteCo-authorsNode', 'DeleteCo-authors'), ('AuthorSubmissionNode', 'AuthorsSubmissions'), ('SubmissionInfoNode', 'SubmissionInfo'), ('UpdatePaperInfoandFile', 'UpdateSubmission')};

$\mathbf{L}_{[S]}$ = {('AuthorTrack', '', 'AuthorSubmissionNode'), ('AuthorTrack', '', 'RegisterNewPaperNode'), ('RegisterNewPaperNode', '', 'AddCo-authorNode'), ('RegisterNewPaperNode', '', 'SubmitPaperInfoandFile'), ('AddCo-authorNode', '', 'AddCo-authorNode'), ('SubmitPaperInfoandFile', '', 'AuthorSubmissionNode'), ('AuthorSubmissionNode', '', 'SubmissionInfoNode'), ('SubmissionInfoNode', '', 'UpdatePaperInfoandFile'), ('UpdatePaperInfoandFile', '', 'SubmissionInfoNode'), ('UpdatePaperInfoandFile', '', 'Update-deleteCo-authorsNode'), ('Update-deleteCo-authorsNode' , '', 'UpdatePaperInfoandFile'), ('UpdatePaperInfoandFile', '', 'Update-deleteCo-authorsNode'), ('Update-deleteCo-authorsNode', '', 'UpdatePaperInfoandFile')};
$\mathbf{L}_{[R]}$ ={};
$\mathbf{L}_{[P]}$ ={};
$\mathbf{L}_{[A]}$ ={}

The root for this audience track is the 'AuthorTrack' node: each node is reachable from this node. An example of a path, connecting the nodes 'AuthorTrack' and 'UpdatePaperInfoandFile' in this Navigation Model is:

(('AuthorTrack', '', 'AuthorSubmissionNode'), ('AuthorSubmissionNode', '', 'SubmissionInfoNode'), ('SubmissionInfoNode', '', 'UpdatePaperInfoandFile'))

4.4 Implementation Design

4.4.1 Site Structure Design

The first step of the implementation design is to decide which logical units of information, represented by nodes created during the navigation design, will be grouped together in (abstract) pages.

We thus define an abstract page as follows:

Definition [Abstract Page]
Let
$\mathbf{M} = (\mathbf{N}, \mathbf{H}, \mathbf{C}, \mathbf{L})$ be the Navigation Model for a web site **W**

A page $\mathbf{P} \subseteq \mathbf{N}$ from the web site **W** is a set of nodes from N.

End definition [Abstract Page]

Intuitively, an Abstract Page is simply a collection of nodes. It is the designer who decides which nodes to group in an abstract page.

Note that, although chunks are not explicitly specified in the page definition, the Navigation Model defines connections between chunks and nodes, and thus pages implicitly have zero or more chunks through their nodes.

Definition [Site Structure Model]
Let
$\mathbf{M} = (\mathbf{N}, \mathbf{H}, \mathbf{C}, \mathbf{L})$ be the Navigation Model for a web site **W**

A *Site Structure Model* **SSM** is a set of pages $\{P_1, \dots P_m\}$ with $m \geq 1$, where each P_i is an Abstract Page for the web site W and

- $\forall\, n \in P_i,\ 1 \leq i, j \leq m: \neg\exists\, P_j \wedge n \in P_j \wedge i \neq j$

End definition [Site Structure Model]

In other words, the Site Structure Model is a partition of the set N of nodes.

Intuitively, a Site Structure Model is a set of pages, grouping the nodes of the web site. Furthermore, if a node is present on a page, it cannot be present on another page (e.g. a node belongs to at most one page in a Site Structure Model). Note however that the same object chunk may be connected to several nodes and therefore the same information or functionality may appear one more then one page. The distinction between nodes and chunks avoids redundancy (i.e. an object chunk needs to be modelled only once).

Although the definition of Site Structure Model does not require this, in general, for each node in the Navigation Model, there will be a page in the Site Structure Model (i.e. each node is represented on the actual webpage).

As example from the conference review web site, consider the Site Structure Model for the Author Audience Track which was already schematically presented in figure 3.8, Chapter 3, section 3.5.1 (Site Structure Design).

{{'Author Track'}, {'Author SubmissionNode'}, {'SubmissionInfoNode'}, {'UpdatePaperInfo andFile', 'Update-DeleteCo-authorsNode', 'Update-AddCo-authorNode'}, {'RegisterNewPaperNode'}, {'AddCo-authorNode'}, {'SubmitPaperInfoandFile'}}

The example Site Structure Model is furthermore *complete*, as defined in the following definition:

Definition [Complete Site Structure Model]
Let

M = (**N**, **H**, **C**, **L**)	be the Navigation Model for the web site **W**
SSM	be a Site Structure Model for the web site **W**

We call **SSM** *complete* if and only if:

- $\forall n \in \mathbf{N}: \exists \mathbf{P} \in \mathbf{SMM} \wedge n \in \mathbf{P}$

End Definition [Complete Site Structure Model]

Intuitively, a Site Structure Model is complete if all nodes of the web site are represented on some page.

In mathematics, a complete site structure model is called *a partition* of the set of nodes **N**.

4.4.2 Presentation design

Where the Site Structure Design was concerned with the grouping of nodes onto (abstract) pages, the Presentation Design deals with the layout (positioning, presentation) of the different pages of the web site.

As explained in Chapter 3, section 3.5.2 (Presentation Design), WSDM has a layered set of modelling concepts to define structure and layout. Primitive Presentation Concepts are the basic building blocks used on a (web) page. We start by defining these basic building blocks in the next subsection.

4.4.2.1 Primitive Presentation Concepts

Primitive Presentation Concepts in WSDM are the basic building blocks used to specify the structure and layout of a webpage. They can be classified into two categories: **multi media elements** (the basic data types that are supported) and **positioning concepts** (used to position the different multi media elements on a page).

Multi Media Elements

We start with the definitions of the first category, the multi media elements, representing data of type String, Integer, Email, Resource, Audio, Video or Image Element.

Note that the types of the multi media elements correspond with the (primitive) types(s) of the Lexical Object Types (i.e. Labels) of Object Chunks. While multi media elements are actual instances of data types, the type of a Lexical Object Type only indicates the type of data it may contain. In the actual implementation, data originating from the Lexical Object Types is represented as multi media elements.

Definition [Audio Element]

An *audio element* ae is a tuple (Description, Source), where Description $\in \mathbb{S}$ and Source $\in \mathbb{S}$

End Definition [Audio Element]

Intuitively, an audio element consists of a textual description of the audio (e.g. “Rainbow - Cranes”), and a source (the location of the actual audio).

Definition [Email Element]

An *email element* em is a tuple (Description, Address), where Description $\in \mathbb{S}$ and Address $\in \mathbb{S}$

End Definition [Email Element]

Intuitively, an email element consists of a textual description of the email address (e.g. “Email address of Sven Casteleyn”), and an Address (e.g. “Sven.Casteleyn@vub.ac.be”)

Definition [Image Element]
Let
Dim be $\mathbb{N} \times \mathbb{N}$

An *image element* im is a triple (Description, Source, Dimension), where Description $\in \mathbb{S}$, Source $\in \mathbb{S}$ and Dimensions $\in$ **Dim**

End Definition [Image Element]

Intuitively, an Image element consists of a textual description of the image (e.g. “left banner”), a source (e.g. the location of the image) and the size of the Image (e.g. “30 x 200).

Definition [Resource Element]

A *resource element* re is a tuple (Description, Source), where Description $\in \mathbb{S}$ and Source $\in \mathbb{S}$

End Definition [Resource Element]

Intuitively, a resource element consists of a textual description of the resource (e.g. “Windows Service Pack 2.0”) and a source (the location of the actual resource).

Definition [Video Element]

A *video element* vi is a tuple (Description, Source), where Description $\in \mathbb{S}$ and Source $\in \mathbb{S}$

End Definition [Video Element]

Intuitively, a video element consists of a textual description of the video (e.g. “BBC News 01-07-2005”), and a source (the location of the actual video).

A Multi Media Item is now defined as one of these basic elements defined above, or a standard string or integer:

Definition [Multi Media Element]

A *Multi Media Element* is either a Video, Audio, Image, Email, Resource Element, a string or an integer.

End Definition [Multi Media Element]

Basic Positioning Concepts

The basic positioning mechanism used in WSDM to spatially order data on an Abstract Page, are grids. A grid can be considered a matrix, where each cell of the matrix contains some (web) content[43] and for which its spatial dimensions (e.g. height and width) can be specified. Grids can be nested. By nesting grids, the designer can gradually specify the positioning for parts of an Abstract Pages in more detail. Given the arbitrarily nesting of grids, and the possibility to specify height and width of the various cells of a grid, the designer wields a versatile position mechanism that is extremely flexible and allows specifying arbitrary layouts for web pages and web sites.

A grid is formally defined as a tuple of matrices (of equal dimension), where the first matrix of the tuple specifies the actual content of the grid, and the second matrix specifies the size of each cell of the grid (e.g. the actual width and height of that cell). Before further explaining grids, we first give a formal definition of a matrix as used in the remainder of this section:

Definition [Matrix]
A *matrix* M is an ordered tuple (row_1, …, row_m) of rows, where each row_i with $1 \leq i \leq m$ is an ordered tuple (el_1, …, el_n) of arbitrary elements.

End Definition [Matrix]

The matrix M has m rows and n columns; we say the matrix M has dimensions m x n. The elements (of the rows) of the matrix are called the *content* of the matrix. To access the different rows and elements, we define the functions:

- row(M, i) with $1 \leq i \leq m$ to return the *i*th row of the matrix M
- element(r, *i*) $1 \leq i \leq n$ to return the *i*th element of the row r of the matrix M
- element(M, x, y) with $1 \leq x \leq m$ and with $1 \leq y \leq n$ to return the xth element of the yth row of matrix M

This definition of matrix corresponds with the usual definition of a matrix in mathematics. Notice that, at this stage, we did not yet formally specified the content of the matrix. This will be done when defining the different types of grids.

As was explained, a Grid is the basic positioning mechanism in WSDM. It consists of a tuple of matrices, one specifying the content and the other specifying the size of each cell of the grid. This is defined in the next definition:

Definition [Grid]
Let
M_1, M_2 be matrices with dimensions m x n

A *Grid* G is a tuple (M_1, M_2) where:

- element(M_2, x, y) $\in$ ($\mathbb{N}$ x $\mathbb{N}$), with $1 \leq x \leq m$ and $1 \leq y \leq n$

[43] We will formally specify which web content is allowed in a grid further in this section.

End Definition [Grid]

Note that in this definition, the content of the first matrix is (still) deliberately left open; it will be defined when defining particular kinds of Grids (see further on).

For a Grid G = (M_1, M_2), we define the following accessor function:

- content(G, x, y) = element(M_1, x, y)

Intuitively, a grid is a tuple consisting of a *content* matrix (the first element of the tuple) coupled with a *spatial dimensions* matrix that specifies the width and height of each content element (the second element of the tuple).

The dimensions of the Grid are equal to the dimensions of the content and spatial dimensions matrices, which need to be equal.

To support a uniform look and feel for a web site, WSDM supports the notion of *templates.* A template specifies the positioning of the common parts of a set of Abstract Pages of a web site. Templates are specified using grids, where the common parts of the template are specified using multi media elements, and the generic part is denoted using *editable regions*. When the presentation design of an actual page is given these editable regions will be filled in using Presentation Grids[44]. The definition of a Template Grid follows:

Definition [Template Grid]

Let

G	be a grid of dimension m x n
MME	be a set of Multi Media Elements

A *Template Grid* G is a Grid where:

- $\forall$ content(G, *x, y)* with $1 \leq x \leq$ m and $1 \leq y \leq$ n**:** content(G, *x, y)* $\in$ **MME** $\vee$ content(G, *x, y)* = $\varnothing$ $\vee$ content(G, *x, y)* $\in$ {**'**editableRegion'} **x** $\mathbb{N}$ $\vee$ content(G, *x, y)* = TG' where TG' is a Template Grid

End Definition [Template Grid]

Intuitively, a Template Grid is a grid where each content cell of the Grid is either empty, or a Multi Media Element, an editable region or another (nested) Template Grid. Nesting grids allows gradually specifying the template in more detail (i.e. each nested grid specifies in detail the cell in which it is contained). Empty cells can be used for adding white space. The Multi Media Elements are used to specify the fixed parts of the template. These parts will appear on every Abstract Page based on the template. Editable regions, identified by a marker, denote the areas that each Abstract Page created on basis of the template can use for its specific information. A template can contain several editable regions. Therefore, each editable region is numbered (e.g. the third editable region is denoted (editableRegion, 3)).

As an example of a template grid, consider the template grid given for the conference review web site in Chapter 3, section 3.5.2 (Style & Template Design). The formal definition of this template grid is given as follows[45]:

[44] Definition follows.

[45] To improve readability, the presentation grid is given in two steps.

matrix1 = (((“Left banner”, “//images/leftbanner.jpg”, (100,800)), matrix2))
dimMatrix1 = ((100, 800), (1100, 800))

matrix2 = (((“Top banner”, “//images/topbanner.jpg”, (1100,100))), ((‘editableRegion’, 1)))
dimMatrix2 = ((1100, 100), (1100, 700))

ConferenceTemplateGrid = (matrix1, dimMatrix1)

Note how image multi media elements are used for both left and top banner; matrices are used to position elements.

Template Grids are thus used to specify the position of the common part of a group of Abstract Pages. The position of the parts specific for each Abstract Page is done using Presentation Grids. As Abstract Pages in WSDM consist of nodes, which have connected Object Chunks, Presentation Grids need to be able to position (the information/functionality in) these Object Chunks relatively to each other, and to position the information/functionality specified in these Object Chunks (i.e. the different Lexical Object Types of the Object Chunk). We first give the general definition of a Presentation Grid, followed by the definition of a Presentation Grid used to present an Object Chunk:

Definition [Presentation Grid]
Let

G	be a grid of dimension m x n
LABELS	be a set of Lexical Object Types
MME	be a set of Multi Media Elements

A *Presentation Grid* G is a Grid where:

- $\forall$ content(*G, x, y)* with $1 \leq x \leq$ m and $1 \leq y \leq$ n**:** content(*G, x, y)* $\in$ **LABELS** $\vee$ content(*G, x, y)* = $\emptyset$ $\vee$ content(*G, x, y)* $\in$ **MME** $\vee$ content(*G, x, y)* = G’ where G’ is a Presentation Grid

End Definition [Presentation Grid]

Intuitively, a Presentation Grid is a Grid where each content cell of the Grid is either empty, a Multi Media Element, a Lexical Object Type or another (nested) Presentation Grid. Similarly as for Presentation Grids, empty cells can be used for introducing white space. Presentation Grids can be arbitrarily nested to allow gradual specification of detail. Multi Media Elements can be used at the discretion of the designer to add primitive content. Lexical Object Types are placed in cells to mark that, during the implementation phase, particular data need to be retrieve from the data source (represented in the Object Chunk).

Presentation Grids are used to specify the internal positioning of Lexical Object Types of Object Chunks. We define a Presentation Grid providing a presentation for an Object Chunk, as follows:

Definition [Presentation Grid providing a presentation for an Object Chunk]
Let

CH	be an Object Chunk
G	be a Presentation Grid

A Presentation Grid G *provides a presentation for* the Object Chunk CH, denoted as G->CH, if and only if:

- $\forall$ *label* $\in$ $\mathbf{Labels_{CH}}$: $\exists$ content(*G, x, y) = label*

End Definition [Presentation Grid providing a presentation for an Object Chunk]
Intuitively, a Presentation Grid provides a presentation for an Object Chunk if each Lexical Object Type of that Object Chunk is at least once the content of a cell of the Grid, thereby specifying the positioning of the Lexical Object Types of the Object Chunk.

As an example of a Presentation Grid providing a presentation for an Object Chunk, taken from the conference review web site, consider the presentation grid representing the "Paper Detail" Object Chunk of Chapter 3, section 3.5.2 (Page Design) figure 3.5.

PaperDetailPresentationMatrix = (("Paper Detail:"), ("Title: ", $title), ("Keywords: ", *PaperDetailKeyWordGrid*), ("Authors: ",*PaperDetailAuthorsGrid*), ("Abstract: ", $abstract))
PaperDetailPresentationDimensionMatrix = ((300, 40), ((120, 35), (580, 35)), ((120, 35), (580, 35)), ((120, 35), (580, 35)), ((120, 35), (580, 35)))

PaperDetailKeyWordMatrix = (($keyword))
PaperDetailKeyWordDimensionMatrix = ((580, 35))

PaperDetailAuthorsMatrix = (($name, ",", $affiliation))
PaperDetailAuthorsDimensionMatrix = (((120, 35), (7, 35), (120, 35)))

PaperDetailPresentationGrid =
(*PaperDetailPresentationMatrix*, *PaperDetailPresentationDimensionMatrix*)

This example Presentation Grid will be used in the next section to illustrate a Page Presentation Design.

4.4.2.2 Page and Web site Presentation Design

A Page Presentation Design for a page makes use of Template and Presentation Grids to position the nodes and the content provided by the Object Chunks on an Abstract Page. Each Abstract Page, is based on one single Template Grid. Subsequently, a set of Presentation Grids is used to describe the editable regions of the template. The definition of a Page Presentation Design is as follows:

Definition [Page Presentation Design]
Let

W	be a web site
M = (**N**, **H**, **C**, **L**)	be the Navigation Model for the web site **W**
P	$\subseteq$ **N** be an abstract page from the web site **W**
TemplateGrids	a set of Template Grids
PresentationGrids	a set of Presentation Grids

A *Page Presentation Design* **PPD** for a page P is a triple (**P**, templateGrid, **ChunkGrids**) where

- **P** is a Page
- templateGrid $\in$ **TemplateGrids**
- **ChunkGrids** $\subseteq \mathbb{N}$ x **PresentationGrids**, the set of tuples (*editableRegion, PresentationGrid*) where *editableRegion* denotes the editable region in which *PresentationGrid* is located
- $\forall$ *(node, chunk)* $\in$ **C:** $\exists$ (*i*, PresentationGrid) $\subseteq$ **ChunkGrids** $\wedge$ *PresentationGrid->chunk*

End Definition [Page Presentation Design]

Intuitively, a Page Presentation Design for a page consists of a Template Grid, upon which the page presentation is build, and a number of Presentation Grids providing presentation for the Object Chunks of the nodes of the page. Notice that the index *i* in each tuple (*i*, *PresentationGrid*) of **ChunkGrids** refers to the ith *editableRegion* of the Template Grid *templateGrid*, thus specifying *in which* editable region the Presentation Grid is placed.

As an example taken from the conference review web site, consider the Page Presentation Design for the page “Paper Detail Page”, containing the node “Paper Detail Node” with the chunk “Paper Detail” (see Chapter 3, section 3.5.2.3). The Page Presentation Design for this single page is as follows:

({Papter Detail Node}, *ConferenceTemplateGrid*, {(1, *PaperDetailPresentationGrid*)})

Finally, we are ready to define the Web site Presentation Model, which describes the presentation for all Abstract Pages of the web site.

Definition [Web site Presentation Model]
Let

W	be a web site
M = (**N**, **H**, **C**, **L**)	be the Navigation Model for the web site **W**
SSM = {$P_1, \ldots, P_n$**}**	be a (complete) Site Structure Model for the web site **W**

The Web site Presentation Model **WPM** = {$PPD_1, \ldots, PPD_n$} is a set of Page Presentation Designs where:

- $\forall$ **P** $\in$ **SSM**: $\exists$ ***PPD*** $\in$ **WPM** $\wedge$ ***PPD*** = (**P,** *y, z*)

End Definition [Web site Presentation Model]

Intuitively, a Web site Presentation Model is a set of Page Presentation Designs, where all pages of the web site have a page presentation design.

Chapter 5: WSDM Adaptation Model

“Prediction is very difficult, especially about the future”
(Niels Bohr)

The previous chapter presented the WSDM design models in a formal way. In this chapter, the necessary extensions to this formal model to support adaptive behaviour in WSDM, are defined. The next chapter will show how to use this adaptation model in order to improve the navigation structure of the web site.

The ability for specifying adaptive behaviour (based on information gathered from *all* visitors) requires four additions to the WSDM formal models defined in the previous section:

- **Storage Facility:** some information concerning the user's browsing behaviour needs to be stored (at runtime). To allow storing information related to elements of WSDM design models, *overlay models* will be used.
- **Basic Information Storage Operations:** to allow the designer to specify what information needs to be stored (in the storage facility described above) or how information that is stored needs to be modified, some operations are needed
- **Basic Model Transformations:** defines the elementary transformations (defined on the WSDM design models), to alter (adapt) an (existing) web site design (e.g. navigation, page structure, etc).
- **High Level Adaptation Specification**: a language that allows the designer to specify which actions need to be undertaken as a result of some behavioural pattern that was detected (at runtime). This language combines the Basic Information Storage Operations and model transformations with basic language constructs (e.g. events, rules, expressions, etc) needed to support design time specified adaptive behaviour (at runtime)

To increase readability, the same notational conventions will be used as in the previous chapter:

- Where needed, immediately after stating a definition, a short intuitive explanation is given, so it will be easier for the reader to understand the formalization.
- In definitions, sets are denoted **in bold.**
- The web site **W** and the Navigation Model **M** are denoted **in bold**
- Variables used in definitions are denoted *in italics*.

Again, the following sets will be used frequently. Therefore, we define them here:

Let

$\mathbb{S}$	be the set of strings
$\mathbb{N}$	be the set of integers

The remainder of this chapter is structured as follows. Section 5.1 describes the additions to WSDM needed for Storage Specification. Section 5.2 defines how the Storage facility can be populated and manipulated. Section 5.3 presents the basic model transformations needed to alter (part of) a web site design. Finally, section 5.4 presents ASL, a high level adaptation language, based on the formal WSDM design models described in the previous chapter, and the adaptation models described in section 1 to 3 of this chapter.

5.1 Web site Overlay Model

The Web site Overlay Model is specified as a set of overlay models. An overlay model is a model that is defined on top of another model, enriching some or all elements of the original model with extra information. Overlay models [Carr and Goldstein, 1977] are extensively used in (adaptive) hypermedia (e.g., AHAM [De Bra et al, 1999], AHA! [De Bra et al, 2003], LAOS [Cristea and Kinshuk, 2003]) to store relevant knowledge about the concepts of the domain represented on the web site (called *domain model*, *conceptual model* or *application model*, depending on the source and granularity at which the domain is considered). Overlay models are popular because they leave the original model (in Adaptive Hypermedia Systems, or short AHS, most commonly the domain model) intact, yet provide the means to store additional information needed for adaptation (in AHS most commonly the 'knowledge level' about the concepts of the domain model).

For WSDM, we will allow storage specification on top of the different WSDM design models, offering the provision to store relevant information at different levels of abstraction of the design. The full Web site Overlay Model will thus be a set of individual overlay models, one for each relevant design model of the WSDM method.

First, a generic definition for an overlay model (here called Overlay Set) is given. Based on this generic definition, the different overlay models used for the Web site Overlay Model will be defined.

Definition [Overlay Set]
Let
Set be an arbitrary set

Then, $\mathbf{Set}^{ov} \subseteq \mathbf{Set} \times \mathbb{S} \times \mathbb{N}$ is an overlay set for **Set.**

End Definition [Overlay Set]

Intuitively, in an overlay set, each element of the original set can have one or more attribute/value pairs attached. In this definition, integer values are used as values, as this will be sufficient for the purpose of this dissertation.

Using the definition of overlay set, the overlay models needed to perform adaptation in WSDM are defined. As we will see later on adaptation in WSDM will be possible at two different levels:

- ***Conceptual Level***: to allow the adaptation at the level of the conceptual design, relevant information about Object Chunks (data models describing the information needed to satisfy a requirement)[46], Nodes (conceptual navigation entities, grouping information/functionality logically belonging together), Connections (between Object Chunks and Nodes), and Links will be stored in overlay sets.
- ***Implementation Design Level***: to allow the adaptation at the level of the implementation design, relevant information about Pages and their corresponding Presentation Design will be stored in overlay sets.

For the Conceptual Level, an overlay model on the Navigation Model is defined:

Definition [Navigation Overlay Model]
Let
$\mathbf{M} = (\mathbf{N}, \mathbf{H}, \mathbf{C}, \mathbf{L})$ be the Navigational Model for a web site **W**

A *Navigation Overlay Model* for **M**, denoted $\mathbf{M}^{ov}$, is a four-tuple $(\mathbf{N}', \mathbf{H}', \mathbf{C}', \mathbf{L}')$ where:

- $\mathbf{N}'$ is an overlay set $\mathbf{N}^{ov}$ for **N**
- $\mathbf{H}'$ is an overlay set $\mathbf{H}^{ov}$ for **H**
- $\mathbf{C}'$ is an overlay set $\mathbf{C}^{ov}$ for **C**

[46] Note that we consider Object Chunks as atomic units for which relevant information can be stored. We do not store information at a finer granularity (i.e. the (domain) concepts used to model object chunks).

- L' is an overlay set $L_{[S, P, A, R]}{}^{ov}$ for $L_{[S, P, A, R]}$

End Definition [Navigation Overlay Model]

Intuitively, a Navigation Overlay Model provides an arbitrary number of attribute/value pairs for nodes, chunks, connections and links. In practice, the connections overlay set will not be used in this dissertation (e.g. it will remain empty), but it is specified for reasons of completeness.

For the Presentation Design, an overlay set for both the Site Structure Model (i.e. the set of pages of the web site) and the Web site Presentation Model (the set of corresponding Page Presentation Designs) is defined. As both the Site Structure and the Web site Presentation Model are single sets, their corresponding overlay models are simply overlay sets:

Definition [Site Structure Overlay Model]
Let

SSM	be the (complete) Site Structure Model for the web site **W**

A *Site Structure Overlay Model* for **SSM** is an overlay set $\mathbf{SSM}^{\mathbf{ov}}$ for **SSM**.

End Definition [Site Structure Overlay Model]

Intuitively, a Site Structure Overlay Model is an overlay set for the Site Structure Model (i.e. a set of Abstract Pages). It thus attaches an arbitrary number of attribute/value pairs to the set of Abstract Pages of a web site. The overlay model for the Web site Presentation Model is defined as follows:

Definition [Web site Presentation Overlay Model]
Let

WPM	be a Web site Presentation Model for **W**

A *Web site Presentation Overlay Model* for **WPM** is an overlay set $\mathbf{WPM}^{\mathbf{ov}}$ for **WPM**.

End Definition [Web site Presentation Overlay Model]

Intuitively, the Web site Presentation Overlay Model is an overlay set for the Web site Presentation Model (i.e. the set of Presentations for Abstract Pages) and attaches an arbitrary number of attribute/value pairs to the set of Page Presentation Designs of a web site.

The Navigation, Site Structure and the Web site Presentation Overlay Models allow the designer to specify storage capability for relevant information at different levels of the web site design process. Relevant information not limited to one single design element, but global to the whole web site (e.g. amount of visitors, average amount of clicks per session, etc.) is also desirable. Therefore we define a Web site Overlay Model that contains, next to the Navigation, Site Structure and Web site Presentation Overlay Models, also a set of attribute/value pairs, serving as overlay set for information global to the web site. The Web site Overlay Model is defined as follows:

Definition [Web site Overlay Model]
Let

W	be a web site
M = (**N**, **H**, **C**, **L**)	be the Navigational Model for the web site **W**
SSM	be the (complete) Site Structure Model for the web site **W**

WPM	be a Web site Presentation Model for **W**

A Web site Overlay Model is a four-tuple (**G, M^{ov}, SSM^{ov}, WPM^{ov}**), where G $\subseteq$ {W} x $\mathbb{S}$ x $\mathbb{N}$ is an Overlay Set for **W**, and **M^{ov}, SSM^{ov}, WPM^{ov}** are respectively a Navigation-, a Site Structure-, and a Web Presentation Overlay Model.

End Definition [Web site Overlay Model]

Using a Web site Overlay Model, attributes can be attached to arbitrary design elements, at different levels of abstraction as well as to the global web site. This makes it possible to store relevant information (gathered at runtime) about the use of the web site (at runtime). For example, one might track the total amount of sessions (for the web site), the amount of accesses to a certain page, the amount of link-traversals, etc. The operations needed to update the Overlay Sets in the Web site Overlay Model are described in the next section.

5.2 Basic Information Storage Operations

Populating a Web site Overlay Model requires some basic operations to attach attributes, from now on called (tracking) variables[47], to (existing) design elements, and to set and retrieve the value of these variables. In this section, we will define these basic operations for the Web site Overlay Model.

Definition [addVariable]
Let

WOM = (**G, M^{ov}, SSM^{ov}, WPM^{ov}**)	be a Web site Overlay Model for the Web site **W**
M = (**N**, **H**, **C**, **L**)	be the Navigational Model for the web site **W**
variableName	$\in \mathbb{S}$, a string

The operation addVariable(**WOM**, element, variableName) is an injective function that transforms a Web site Overlay Model **WOM** to a Web site Overlay Model **WOM' = (G', M'^{ov}, SSM'^{ov}, WPM'^{ov})**, where:

If *element* $\in$ **N**
- **G' = G**
- **M'^{ov}** = (**N^{ov}** $\cup$ {(*element*, variableName, 0)}, **H^{ov}**, **C^{ov}**, **L^{ov}**)
- **SSM'^{ov} = SSM'^{ov}**
- **WPM'^{ov} = WPM'^{ov}**

If *element* $\in$ **H**
- **G' = G**
- **M'^{ov}** = (**N^{ov}**, **H^{ov}** $\cup$ {(*element*, variableName, 0)}, **C^{ov}**, **L^{ov}**)
- **SSM'^{ov} = SSM'^{ov}**
- **WPM'^{ov} = WPM'^{ov}**

If *element* $\in$ **C**
- **G' = G**

[47] Attributes and variables are synonyms in this case: they are used to store and refer to a certain value (attached to a design element). Speaking of attributes, as was done in the previous section, is consistent with AHS terminology; speaking of variables will be more intuitive when describing the Adaptation Specification Language in section 5.4

- $\mathbf{M'^{ov}} = (\mathbf{N^{ov}}, \mathbf{H^{ov}}, \mathbf{C^{ov}} \cup \{(\mathit{element}, \text{variableName}, 0)\}, \mathbf{L^{ov}})$
- $\mathbf{SSM'^{ov}} = \mathbf{SSM'^{ov}}$
- $\mathbf{WPM'^{ov}} = \mathbf{WPM'^{ov}}$

If *element* $\in$ **L**
- $\mathbf{G'} = \mathbf{G}$
- $\mathbf{M'^{ov}} = (\mathbf{N^{ov}}, \mathbf{H^{ov}}, \mathbf{C^{ov}}, \mathbf{L^{ov}} \cup \{(\mathit{element}, \text{variableName}, 0)\})$
- $\mathbf{SSM'^{ov}} = \mathbf{SSM'^{ov}}$
- $\mathbf{WPM'^{ov}} = \mathbf{WPM'^{ov}}$

If *element* $\in$ **SSM**
- $\mathbf{G'} = \mathbf{G}$
- $\mathbf{M'^{ov}} = (\mathbf{N^{ov}}, \mathbf{H^{ov}}, \mathbf{C^{ov}}, \mathbf{L^{ov}})$
- $\mathbf{SSM'^{ov}} = \mathbf{SSM'^{ov}} \cup \{(\mathit{element}, \text{variableName}, 0)\}$
- $\mathbf{WPM'^{ov}} = \mathbf{WPM'^{ov}}$

If *element* $\in$ **WPM**
- $\mathbf{G'} = \mathbf{G}$
- $\mathbf{M'^{ov}} = (\mathbf{N^{ov}}, \mathbf{H^{ov}}, \mathbf{C^{ov}}, \mathbf{L^{ov}})$
- $\mathbf{SSM'^{ov}} = \mathbf{SSM'^{ov}}$
- $\mathbf{WPM'^{ov}} = \mathbf{WPM'^{ov}} \cup \{(\mathit{element}, \text{variableName}, 0)\}$

If *element* = W
- $\mathbf{G'} = \mathbf{G} \cup \{(\text{W}, \text{variableName}, 0)\}$
- $\mathbf{M'^{ov}} = (\mathbf{N^{ov}}, \mathbf{H^{ov}}, \mathbf{C^{ov}}, \mathbf{L^{ov}})$
- $\mathbf{SSM'^{ov}} = \mathbf{SSM'^{ov}}$
- $\mathbf{WPM'^{ov}} = \mathbf{WPM'^{ov}}$

End definition [addVariable]

Intuitively, the *addVariable* operation attaches to a particular design element in one of the sub-models of the Web site Overlay Model, a (tracking) variable with an initial value 0. Once attached, the value of the (tracking) variable can be set, or retrieved. We define operations to do this next.

To avoid lengthy repetitions for each element, as it has been done in the definition of *addVariable*, the operations *value* and *setVariable* are defined in a more generic way. Note that a more precise definition, similar to the *addVariable* operation, could be given.

Definition [value]
Let

$\mathbf{WOM} = (\mathbf{G}, \mathbf{M^{ov}}, \mathbf{SSM^{ov}}, \mathbf{WPM^{ov}})$	be a Web site Overlay Model for the Web site **W**
element	be a node, chunk, connection, link, page or page presentation for the web site **W**, or **W** itself
(*element,* variableName*,* variableValue) $\in (\mathbf{G} \cup \mathbf{M^{ov}} \cup \mathbf{SSM^{ov}} \cup \mathbf{WPM^{ov}})$	be a triple of the Web site Overlay Model WOM

The operation value(**WOM**, *element*, variableName) is a surjective function that returns the integer value *variableValue* of the corresponding (*element,* variableName*,* variableValue) of the Web site Overlay Model **WOM**.

End Definition [value]

Intuitively, the *value* operation returns the value corresponding to a certain (tracking) variable attached to a design element (stored in the Web site Overlay Model).

The *setVariable* operation is defined as follows:

Definition [setVariable]
Let

$\mathbf{WOM} = (\mathbf{G}, \mathbf{M}^{ov}, \mathbf{SSM}^{ov}, \mathbf{WPM}^{ov})$	be a Web site Overlay Model for the Web site **W**
element	be a node, object chunk, connection, link, page or page presentation for the web site **W**, or **W** itself
(*element, variableName, variableValue*) $\in (\mathbf{G} \cup \mathbf{M}^{ov} \cup \mathbf{SSM}^{ov} \cup \mathbf{WPM}^{ov})$	be a triple of the Web site Overlay Model WOM

The operation setVariable(**WOM**, *element*, *variableName, newValue*) is an injective function that transforms a Web site Overlay Model **WOM** to a Web site Overlay Model $\mathbf{WOM'} = (\mathbf{G'}, \mathbf{M'}^{ov}, \mathbf{SSM'}^{ov}, \mathbf{WPM'}^{ov})$, where the tuple (*element*, *variableName, oldValue)* in **WOM** is replaced by the tuple (*element*, *variableName, newValue*) in **WOM'**.

End Definition [setVariable]

Intuitively, the *setVariable* operation sets the value corresponding to a certain variable attached to a design element (stored in the Web site Overlay Model) to a new value.

Using the *addVariable* and *setVariable* operations to create (tracking) variables and set their value, and the *value* operation to retrieve a variables value, numerical information can be stored in the Web Site Overlay Model about the different design elements of the web site. Based upon this information, the Navigation Model and Site Structure Model of a web site can be changed using adaptive behaviour. The basic operations needed to specify adaptive behaviour are described in the next section.

5.3 Basic Model Transformation Operations

Adapting the structure and navigation of a web site requires some basic operations to manipulate the Navigation Model and the Site Structure Design. The following four types of basic model transformation operations are distinguished:

(1) ***Operations on nodes:*** it needs to be possible to add or delete nodes of the Navigation Model. Adding nodes to the Navigation Model will give the designer the possibility to re-arranging the conceptual navigation structure. Using operations to connect and disconnect object chunks (see (2)), the designer can associate information (object chunks) to newly created nodes. Using link operations (see (3)), the designer can place the node into the conceptual site structure.

(2) ***Operations on object chunks:*** it needs to be possible to connect or disconnect object chunks to or from a node in the Navigation Model. Doing so, the designer can re-arrange the information/functionality in the web site.

(3) ***Operations on links:*** it needs to be possible to add or remove links of the Navigation Model. Doing so, the designer can re-arrange the link structure of the web site.

(4) ***Operations on pages:*** it needs to be possible to add or remove nodes from Pages in the Site Structure Model, and to add or remove pages from the Site Structure Model. Doing so, the designer can change the site structure of the web site.

In the following sub-sections, these four types of operations will be defined:

5.3.1 Operations on nodes

Two operations on nodes are needed: *addNode* and *deleteNode*.

Definition [addNode]

Let

M	be the navigational model (**N**, **H**, **C**, **L**) for the web site **W**
n	$\notin$ N be a node

The opertion addNode(**M**, n) is an injective function that transforms a Navigational Model **M** to a Navigational Model **M’** = (**N’**, **H’**, **C’**, **L’**) , where:

- $N' = N \cup \{n\}$
- $H' = H$
- $C' = C$
- $L' = L$

End definition [addNode]

Intuitively, the *addNode* operation adds a node to the navigational model. Note that after the operation *addNode*, the node is not connected to other nodes, nor does it have (yet) any information associated (i.e. object chunks). For this purpose, other operations, respectively on links and chunks, are defined (see next sub sections).

Definition [deleteNode]

Let

M	be the navigational model (**N**, **H**, **C**, **L**) for the web site **W**
n	$\in$ N be a node $\wedge$ n is not the root of **M**

The operation deleteNode(**M**, n) is an injective function that transforms a Navigational Model **M** to a Navigational Model **M’** = (**N’**, **H’**, **C’**, **L’**) , where:

- $N' = N \setminus \{n\}$
- $H' = H$
- $C' = C \setminus \{c \in C \mid source(c) = n\}$
- $L' = (L_{[S]} \setminus \{l \in L_{[S]} \mid source(l) = n \vee target(l) = n\}, L_{[R]} \setminus \{l \in L_{[R]} \mid source(l) = n \vee target(l) = n\}, L_{[P]} \setminus \{l \in L_{[P]} \mid source(l) = n \vee target(l) = n\}, L_{[A]} \setminus \{l \in L_{[A]} \mid source(l) = n \vee target(l) = n\})$

End definition [deleteNode]

Intuitively, the *deleteNode* operation removes a node from the set of nodes of the navigational model. All connections to object chunks, or links, from or to the removed node are also removed from the Navigation Model. Deletion of a node can thus possible cause that some information (conveyed in the object chunks connected to that node) is no longer shown on the web site, however the object chunk itself is not removed (from the set of object chunks H).

When the Navigational Model **M** is clear from the context, for both the *addNode(M, n)* and *deleteNode(M, n)* operation, we can also use the short notation *addNode(n)* and *deleteNode(n)*.

5.3.2 Operations on chunks

Two operations on object chunks are needed: *connectChunk* and *disconnectChunk*. Adding and deleting object chunks themselves is not considered useful, as we aim to allow adapting the *existing* Navigation Model, rather than adding or deleting information/functionality (represented by means of object chunks) to/from the web site.

Definition [connectChunk]

Let

M	be the navigational model (**N**, **H**, **C**, **L**) for the web site **W**
n	$\in$ N be a node
h	$\in$ H be an object chunk

The operation connectChunk(**M**, n, h) is an injective function that transforms a Navigational Model **M** to a Navigational Model **M'** = (**N'**, **H'**, **C'**, **L'**) , where:

- $N' = N$
- $H' = H$
- $C' = C \cup \{(n, h)\}$
- $L' = L$

End definition [connectChunk]

Intuitively, the *connectChunk* operation connects a piece of information/functionality, modelled by means of an object chunk, to a node. Connecting an object chunk to a node causes the information conveyed in the object chunk to be available on the page containing the node.

Definition [disconnectChunk]

Let

M	be the navigational model (**N**, **H**, **C**, **L**) for the web site **W**
(n, h) $\in$ **C**	where n $\in$ N and h $\in$ H, be a connection between a node and an object chunk

The operation disconnectChunk(M, n, h) is an injective function that transforms a Navigational Model M to a Navigational Model M' = (N', H', C', L') , where:

- $N' = N$
- $H' = H$
- $C' = C \setminus \{(n, h)\}$
- $L' = L$

End definition [disconnectChunk]

Intuitively, the *disconnectChunk* operation disconnects an object chunk from a node. Consequently, the information/functionality conveyed in the object chunk may no longer be available on the page containing the particular node (the information/functionality may still be reachable via another node if the object chunk was also connected to this other node).

Again, when the Navigational Model **M** is clear from the context, for both the *connectChunk(M, n, h)* and *disconnectChunk (M, n, h)* operation, we can use the short notation *connectChunk(n, h)* and *disconnectChunk (n, h).*

5.3.3 Operations on links

Next to connecting and disconnecting chunks to nodes, we also define operations to connect or disconnect the nodes themselves by means of links. Adding and removing links effectively gives the designer the possibility to change the conceptual navigation structure of the web site.

As WSDM makes a distinction between structural-, semantic relationship-, process logic-, and navigational aid links, we need to provide separate operations for all of them. Below, *add*- and *deleteStructuralLink* are given; the other operations are defined similarly.

Definition [addStructuralLink]
Let

M	be the navigational model (**N**, **H**, **C**, **L**) for the web site **W**
$l = (n_1, n_2)$	$\notin L_{[S]}$ and $n_1, n_2 \in$ **N**

The operation addStructuralLink(**M**, l) is an injective function that transforms a Navigational Model **M** to a Navigational Model **M' = (N', H', C', L')** , where:

$N' = N$
$H' = H$
$C' = C$
$L' = (L_{[S]} \cup \{(n_1, n_2)\}, L_{[R]}, L_{[P]}, L_{[A]})$

End definition [addStructuralLink]

Intuitively, the *addStructuralLink* operation adds a structural link (between two nodes) to the Navigation Model. Note that this operation only adds the conceptual link to the Navigation Model; it is not (yet) specified on which primitive presentation element the link will be placed in the actual web site implementation (e.g. the label of the link).

Similarly, the definition for *deleteStructuralLink* is given:

Definition [deleteStructuralLink]
Let

M	be the navigational model (**N**, **H**, **C**, **L**) for the web site **W**
$l = (n_1, n_2)$	$\in L_{[S]}$ be a (structural) link, with $n_1, n_2 \in$ **N**

The operation deleteStructuralLink(**M**, l) is an injective function that transforms a Navigational Model **M** to a Navigational Model **M' = (N', H', C', L')** , where:

$N' = N$
$H' = H$
$C' = C$
$L' = (L_{[S]} / \{(n_1, n_2)\}, L_{[R]}, L_{[p]}, L_{[A]})$

End definition [deleteStructuralLink]

Intuitively, the *deleteStructuralLink* operation deletes a link from the Navigation Model of the web site. Using (multiple) *add-* and *deleteLink* operations, the designer can connect or disconnect nodes from the navigation model, possible making some information (i.e. object chunks) connected to these nodes unavailable (e.g. no longer reachable from the root of the web site). It is the responsibility of the designer not to specify rules that can endanger reachability of nodes from the navigation structure (i.e. reachability from the root)[48].

Again, when the Navigational Model **M** is clear from the context, for both the *addStructuralLink(M, l)* and *deleteStructuralLink(M, l)*[49] operation, , we can use the short notation *addStructuralLink(l)* and *deleteStructuralLink(l).*

5.3.4 Operations on Pages

Four operations on pages are defined: *addPage*, *deletePage*, *addNodeToPage* and *deleteNodeFromPage*. Removing and adding pages from/to the Site Structure Model enables the designer to re-arrange the pages of the web site. By adding and removing nodes to/from pages, the designer can effectively change the content of web pages (e.g. availability of nodes, and thus the information conveyed in their connected object chunks, on pages). Note however that by using these operations the conceptual navigation structure will not change: the Navigation Model, more specifically the links between nodes, is left unchanged.

Definition [addPage]
Let

SSM	be the (complete) Site Structure Model for the web site **W**
P	$\notin$ **SSM** be an Abstract Page

The addPage(**SSM**, **P**) operation is an injective function that transforms a Site Structure Model **SSM** to a Site Structure Model **SSM'**, where:

- **SSM' = SSM $\cup$ {P}**

Definition [addPage]

Intuitively, the *addPage* operation allows the designer to add an Abstract Page to the web site.

Deleting an Abstract Page from the Site Structure Model is defined as follows:

Definition [deletePage]
Let

SSM	be the (complete) Site Structure Model for the web site **W**
P	$\in$ **SSM** be an Abstract Page

The operation deletePage(**SSM**, **P**) is an injective function that transforms a Site Structure Model **SSM** to a Site Structure Model **SSM'**, where:

- **SSM' = SSM / {P}**

[48] Note that this could be easily detected automatically, provided we start with a Navigation Model in which all nodes are reachable. In this case, for each deleted link, it would suffice to detect if the target node of the link is also (still) the target of another link.
[49] And similar for *addSemanticLink*, *addNavigationAidLink*, *addProcessLogicLink*, and their respectively delete-link counterparts.

Definition [deletePage]

Intuitively, the *deletePage* operation allows the designer to delete an Abstract Page from the web site.

Adding and deleting pages does not allow refining the content of an Abstract Page. Using *addNodeToPage* and *deleteNodeFromPage*, the designer is able to add or delete nodes from pages, effectively changing the information that is available on that particular page. Adding nodes to an Abstract Page can be done using the *addNodeToPage* operation:

Definition [addNodeToPage]
Let

M	be the navigational model (**N**, **H**, **C**, **L**) for the web site **W**
SSM	be the (complete) Site Structure Model for the web site **W**
n	$\in$ N be a node
P	$\in$ **SSM** be an Abstract Page
P'	be an Abstract Page

The addNodeToPage(**SSM**, n, **P**) operation is an injective function that transforms a Site Structure Model **SSM** to a Site Structure Model **SSM'**, where:

- $\mathbf{P'} = \mathbf{P} \cup \{n\}$
- $\mathbf{SSM'} = (\mathbf{SSM} / \{\mathbf{P}\}) \cup \{\mathbf{P'}\}$

End definition [addNodeToPage]

Intuitively, the *addNodeToPage* operation adds a node to an (existing) Abstract Page. As a result, the information/functionality conveyed in the object chunks connected to the node will be available on the Abstract Page.

Next to adding nodes, the page content can also be changed by deleting nodes from Abstract Pages. The definition for *deleteNodeFromPage* is as follows:

Definition [deleteNodeFromPage]
Let

M	be the navigational model (**N**, **H**, **C**, **L**) for the web site **W**
SSM	be the (complete) Site Structure Model for the web site **W**
P	$\in$ **SSM** be an Abstract Page
n	$\in$ **P** be a node

The operation deleteNodeFromPage(**SSM**, n, **P**) is an injective function that transforms a Site Structure Model **SSM** to a Site Structure Model **SSM'**, where:

- $\mathbf{P'} = \mathbf{P} / \{n\}$
- $\mathbf{SSM'} = (\mathbf{SSM} / \{\mathbf{P}\}) \cup \{\mathbf{P'}\}$

End definition [deleteNodeFromPage]

Intuitively, the *deleteNodeFromPage* operation removes a node from an Abstract Page of the Site Structure Model of the web site. Note that this does not delete the node from the web site: it is still present in the Navigation Model. The designer can chose to add the node to another Abstract Page using the addNodeToPage operation. It is the responsibility of the designer to make sure the Site Structure Model remains complete (i.e. all nodes are represented on some Abstract Page).

When the Site Structure Model **M** is clear from the context, for addPage(**SSM**, **P**), deletePage(**SSM**, P), addNodeToPage(**SSM**, n, **P**) and deleteNodeFromPage(**SSM**, n, **P**) operations, we can use the short notation addPage(**P**), deletePage(**P**), addNodeToPage(n, **P**) and deleteNodeFromPage(n, **P**).

5.4 Adaptation Specification Language (ASL)

In the first two sections of this chapter, overlay models and the Basic Information Storage Operations, to store and manipulate relevant information about elements of different levels of abstraction within the web site, were defined. Basic model transformation operations to alter the navigation and presentation design of a web site were defined in the previous section. These models, and the corresponding operations defined upon them, are sufficient to specify basic transformations on the WSDM models. Although these transformations can be used to adapt the design of a web site based on runtime observations of how the web site is used, it is not possible for the designer to specify *which* (runtime) information should be stored, *when* the WSDM design models should be adapted and *how frequently*. In other words, it is not (yet) possible to specify the adaptive (runtime) behaviour of the web site at design time.

Therefore, a high level rule-based language, called Adaptation Specification Language (ASL), is defined, aimed at providing the designer with an easy formalism to specify complex adaptation strategies on a WSDM web site design. This allows the designer to specify, at design time, which adaptation will be permitted at runtime (e.g. during the life time of the web site). The rules of the language are event-condition-action based (i.e. ECA rules, see [Dayal, 1988]). ASL allows specifying the relevant events for a web site and its browsing sessions. Conditions are specified on information stored in the Web site Overlay Model and use basic arithmetic expressions. Actions are specified using the Basic Model Transformation Operations defined on the different WSDM design models. Thus, the basis of ASL is the Information Storage Operations defined on the Web site Overlay Model, and the Model Transformations defined on the WSDM design models. This un-ambiguously defines the semantics of this part of[50] the Adaptation Specification Language. Next to the possibility to apply the basic model transformations defined in the previous sections, ASL also offers the designer the necessary constructs to specify:

- ***Expressions, basic arithmetic operations*** and ***statistical functions***: needed to specify conditions (using values from the Web site Overlay Model).
- ***Control flow*** (loop, if-then, …): needed to specify actions on (sub)sets of elements (e.g. nodes, links, object chunks, etc).
- ***Rules*** (i.e. conditions and actions): needed to define complex adaptation strategies.
- ***Adaptation Strategies and Policies:*** needed to specify exactly which and when adaptation needs to be performed

For this dissertation, we opted for the specification of the intuitive, expressive and specific programming language, instead of using an existing (high level) general programming language (e.g. Java). Note that the problem at hand (i.e. offering the designer the possibility to define runtime adaptation in a web site design) is a very specific one, in a very specific domain (i.e. the domain of web site engineering/design). In such a case, it is well supported by the field of *domain specific languages* (for an extensive survey on domain specific language, see [van Deursen et al, 2000]) that constructing a designated language which' instructions lean closely towards the level of abstraction of the domain, provides several advantages (see [van Deursen et al, 2000]):

[50] Namely, the part that concerns the Adaptation Model (i.e. adding, modifying and retrieving attribute/value pairs from/to the Web site Overlay Model) and the basic transformation operations on the WSDM design models.

- **Solutions can be expressed at the level of abstraction of the problem domain level**: intuitive action in the problem domain (e.g. linking two nodes, deleting a page, …) are translated in native ASL operations.
- **Re-usable**: Scripts allow specification of re-usable actions specified on the level of abstraction of the domain. Adaptation policies and strategies can also be re-used.
- **Portability**: an ASL implementation is not dependent on any implementation platform
- **Validation and optimization**: although not discussed in this dissertation, as most instructions in ASL can be mapped one-to-one on basic model transformations or information storage operations, optimization and validation, to some extent, can be done (theoretically) using the formal specifications (e.g. reachability after a transformation).

In this sense, ASL can be called a domain-specific language. ASL aims to keep the middle between the expressiveness of a generic (high level) programming language and the specificity of a domain specific language. ASL does not offer user defined functions, free loops (e.g. while-loops), recursion, error handling, goto's, etc … features commonly found in a higher level programming language. Hereby, it is aimed to drastically reduce the learning curve for web designer to go through to be able to specify adaptation using ASL. In general, a web designer may not be able or willing to cope with the complexity of a generic (high level) programming language.

The syntax of ASL is defined using Backus Naur Form (BNF) [Backus et al, 1960]. In the next subsections, the BNF notation of ASL is given in steps, and explained gradually. The complete BNF can be found in Appendix 1.

To increase readability, the following notational conventions will be used in the next subsection describing the BNF of ASL:

- Lines in the BNF will be indented (relative to the surrounding explanatory text)
- Keywords of ASL will be **in bold**
- BNF production rules defining the same element are grouped together

5.4.1 General Overview

Before going into deeper detail on the formal (syntactic) specification of the Adaptation Specification Language, a general overview of its architecture is given. In the BNF specification of the Adaptation Specification Language, there are four main sets of production rules:

- **Adaptation Policies and Strategies**, used to express when and what adaptation needs to be performed
- **(ASL) Rules,** used to express both complex and simple adaptation (rules)
- **Expressions,** used to do (arithmetic) calculations
- **(Ordered) Set Expressions,** used as a grouping mechanism to use and manipulate the elements contained in the different design models

The following figure schematically shows which syntactical constructions *glue* these four main sets together. For example, a 'statistical operation' takes as input an ordered set and returns a number (i.e. an expression). The four main sets (adaptation strategies & policies, rules, expressions and set expressions) are explained in detail in the following sub sections, as are the syntactical constructions for the 'glue' between these sets (found in figure 5.1).

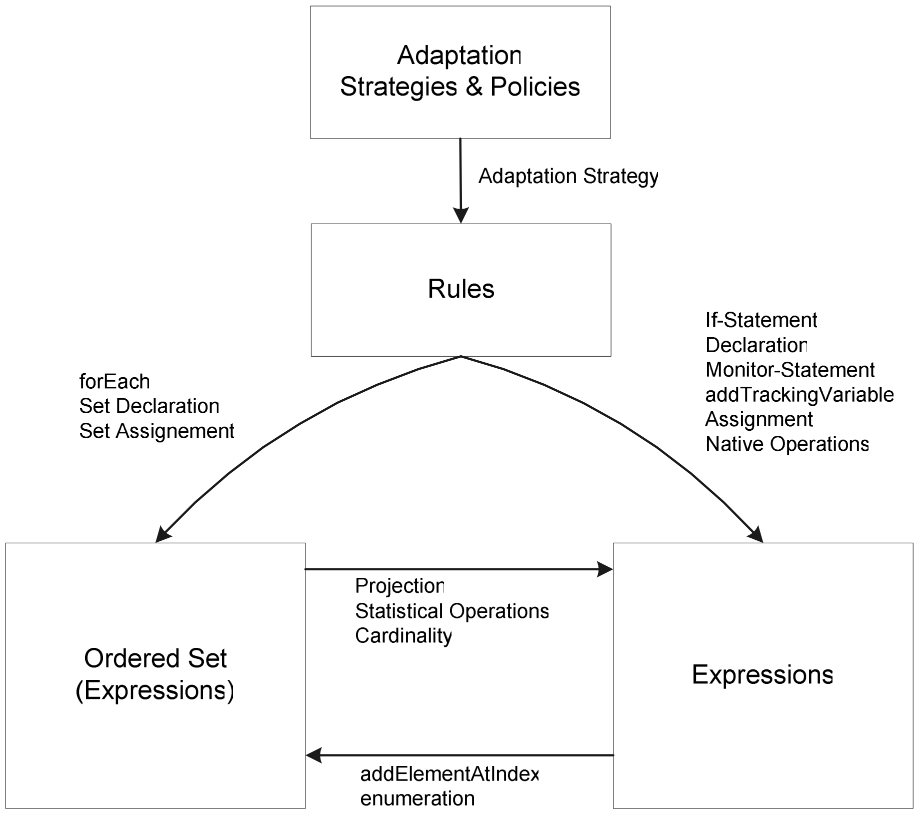


Figure 5-1 Schematic Overview of the Adaptation Specification Language

The next section gives an informal overview of how the (adaptation) rules will be used to express adaptation strategies and policies.

5.4.2 Adaptation Strategies and Policies

Using one or more (adaptation) rules, the designer is able to specify an **adaptation strategy** for the web site. An adaptation strategy specifies how, based on the monitoring of user browsing behaviour, the structure and presentation of the web site should be adjusted. It also specifies which information about the user browsing behaviour should be monitored. Both the specification of which user behaviour (at runtime) needs to be monitored and which changes (to structure and presentation of the web site) this behaviour may induce are given at design time.

Specifying at design time adaptation strategies for a web site, the designer wields a powerful tool to:

1. ***anticipate and incorporate runtime information in the design:*** constructing a web site is usually done following the requirement gathering - design - implement - deploy cycle[51]. If the designer wants to take into account runtime information in his design (e.g. offer a link to a popular page on the homepage), he is forced to go through a new design cycle, manually re-designing (and re-implementing, deploying) the web site to reflect the desired change(s). Alternatively, he can (manually) implement the desired changed directly at implementation level, leaving his design and implementation out of sync and decreasing the maintainability of the web site as a whole. Using adaptation strategies, the designer is given the opportunity to anticipate to runtime information in his design, and specify rules to automatically adjust the

[51] For smaller web sites, the design cycle is often (and unfortunately) collapsed to an implement – deploy cycle.

navigation structure according to this information. A new design – implement – deploy cycle is not needed, and the design and implementation remain coherent. In the next chapter, an example Adaptation Strategy that promotes popular nodes closer to the homepage of the site will be given (see section 6.1.1, Promotion).

2. ***(automatically) evaluate and select among design alternatives*****:** when designing the structure and organization of a web site, the designer is constantly confronted with design choices. E.g. does certain information belong together on a page, or is it better to separate it using multiple pages? Is it better to use a flat navigation structure, or do users prefer a more hierarchical structure? Should certain information be easier accessible (e.g. available in less clicks from the homepage) compared to other? Although design guidelines and methodologies[52] exist to help the designer to make these kinds of design decisions, it is still a difficult task with no single correct answer. Using Adaptation Strategies, the designer is given the opportunity to design the site in the way that to the best of his knowledge would be the most appropriated for the users, yet being able to monitor the use of the navigation structure once the web site has been deployed, and possibly fall back on an alternative design should the use of the existing structure prove to be problematic. In the next chapter, an example Adaptation Strategy to change the main navigation structure in order to limit navigation choices on the homepage will be presented (see section 6.1.2; Demotion).
3. ***(automatically) detect and correct possible design flaws*****:** even using design guidelines and methods, formulating a web site design is a human activity and thus error prone. Using Adaptation Strategies, certain properties of the (organization of the) web site can be validated for (a certain degree of) correctness and possible design errors can be detected. Furthermore, automatic correction of these design flaws is possible, or at least the web administrator can be alerted of the problem. In the next chapter, an example adaptation strategy to automatically detect incorrectly located information/functionality in an audience driven design will be presented, along with the specification how to correct detected problems (see section 6.3, Validation of Audience Driven Navigation Structure).
4. ***better tailor the web site to satisfy business requirements:*** designing web sites, in particular web sites with a commercial aim (e.g. e-commerce sites), the designer might need to consider an implicit business goal (e.g. offer commercial information on the most popular pages, gather information from the user). At design time, the designer doesn't know how the web site will be used, and thus he might not be able to select the most optimal solution for realizing the business goals (e.g. commercial information could be put on a more strategic place reaching more visitors; or more information about the users could be gathered if the information gathering was done in a different order). Using Adaptation Strategies, the designer is able to monitor (runtime) browsing behaviour, and effectively optimize the results of business goals set by the web site owners. In the next chapter, an example adaptation strategy to re-order nodes in a sequential information gathering process, based on the existence of a bottleneck, is presented in order to optimize the amount of information gathered (see section 6.2, Re-ordering sequential information (gathering)).

The next chapter will describe in detail, example adaptation strategies covering the four cases mentioned above and show how they can be specified using the Adaptation Specification Language.

The exact time or time intervals when each adaptation strategy needs to be performed are specified in the web site's **adaptation policy**. An adaptation strategy can thus possibly be re-used for different

[52] WSDM, described in Chapters 3 (WSDM Informal) and 4 (WSDM Formalization), is one of the few web site design methods that provides a methodology. Most methods only provide the designer with the necessary notations and abstractions to describe a web site design, yet leave him in the dark on how to arrive to a good design using these techniques and notations.

web sites, possibly be performed at different time(s) or time intervals. It is the web designer who decides both which adaptation strategies to use (be it re-used from another web site, or implemented by the designer himself) and what the adaptation policy will be, depending on the web site under construction, its size, goal, intention, etc. Adaptation policies are based on events: occurrence of events triggers the evaluation (with possible adaptive changes as a result) of associated adaptation strategies. Events can be user generated (e.g. clicking a link, visiting a node, page, …) or system generated (elapse of a time interval, tracking variables attaining a certain value, …).

Without further ado we present, step by step, the Adaptation Specification Language starting from the top of the BNF specification, i.e. Adaptation Policies:

```
<adaptationPolicy> ::=
        (<script> ';')*
        <adaptationSpecification> ';' (<adaptationSpecification> ';')*
```

An adaptation policy consists of zero or more script specifications (see section 5.4.4, Scripts) followed by one or more adaptationSpecifications. An adaptationSpecification is a specification of an adaptation strategy and a trigger that specifies *when* the strategy needs to be evaluated and applied:

<adaptationSpecification> ::= ‘**when’** <trigger> ‘**do’** <adaptationStrategy>

So, the exact time(s) when Adaptation Strategies need to be performed are described in an Adaptation Policy. Four kinds of ‘triggers’ may cause (the evaluation and execution of) an adaptation strategy. Three kinds of triggers correspond to the occurrence of a certain event (i.e. system-, time- and user events); the fourth kind of trigger corresponds with the fulfilment of a condition involving tracking variables. The events are explained here, conditions are specified in section 5.4.5 (Expressions).

```
<trigger> ::= <timeEvent>
<trigger> ::= <systemEvent>
<trigger> ::= <userEvent> ['on' <reference>]
<trigger> ::= <condition>
```

User events are caused by an action of the user: starting or ending a session, clicking a link or loading a web site element (e.g. a page, a node, a chunk, …).

<userEvent> ::= ‘**click’ | ‘sessionStart’ | ‘sessionEnd’ | ‘load’**

Note that we elected to take into account only those events and user information which is reliable and easily obtainable in the context of web applications. For example, (estimated) reading time of pages, or click sequences are not taken into account, as this information cannot be derived with certainty (i.e. it is impossible to know how long a user is looking at a page; it is also uncertain what is the actual browsing order of the user, as he might open multiple pages and read them in another order)[53].

There is only one system event, the initialization of the web site itself. Typically, tracking variables will be installed when the web site is initialized (see further on).

<systemEvent> ::= ‘**initialization’**

[53] Note also that the user interface events that can be gathered in classical applications are typically richer, see e.g. [Hilbert and Redmiles, 2000].

Time events are used to specify the elapse of a certain time (interval). Time events can be absolute (e.g. *when the date is the 21st of August)* or relative (e.g. *10 weeks from now*). Furthermore, recurrent events can be specified (e.g. *every month*).

<timeEvent>::= <dateSpecifier> | ['**every**'] <timeExpression> [**'from now**']

A date is specified using the following intuitive format e.g. *10 January 2006*:

<dateSpecifier>	::= <number> <month> <number>
<month>	::= 'January'| 'February' | 'March' | 'April' | 'May' | 'June' | 'July' | 'September' | 'October' | 'November' | 'December'

A time expression is used to express some amount of time. Examples include *5 days, 1 month 2 weeks 3 days 2 minutes, etc*:

<timeExpression>	::= <timePrimitive> <timePrimitive>*
<timePrimitive>	::= <number> <timeUnit>
<timePrimitive>	::= <day>
<timePrimitive>	::= <month>
<timeUnit>	::= 'seconds' | 'minutes' | 'hours' | 'days' | 'weeks' | 'months' | 'years'
<day>	::= 'Monday' | 'Tuesday' | 'Wednesday' | 'Thursday' | 'Friday' | 'Saturday' | 'Sunday'

Consider the following examples for a better understanding of an Adaptation Policy. Note that <script> and <adaptationStrategy> are not yet given; they will be explained in detail in the next sections:

(1) <script>*
(2) **when** 20 January 2005 **do** <adaptationStrategy> ;
(3) **when** 2 hours 3 minutes **from now do** <adaptationStrategy> ;
(4) **when** initialization **do** <adaptationStrategy> ;
(5) **when every** 2 weeks **do** <adaptationStrategy> ;
(6) **when every** Monday **do** <adaptationStrategy> ;
(7) **when** sessionStart **do** <adaptationStrategy> ;
(8) **when** web site.amountOfSessions = 10000 **do** <adaptationStrategy> ;

In (2) an absolute time event is illustrated; line (3) the uses a relative time event ,a system event is used in line (4), line (5) and (6) illustrate a recurring time event, and line (7) a user event and line (8) a tracking-variable-event.

As already explained, an Adaptation Strategy consists of:

- ***a specification of what information needs to be stored*** (at runtime), concerning the use of the various web site elements (e.g. pages, page presentations, nodes, chunks, links, connections), and/or

- ***a specification of how to change the structure and/or navigation*** of the web site, using the information gathered

Adaptation Strategies are expressed by means of one or more rules:

```
<adaptationStrategy> ::= <ruleSequence> | <rule>
<ruleSequence> ::= 'begin' <rule>  (';' <rule>)* 'end'
```

Rules are explained in the next sub section.

5.4.3 Rules and Statements

Adaptation Strategies are made up of rules. In ASL there are two kinds of rules:

- ***For-each rules***: used to iterate over a certain set (e.g. the set of nodes, a set of links, etc). ASL allows specifying more than one iteration-set in a for-each rule; in this case, the for-each rule iterates over the Cartesian product of the sets.
- ***Simple rules***: used to specify simple statements (i.e. not iterating over a set)

The BNF production rules for for-each and simple rules are given as follows:

```
<rule> ::= <foreachRule> | <simpleRule>

<foreachRule> ::= 'forEach' <setIterator> <setIterator>* ':'  <adaptationStrategy>
<setIterator> ::= <reference> 'in' <setExpression>

<simpleRule> ::= <ifStatement> | <trackingVariableStatement> | <letStatement> |
<assignment> | <setLetStatement> | <setAssignment> | <nativeOperation> | <callStatement>
```

To clarify, consider the following (simple) examples of for-each rules. The first for-each rule iterates over a set (in this case, the set only contains a few nodes)[54], and alerts[55] the webmaster which elements have been processed:

```
forEach element in {node1, node2, node3} :
        alert("Processed the following node: ", element);
```

The second rule illustrates the use of two iterator-sets in a for-each loop. When using multiple iterator-sets, the iteration is done over the elements of the Cartesian product of the multiple sets.

```
forEach element1 in {node1, node2}, element2 in {node3, node4} :
        alert("Processed the following tuples: (", element1,",", element2, ")");
```

Result of execution of this rule will be the following alert to the webmaster[56]:

```
"Processed the following tuples: (node1, node3)
 Processed the following tuples: (node1, node4)
 Processed the following tuples: (node2, node3)
 Processed the following tuples: (node2, node4)"
```

[54] For the exact specification of sets, and native operations (such as the 'alert' operation), see section 5.4.7 (Set Expressions) and 5.4.8 (Native Operations and Functions).
[55] Native operations, such as the alert-operation, are discussed in section 5.4.8 (Native Operations and Functions)
[56] In any particular order, as the Cartesian product does not induce an order to the resulting tuples.

Next to for-each rules, simple-rules are used to express simple adaptation strategies over one single element (i.e. they do not require iteration). Simple rules are thus statements, and come in the following varieties:

- ***If-statements*:** used to specify the conditional execution of one or more rules (i.e. an Adaptation Strategy). The condition is specified using an expression, which is further explained in section 5.4.5 (Expressions).

  <ifStatement> ::= '**if**' <condition> '**then**' <adaptationStrategy>

  As an example, consider the following if-statement:

  if node.amountOfAccesses **>** 50 **then** <adaptationStrategy>;

- ***Let-statements:*** used to declare variables fit to hold values resulting from expressions. These variables are defined in the current environment[57]. Ordinary variables (as opposed to set variables, see next bullet) should start with a non-capitalized letter[58]. If no initial expression is specified when declaring a variable, its initial value is undefined. Variables are typically used to calculate a certain threshold to condition an adaptation rule.

  <letStatement> ::= '**let**' <variable> ['**be**' <expression>]

  As an example, consider the following let-statement:

  let threshold **be** 0;

- ***Set-let-statements:*** used to declare variables containing a set. As ordinary variables, sets are defined in the current environment. They must start with a capitalized letter[59]. If no initial set-expression is specified when declaring a set variable, its initial value is undefined. Set-variables are typically used to bind a name to a certain (frequently used) set.

  <setLetStatement> ::= '**let**' <setVariable> ['**be**' <setExpression>]

  As an example, consider the following setLet-statement:

  let ConsideredNodes **be** {node1, node2, node3};

- ***addTrackingVariable-statements:*** used to declare a tracking variable, which is connected to an element of a design models (i.e. pages, nodes, links, object chunks and connections). Tracking variables correspond to the attribute/value pairs defined in the web site overlay model. They are denoted using the relevant design element to which they are attached as a quantifier, e.g. node1.amountOfAccesses. When tracking variables are defined without explicitly initializing them, their initial value is zero.

[57] In ASL, due to the lack of user defined functions, the environment model is very simple. Next to the global environment, only for-each rules can give rise to local environments. These environments are organized in a stack-based fashion, where nested for-each rules give rise to a new local environment being pushed on the stack.
[58] See section 5.4.6, (Tracking) Variables and Set Variables
[59] See section 5.4.6, (Tracking) Variables and Set Variables

<addTrackingVariable> ::= **'addTrackingVariable'** <trackingVariable>
['**initialized on**' **<**expression>]

As an example, consider the following addTrackingVariable-statement, which attaches the tracking variable *amountOfAccesses* to the node *node1*:

addTrackingVariable *node1*.amountOfAccesses ;

- ***monitor-statements:*** used to specify which events on which design elements (i.e. pages, nodes, links, object chunks and connections) require tracking of their use (at runtime), and will result in updating one or more tracking variables (using assignments). Each tracking variable used in a monitor-statement, first needs to be declared using an addTrackingVariable statement. Updating tracking variable values corresponds to the operations *addVariable* and *setValue* (defined in the web site adaptation model).

  <monitorStatement> ::=
  '**monitor**' <userEvent> ['**on**' <reference>]
  ['**if**' <condition>] '**do**' <trackingVariableAssignmentSequence>

  Only user events may give rise to the updating of a tracking variable. Conditions will be further explained in the next section; a tracking-variable-assignment sequence consists of one or more assignments to monitoring variables. Note that the occurrence of an event on a particular element may result in updating several tracking variables.

  <trackingVariableAssignmentSequence> ::=
  <trackingVariableAssignment> |
  '**begin**'
  <trackingVariableAssignment> ('**;**' <trackingVariableAssignment >)* '**end**'

  <trackingVariableAssignment> ::= <trackingVariable> '**:=**' <expression>

  The trackingVariableAssignment statement corresponds with the basic information storage operation "setVariable(*element*, *variableName, newValue*)" defined for the Web site Overlay Model, where <trackingVariable> corresponds to the *element* and *variableName*[60], and *newValue* corresponds with the value resulting from evaluating <expression>.

  A typical example of a monitor-statement, with an assignment, is given next. In this example, the tracking variable *amountOfAccesses* that was attached to *node1* (see previous bullet) is incremented each time *node1* is loaded (i.e. requested by the user):

  monitor load **on** *node1* **do**
  node1.amountOfAccesses := *node1*.amountOfAccesses + 1 ;

- ***Assignments:*** used to assign/update values (specified using expressions, explained in section 5.4.5 (Expressions) to/on variables or tracking variables (explained in the previous bullet).

  <assignment> ::= < trackingVariableAssignment>
  <assignment> ::= <variable> '**:=**' <expression>

[60] See also section nr 5.4.6, (Tracking) Variables and Set Variables

As an example, consider the following assignment statement, which increments the *threshold* variable with 10%:

threshold **:=** threshold + (0.1 * threshold)

- ***Set-assignments:*** used to assign/update sets contained in set variables (specified using set-expressions, explained in section 5.4.7 (Set Expressions).

  <setAssignment> ::= <setVariable> ':=' <setExpression>

  As an example, consider the following set-assignment statement, which adds the nodes *node4* and *node5* to (the variable) *ConsiderNodes*:

  ConsideredNodes := ConsideredNodes UNION {node4, node5};

- ***Native-Operations:*** used to specify basic changes to the relevant design models, and correspond to basic model transformation operations defined on the Navigation and Presentation Models (see section 5.3). In ASL, these operations are called *native operations*[61]. They are further specified in section 5.4.8 (Native Operation and Functions).

- ***Call-statement:*** a call statement is used to call a script. Scripts, their intention and use are explained in the next sub section.

5.4.4 Scripts

Scripts are an abstraction mechanism similar to macros available in several well known high level programming languages (e.g. Scheme, C, …). Just like macro's, scripts do not have parameter evaluation, they do not have scope, they cannot return values and they cannot contain recursive calls. In fact, where a script is called, the body of the script is simply (lexically) copy/pasted.

Using scripts, the designer (or somebody else, e.g. an ASL expert) is given the possibility to define some frequently used higher level operations once, and use them several times. Indeed, using scripts, higher-level operations on the relevant design models can be defined using native operations. In this way, more intuitive (composed) operations can be defined, which makes the Adaptation Specification Language more intuitive to use for a non-programmer. Moreover, scripts are independent of any Adaptation Policy, and thus can be used in combination with different Adaptation Policies. In this way, libraries of scripts can be shared among designers, and even novice designers can use advanced Adaptation Strategies. In the next chapter, some useful scripts will be defined.

Scripts can be specified using the following designated syntax:

<script> ::= '**script**' <reference> '(' [<parameter> ('**,**' <parameter>)*] ')' '**:**'
<adaptationStrategy>

<parameter> ::= <variable> | <setVariable>

[61] This is conform standard programming language terminology, where pre-defined operations are called *native*.

A script has a name (i.e. the <reference>), and an arbitrary number of parameters. Parameters can be either expressions or set expressions (explained respectively in sub section 5.4.5 (Expressions) and 5.4.7 (Set Expressions)).

Scripts can be called using the following syntax:

```
<callStatement> ::= 'call' <reference> '(' [(( <expression> | <setExpression> )','
                                             ( <expression> | <setExpression> )*)] ')'
```

As a simple example, consider the following script, which changes the target of a link:

```
script moveTarget(link, newTarget) :
    begin
        addLink(getLinkType(link), source(link), newTarget);
        deleteLink(getLinkType(link), link)
    end
```

Calling this script for link1 and target node4 is done as follows:

```
call moveTarget(link1, node4);
```

The next section explains expressions, one of the pillars of ASL (see section 5.4.1, ASL General overview).

5.4.5 Expressions

Expressions are quite similar to expressions in well-known programming language such as C or Pascal: infix notation is used, and the standard precedence rules apply. Boolean operators are evaluated in the following order: not, and, or, =. Arithmetic operators are evaluated in the following order: / and *, + and -. Parentheses are used to denote higher precedence. Operators of the same precedence level are evaluated from left to right.

The BNF production rules for expressions are as follows:

```
<condition> ::= <expression>

<expression>    ::=  <comparand> (<comparator> <comparand>)*

<comparand>     ::= <term> (<adder> <term>)*

<term>          ::= <factor> (<multiplier> <factor>)*

<factor>        ::= <primitive>
<factor>        ::= <not> <primitive>

<primitive>     ::= <number>
<primitive>     ::= <boolean>
<primitive>     ::= <reference>
<primitive>     ::= <variable>
<primitive>     ::= <trackingVariable>
<primitive>     ::= <nativeFunction>
```

```
<primitive>      ::= <statisticalOperator>
<primitive>      ::= '('<expression>')'

<comparator>  ::= '<' | '=' | '>' | '<=' | '>=' | '!='
<adder>            ::=  'OR' | '+' | '-'
<multiplier>    ::= 'AND' | '/' | '*'
<not>               ::=  'NOT' | -
```

Some of the primitive expressions of ASL are those found in most common programming languages: numbers (i.e. <number>, positive integers), booleans (i.e. <boolean>, true and false), references (i.e. <reference>, names of nodes, links, chunks, connections) and variables (i.e. <variable>, explained in the next sub section). Furthermore, ASL offers access to tracking variables (i.e. <trackingVariable>, the attributes in the web site overlay model, see section 5.1, Web site Overlay Model), some native functions[62] (i.e. <nativeFunction>) and statistical operators (i.e. <statisticalOperator>), both further explained in section 5.4.8 (Native Operations and Functions).

Some examples of expressions include:

(1) (*node1*.amountOfAccesses * 50) / 3 ;
(2) ((*node1*.amountOfAccesses + *node2*.amountOfAccesses) / 2 = *threshold*)
OR (*node1*.amountOfAccesses > 10000) ;
(3) true ;
(4) 5 ;

Variables, tracking variables and set variables are explained in the next section.

5.4.6 (Tracking) Variables and Set Variables

Tracking variables can be attached to any web site element (i.e. pages, page presentations, nodes, object chunks, connections and links) or to the web site itself using the <addTrackingStatement> (see section 5.4.3, Rules and Statements). They are used to track the user browsing behaviour. Tracking variables are specified as follows:

```
<trackingVariable> ::=
        <reference> [ '['<reference> (',' <reference>)* ']' ] '.' < reference >
```

This statement correspond with the primitive operation "value(*element, variableName*)" defined for the Web site Overlay Model, that returns the value of a certain attribute (variable) attached to a certain design element. The basic form of this statement is <reference>.<reference>, where the first reference[63] denotes the design element and the second the attached attribute (variable). Consider the following example, where the value of the *amountOfAccesses* attribute attached to the design element *node1* is retrieved:

```
node1.amountOfAccesses ;
```

[62] Note that native functions are different from native operations. The native operations correspond to the basic model transformation operations defined in section 5.3 (Basic Model Transformation Operations), where the native functions do not transform any model. They are used to test certain properties of the models (e.g. is a certain node in an audience track?) or return certain elements (e.g. the root of an audience track).
[63] References are defined at the end of this section.

In a more complex form, a tracking variable can be qualified with one or more references (between "[" and "]"). This will be done when the designer wants the tracking variable assigned to a certain design element to be parameterized. For example, a designer might not want to track all accesses to a certain node; instead, he might want to track only the accesses to that node made by some persons. In this way, he might for example track the amount of accesses made by a certain audience track, to a certain node. This would be specified as follows (provided there is a 'Children' Audience Class):

```
node1[Children].amountOfAccesses ;
```
[64]

Variables in ASL (i.e. not *tracking variables*) only serve as an expedient to hold certain values, or to aid in certain calculations. There are two kinds of variables: (expression) variables, and set variables. Set variables start with a capital letter; (expression) variables start with a non-capitalized letter. The BNF specification looks as follows:

```
<variable>      ::= <reference>
<setVariable>   ::= <capitalizedLetter> (<letter> | <digit>)*
<reference>     ::= (<nonCapitalizedLetter> | <digit>)* <letter> (<letter> | <digit>)*
```

Both variables, set variables and tracking variables can be assigned a certain value with an assignment statement (see section 5.4.3, Rules and Statements).

5.4.7 Set Expressions

As explained in section 5.4.1 (ASL General Overview) the second pillar of ASL, next to expressions, is set expressions. The evaluation of a set expression always results in a set. ASL provides *native sets*[65] to refer to the sets comprising elements from relevant design models of the web site design: the audience model (i.e. audience classes), navigation model (i.e. nodes, object chunks, connections and links) and the Site Structure Model (i.e. the set of abstract pages for the web site):

<nativeSet> ::= '**Nodes**' | '**Chunks**' | '**Links**' | '**Connections**' | '**Pages'** |
'**AudienceClasses'**

To be able to refer to subsets of these native sets, ASL offers the designer set-creators, functions that create a particular subset of any of the native sets:

```
<setCreator> ::=
        'nodesInAudienceTrack('<expression>')' |
        'chunksInAudienceTrack('<expression >')' |
        'chunksFromNode('<expression >')' |
        'nodesLinkedTo('<expression>, <expression >')' |
        'nodesLinkedFrom('<expression>, <expression >')' |
        'nodesFromChunk('<expression >')' |
        'ALL' < expression > |
        'addElement('<setExpression>, <expression>')'
        'addElementAtIndex('<setExpression>, <expression>, <expression> ')'
```

[64] This statement only shows how to retrieve the value of the parameterized amountOfAccesses from *node1*. A more elaborated example of how to define and update this kind of tracking variables will be shown in section 6.3 of the next Chapter.

[65] This naming is consistent with the naming of native operations and functions introduced in the previous sub section

These set creators are explained as follows:

- **nodesInAudienceTrack**: takes as input an audience class, and outputs all nodes in the audience track of the given audience class
- **chunksInAudienceTrack**: takes as input an audience class, and outputs the object chunks connected to the nodes in the audience track of the given audience class
- **chunksFromNode**: takes as input a node, and outputs all object chunks connected to this node
- **nodesLinkedTo**: takes as input a link type (first parameter) and a node (second parameter), and outputs all the nodes this node links to (according to the specified link type)
- **nodesLinkedFrom**: takes as input a link type (first parameter) and a node (the second parameter), and outputs all the nodes that link to this node (according to the specified link type)
- **nodesFromChunk**: takes as input an object chunk, and outputs all the nodes to which this object chunk is connected
- **ALL**: takes as input an object chunk, a node or an abstract page, and outputs all the (concrete) instances of the respective object chunk, node or abstract page. The ALL set creator allows linking the conceptual representation of a design element, with its concrete instance(s). For example, a (conceptual) parameterized object chunk gives rise to a set of concrete instances of the object chunk.
- **addElement**: takes as input a set (in the form of a set expression) and an expression (the element to add). As a result, the element is added the set.
- **addElementAtIndex**: takes as input a set (in the form of a set expression), an expression (the element to add; the first <expression> parameter) and an index (specified in the form of an expression; the second <expression> parameter). As a result, the element is added to the set, at the specified index. Note that although in the formal models specifying WSDM and the WSDM adaptation model (un-ordered) sets are used, (the implementation of) ASL supports ordered bags. Order is useful to order a set of (design) elements according to a tracking variable. For example, a set of nodes could be ordered according to the *amountOfAccesses* tracking variable. Duplicate elements are needed when handling set of numbers, typically resulting from values of tracking variables of sets of (design) elements (an example will be given shortly).

When expressing adaptation strategies, it is necessary to be able to manipulate sets. ASL provides set operations for this purpose. Three set operations are supported: sorting, filtering and mapping. Set operations are denoted between [ ] immediately after a set (see explanation for setExpressions):

<setOperation> ::= <setSorter> | <setFilter> | <setMap>

- **Sorting**: only sets containing design elements (e.g. the native sets) can be sorted according to the value of tracking variables attached to these elements. The BNF specification is given as follows:

  <setSorter> ::= '**SORT ON**' <reference> ':' <trackingVariable>

  As an example, consider the following statement, where the set of (all) nodes is sorted according to the tracking variable *amountOfAccesses* (which must previously have been attached to all nodes):

  Nodes [**SORT ON** *element* : *element.amountOfAccesses*] ;

- **Filtering**: sets can be filtered according to a condition. Only the elements for which the condition holds are obtained in the resulting set. The BNF specification is given as follows:

  <setFilter> ::= '**FILTER ON**' <reference> ':' <condition>

  As an example, consider the following statement, where the set of all nodes is filtered so that only the nodes that are accessed more then 50 times are withheld:

  Nodes[FILTER ON *element*: *element.amounfOfAccesses* > 50**]** ;

- **Mapping**: a certain function can be *mapped*, i.e. applied to, each element of a set. A map does not eliminate any element from the set, as a filter does, but it transforms the element according to a function. The BNF specification is given as follows:

  <setMap> ::= '**MAP ON**' <reference> ':' <expression>

  Typically, mapping is used to obtain the set of values of a corresponding tracking variable for a set of design elements:

  Nodes [MAP ON *element*: *element.amountOfAccesses*]

  However, maps can be used to apply arbitrary functions (specified as expressions) on a set of elements. In the following example, an arithmetic function is applied to the same set as used in the previous example. This example also illustrates the use of multiple set operations, which are applied from left to right:

  Nodes [**MAP ON** *element* : *element.amountOfAccesses* ;
  MAP ON *element* : *element / 5*]

This example maps all nodes on their *amountOfAccesses* tracking variable (the first MAP ON set operation), and subsequently takes 20% of each value (the second MAP ON set operation). The result is a set of values.

Finally, set expressions are used to calculate arbitrary sets. The basic set theory operators (union, intersection and set difference) and the equality set comparator are supported. On (two) sets of numbers, element-wise addition, subtraction, multiplication or division is supported. Enumeration sets are defined using curly brackets **{ }**:

```
<setExpression>      ::=  <setComparand> (<setComparator> <setComparand>)*

<setComparand>       ::= <setTerm> (<setAdder> <setTerm>)*

<setTerm>            ::= <setPrimitive> (<setMultiplier> <setPrimitive>)*

<setPrimitive>       ::= <set>
<setPrimitive>       ::= '('<setExpression>')'
<setPrimitive>       ::= <setExpression> '['<setOperation> (';' <setOperation>)*']'

<set>   ::=
              <nativeSet> |
                    <setCreator> |
                    '{' <expression>  [',' <expression>]* '}'

<setComparator> ::= '='
<setAdder>      ::=  'DIFFERENCE' | '+' | '-'
<setMultiplier> ::= 'UNION' | 'INTERSECT' | '*' | '/'
```

Some examples of set expressions include:

{*link1*, *link2*} ;
{node1, node2, node3} UNION *ConsideredNodes* DIFFERENCE *RestrictedNodes* ;
Nodes UNION {node1, node2, node3} [**MAP ON** *element*: *element*.amountOfAccesses];
chunksFromNode(*node1*) [**FILTER ON** *element* : *element*.amountOfAccesses[*node2*] >
element.amountOfAccesses[*node1*]] ;

The next section explains native operations and functions.

5.4.8 Native Operations and Functions

The native operations allow expressing basic adaptive behaviour in ASL and correspond with the basic model transformation operations defined for the Navigation and Site Structure Model (see section 5.3). Their BNF specification is as follows:

```
<nativeOperation> ::=
                    'deleteNode ('<expression>')' |
                    'addNode ('<expression>')' |

                    'connect ('<expression>', '<expression>')' |
                    'disconnect ('<expression>', '<expression>')' |

                    'deleteLink ('<expression>', '<expression>' , '<expression>')' |
                    'addLink ('<expression>', '<expression>', '<expression>')'
                           [as <reference>] |

                    'addPage('<expression>')' |
                    'deleteNodeFromPage('<expression>', '<expression>')' |
                    'addNodeToPage('<expression>', '<expression>')' |

                    'alert('(<expression> | <string> )* ')'
```

As each of these native operations can be one-to-one mapped to a basic model transformations (see section 5.3) defined for the formal model, their definition will not be repeated here. Only the add- and deleteLink may require some extra explanation: the first parameter denotes one of four link types: 'processLogic', 'structural', 'navigationAid' or 'semantic'; the second and third parameter are respectively the source and target of the link. Furthermore, when adding a link the designer can uniquely identify the link giving it a *name* (reference). This name can later be used to uniquely identify the link. One operation, 'alert', has been added and is not map able on a basic model transformation. *Alert* does not specify any adaptive action, yet it can be used to specify an alert (expressed in the form of an expressions or a string) about something to the designer. The alert is simply used for monitoring purposes. Based on the alert, the designer may decide whether he will adapt the web site. Alerting the designer is possible in various ways (e.g. sending an email, output to a log file, etc), and is left to the implementation.

Native functions allow testing basic properties of the Navigation or Site Structure Model, or allow returning particular elements (e.g. the root of an audience track or the audience track of a given node). Where native operations do not return a value (they only cause a side-effect), native functions return a value, which can subsequently be used in an expression. Native functions returning boolean values end with an '?'.

```
<nativeFunction> ::=
        'root('<expression>')'   |   'audienceClass('<expression>')   |
        'audienceTrack('<expression>')   |   'currentAudienceClass()'   |
        'inAudienceTrack?   ('<expression>','   <expression>')'   |
        'linked?('<expression>','   <expression>','   <expression>')'   |
        'project('<setExpression>','  <expression>')'  |  'target('<expression>')'  |
        'source('<expression>')' | 'linktype('<expression>')'
```

The native functions are explained next:

- **root**: takes as input an audience track, and returns the root of this audience track
- **audienceClass**: takes as input a node, and returns the audience class of the audience track this node belongs to
- **audienceTrack**: takes as input an audience class, and returns the audience track for this audience class
- **inAudienceTrack?**: takes as input a node, a chunk or a page (first parameter) and an audience track, and returns true/false if the node, chunk or page is / is not contained in the audience track
- **linked?**: takes as input a link type (i.e. the first expression parameter) and two nodes, and returns true if there is a link of the specified type from the first node to the second node, false otherwise
- **source**: takes as input a link, and returns the source node of the link
- **target**: takes as input a link, and returns the target node of the link
- **currentAudienceClass**: takes no input, but returns the audience class of the current user.[66] This function can only be used in a monitor statement, which' evaluation is caused by the occurrence of a user event.
- **project**: takes as input a set (specified by a set expression), and a number (specified by an expression), and returns the element at the specified index of the (ordered) set. As described in section 5.4.1 (ASL General Overview), the project function is the glue between the two pillars of ASL: expressions and set expressions.

ASL provides support for operators on (ordered) sets, and statistical operators, as these may be useful when specifying the condition part of an adaptation rule. The basic ordered set operators are: *(set) cardinality and set indexing*. The standard statistical operators supported are: *minimum*, *maximum*, *average*, *range, median, middle, median absolute deviation* and *standard deviation.* Other statistical operators can be easily added.

```
<statisticalOperators> ::=
        'average('< setExpression >')' |
```

[66] How the audience class of the current user can be determined will be explained in the next chapter, section 6.3 (Validation of Audience Driven Navigation Structure).

‘max(’<setExpression>’)’ |
‘min(’<setExpression>’)’ |
‘range(’< setExpression >’)’ |
‘mad(’<setExpression>’)’ |
‘median(’< setExpression >’)’ |
‘middle(’< setExpression >’)’ |
‘stdev(’<setExpression>’)’ |

‘length(‘<setExpression>’)’ |

In the next section, the remaining (low level) BNF production rules for ASL are given.

5.4.9 Remaining BNF production rules

The remaining BNF production rules are discussed in this section. They consist mainly of low level BNF rules specifying the basic data types (i.e. Booleans, strings, numbers):

<boolean> ::= **‘false’** | **‘true’**

<string> ::= ‘”’ (<letter> | <digit>)* <letter> (<letter> | <digit>)* ’”’

<letter> ::= < nonCapitalizedLetter> | <capitalizedLetter>

<nonCapitalizedLetter> ::= ‘**a**’ | ‘**b**’ | ‘**c**’ | ‘**d**’ | ‘**e**’ | ‘**f**’ | ‘**g**’ | ‘**h**’ | ‘**i**’ | ‘**j**’ | ‘**k**’ | ‘**l**’ | ‘**m**’ | ‘**n**’ | ‘**o**’ | ‘**p**’ | ‘**q**’ | ‘**r**’ | ‘**s**’ | ‘**t**’ | ‘**u**’ | ‘**v**’ | ‘**w**’ | ‘**x**’ | ‘**y**’ | ‘**z**’

<capitalizedLetter> ::= ‘**A**’ | ‘**B**’ | ‘**C**’ | ‘**D**’ | ‘**E**’ | ‘**F**’ | ‘**G**’ | ‘**H**’ | ‘**I**’ | ‘**J**’ | ‘**K**’ | ‘**L**’ | ‘**M**’ | ‘**N**’ | ‘**O**’ | ‘**P**’ | ‘**Q**’ | ‘**R**’ | ‘**S**’ | ‘**T**’ | ‘**U**’ | ‘**V**’ | ‘**W**’ | ‘**X**’ | ‘**Y**’ | ‘**Z**’

<number> ::= <digit> <digit>*
<digit> ::= ‘**0**’ | ‘**1**’ | ‘**2**’ | ‘**3**’ | ‘**4**’ | ‘**5**’ | ‘**6**’ | ‘**7**’ | ‘**8**’ | ‘**9**’

Chapter 6: Adaptation Strategies

"Sire, je n'avais pas besoin de cette hypothese"
(Pierre Laplace, in response to Napoleon's asking why he had not once mentioned god in his work)

The previous chapter presented the necessary additions to WSDM to be able to specify adaptive behaviour. A formal Web site Overlay Model was introduced, allowing the designer to specify storage for relevant information about the (use of the) different design elements. Basic operations were provided to allow the designer to specify how to collect this information at run time. Subsequently, basic model transformation operations were defined on the Navigation Model and the Site Structure Model, to allow the designer to specify how to alter the design models. Finally, a high level Adaptation Specification Language (ASL) provides the designer with an easy mechanism to specify indeed, at design time, which runtime adaptation is allowed.

In this chapter, some examples of adaptation strategies (specifications of which adaptive behaviour needs to be performed) are explained intuitively. Subsequently they are expressed using ASL. For each strategy, both the information gathering rules (i.e. addMonitor-statements) and the adaptation rules (i.e. the rules transforming the design models) will be given. As explained in chapter 2, section 2.4 it is the aim of this dissertation to provide 'optimization' (as called by Perkowitz and Etzioni in [Perkowitz and Etzioni, 1997a], i.e. improvement of web site for all users based on the information gathered (from all users).

The remainder of this chapter is structured as follows. Section 6.1 describes promotion and demotion of nodes in the Navigation Model. Section 6.2 presents re-ordering of sequentially linked nodes in the Navigation Model. Finally, section 6.3 describes how to validate the design of an audience driven web site, and how to correct for detected flaws. For each section, the principles behind the adaptation strategy, an example and the ASL specification is given.

6.1 Promotion and demotion

Promotion and demotion was first briefly mentioned in [Perkowitz and Etzioni, 1997b]. In the framework of WSDM it was presented and elaborated in an initial version of ASL by the author in [Casteleyn et al, 2003]. In the following subsections, for both promotion and demotion, the principle is discussed (intuitively), followed by an example, which is subsequently expressed in ASL.

6.1.1 Promotion

Principle

Promotion of a node makes the node easier to find by moving it closer to the root (e.g. homepage) of the web site. Applied to the audience driven approach of WSDM, promotion of a node can be done by moving a node closer to the root of the *audience track* (instead of the root of the web site) of which the particular node is a part of. Thereby respecting the fact that visitors from other audience classes should not be bothered with information irrelevant for them[67], but yet an improvement is reached for the relevant users.

Figure 6.1 shows the principle of promotion of a node. Promotion is based on the popularity of the node (i.e. amount of accesses), and possibly also on the access path used to find the node (i.e. a node may be reachable by means of more than one path). Promotion is based on the guideline "the more popular the node, the closer to the root. Please note, that for the type of promotion considered here, all original links to the node are maintained.

[67] In case these other users actually need the information, a design flaw has been made since the information contained in the particular node *was* of interest for the other users as well. How to tackle this problem is described in section 6.3 (Validation of an Audience Driven Navigation Structure)

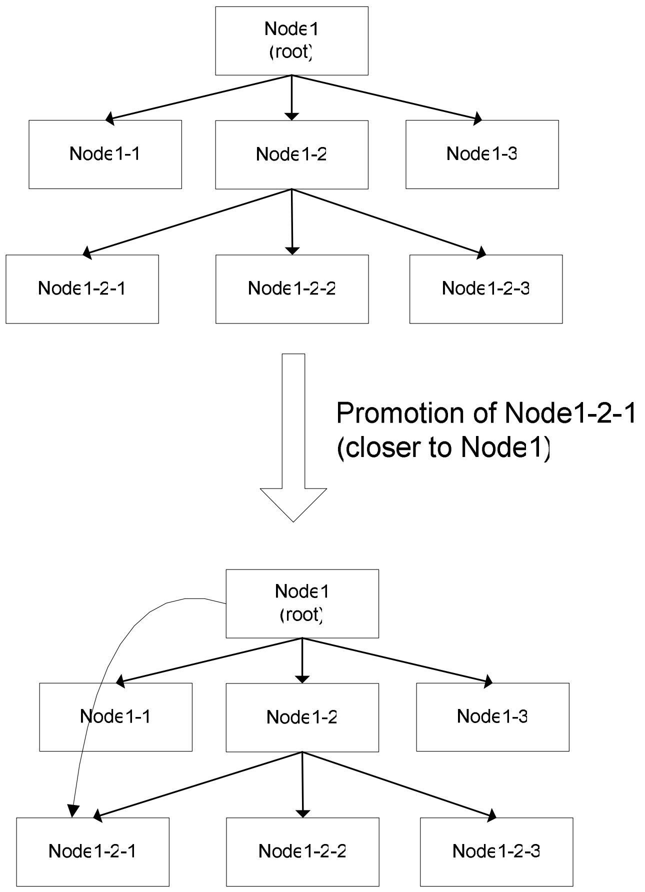


Figure 6-1 Promotion of Node 1-2-1 (closer to root Node1)

To further explain promotion, an example is elaborated in the next sub section; the ASL rules to specify promotion are stated in the following sub section.

Example

To illustrate the principle of promotion, consider the example given in figure 6.2. The example is a web site for a telephone company, and figure 6.2 shows the (overall) navigation model of the web site. Not to overload the figure, only the (simplified) navigation structure for the Personal User audience track is specified in detail; other tracks are not elaborate. Consistent with the graphical notation introduced in Chapter 3, section 3.4.2 (Navigational Design) nodes are represented by rectangles, chunks by rounded rectangles, (structural) links by arrows and the connections between chunks and nodes by lines. A double lined rectangle is used for the root of a track or the root of the web site (in this example the node 'Visitor Track' is the root of the web site).

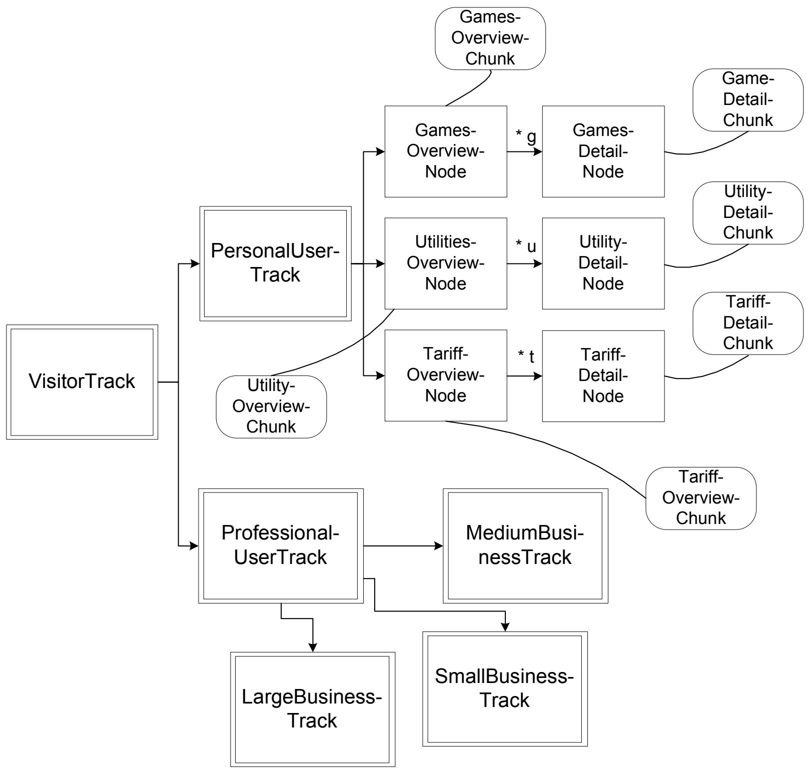


Figure 6-2 Simplified Example Navigation Model for a Telephone Company

The Personal User audience track consists of a navigation choice from the root, to consult an overview of tariffs, games and utilities. For each of these three choices, a general overview showing for each game/utility/tariff a short description and some relevant information is given. From there the user can followed a link to the actual tariff, game or utility[68]. At design time, the designer does not know which tariff, game or utility will be most popular. He might want to *promote* the most popular item of each category to (be available from) the root of the navigation track for these users. Moreover, suppose he also would like the most popular item measured over all categories to appear on the homepage (i.e. the node visitor track), as being interesting for *all* users. ASL provides the designer with the possibility to specify this adaptive behaviour. How this is done is described in the next sub section.

ASL specification

Promotion of nodes is here based on their popularity: the most popular nodes, i.e. the nodes with the highest amount of accesses, are promoted. ASL allows specifying for which nodes the amount of accesses needs to be measures, and how this should be done (i.e. per session, per load, per click). For the telephone company example from the previous sub section, the accesses to each individual tariff, game and utility need to be tracked. As tracking the amount of accesses to a certain set of design element will be needed frequently when devising an adaptation strategy, a script is specified implementing this behaviour. This script can subsequently be re-used in other adaptation strategies:

[68] Note how, for example in case of games, a parameter *g is passed from the GamesOverview chunk connected to the GamesOverview Node to the GameDetail Chunk connected to the GameDetailNode. In this way, we make sure the right game is shown when the user elects to view the details of a certain game.

```
script trackAmountOfAccesses(Set) :
    forEach element in Set
    begin
        addTrackingVariable element.amountOfAccesses ;
        monitor load on element do
        element.amountOfAccesses := element.amountOfAccesses + 1
    end
```

Intuitively, the for-each rule in this script states that a tracking variable *amountOfAccesses* is declared (i.e. addTrackingVariable) and attached to each element of the given set. Furthermore, load events on the elements (i.e. for all users and for all sessions) will give rise to the increment of the *amountOfAccesses* tracking variable of that particular element.

Using the script, the information gathering rule to support promotion as described for the telephone example is given as follows:

```
call trackAmountOfAccesses(ALL TariffDetailNode);
call trackAmountOfAccesses(ALL GameDetailNode);
call trackAmountOfAccesses(ALL UtilityDetailNode);
```

Note that the ALL keyword is used to obtain a set of all instances of a particular (conceptual) node. In this case, all concrete Tariff-, Game- and UtilityDetailNodes will be tracked for (runtime) usage.

The actual promotion in this case consists of linking the most popular tariff/game/utility to the root of the Personal User audience track. First, a script implementing the general principal of promotion is defined. In the definition stated here, the original link(s) to the promoted node are kept. Furthermore, as it concerns a shortcut link for the users to certain popular items, and not a structural change in the navigation organization, a navigation aid type of link is chosen. Alternative promotion strategies can be defined. The promotion script is specified as follows in ASL:

```
script promoteNode(Set, promoteTo) :
begin
    let promoteNodeMaxAccesses be
    max(Set [MAP on element: element.amountOfAccesses]);

    forEach node in Set :
        if  node.amountOfAccesses = promoteNodeMaxAccesses
        then addLink (navigationAid, promoteTo, node)
end
```

Using the promotion script, the actual promotion for the telephone company example is stated as follows:

```
call promoteNode(ALL TariffDetailNode, PersonalUserTrackNode);
call promoteNode(ALL GameDetailNode, PersonalUserTrackNode);
call promoteNode(ALL UtilityDetailNode, PersonalUserTrackNode);
```

Each rule specifies that, for each category (e.g. tariff, game, and utility) a link to the most popular of respectively the individualTariffNode, individualGameNode, individualUtilityNode should be added to the root of the personal user audience track.

Next to these promotions, the designer might want to promote the overall most popular tariff, game or utility to the homepage, offering this item to all users (i.e. not only to the user of the personal user audience track). The ASL rule(s) specifying this behaviour is as follows:

```
call promoteNode(
        (ALL TariffDetailNode) UNION
        (ALL GameDetailNode) UNION
        (ALL UtilityDetailNode), homePageNode);
```

Intuitively, the combined set of all concrete tariff, game, and utility nodes is used as a parameter to select the most frequently visited node; the homePageNode is given as a second parameter to denote where this overall most popular node should be promoted to (i.e. where a link to this node should be added).

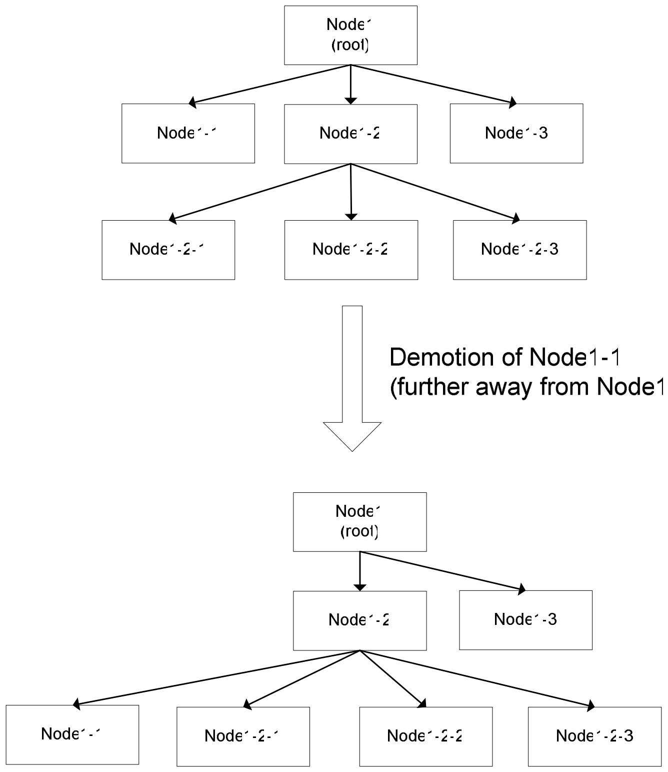


Figure 6-3 Demotion of Node1-1 (away from root Node1)

6.1.2 Demotion

Principle

Demotion, in contrast to promotion, moves a node that is conceived to be *less popular*, further away from the root (e.g. homepage) of the web site. Applied to the audience driven approach of WSDM, demotion of a node more particularly is done by moving a node further away from the root of the *audience track* (instead of the root of the web site) of which the particular node is a part of. In the particular case that the node being demoted is the root of an audience track (and its siblings are roots of other audience tracks, which is commonly the case in audience driven design), the particular audience track will be presented as a sub track of the root of the audience track to which it has been demoted. The example described in the next sub section covers this particular case.

Figure 6.3 shows the general principle of demotion of a node, in the context of WSDM framework. Similar as for promotion, demotion is based on (low) popularity of nodes. However, caution needs to be taken not to demote a node away from the logical place for the visitor to find it. It is the responsibility of the designer not to specify undesirable demotion rules.

Example

To illustrate the principle of demotion, consider the example given in figure 6.4. This figure shows a simplified (overall) navigation model for a university web site and is inspired by real life examples of university web sites applying the audience driven approach (e.g. Vrije Universiteit Brussel, Oxford University, Cambridge University, etc.).

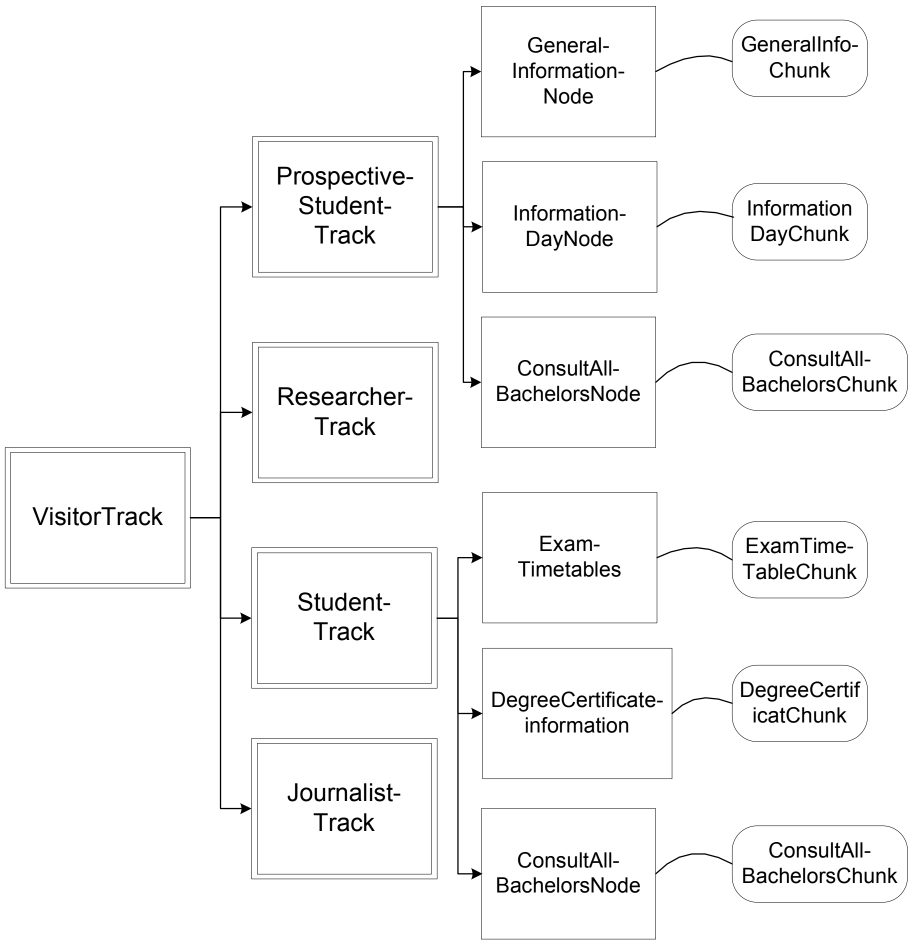


Figure 6-4 Simplified Example Navigation Model for a University Web site

From the Navigation Model, we observe that the Visitor Track node (represented on the homepage) is the starting point of navigation offering navigation paths to each audience track[69]. Not to overload the figure, only the (simplified) navigation structure for the audience tracks relevant for this example have been elaborated in more detail: the Prospective Student and the Student Track; the other tracks are not elaborated. These two Audience Tracks contain some common information (in this example simplified to one shared object chunk: consulting all bachelor programs[70]) and some different information (i.e. exam and degree certificate information for current students, and general information and information on information days for prospective students. The two audience classes cannot be modelled as one being a subclass of the other because the information required by neither audience is a subset of the information required by the other audience. At design time, the designer does not know if a lot of prospective students will visit the web site. Thus, he doesn't know if his choice to specifically accommodating prospective students (by offering them a direct link to a specific navigation track on the homepage) is justified. If it turns out that the use of the specialized audience track for prospective students is low, he might want to limit the navigation choices for all users on the homepage by locating the information for prospective students in the student audience track. ASL provides the designer with the possibility to specify this adaptive behaviour. How this is done is described in the next sub section.

ASL specification

Demotion of nodes is based on their (un-)popularity: unpopular nodes can be moved further away from the root, aiming not to overload the users with navigation choices to information that is rarely accessed and is relevant only for few users.

In the example of figure 6.4, the designer seeks to express that if only few users access the prospective student audience track, he prefer to restrict the navigation choices on the homepage and offer the prospective student audience track as a sub track of the student audience track. It is at the discretion of the designer to specify what exactly is meant by *few users*. In this example, we'll use the rigid threshold that if, for all sessions to either the student or prospective student audience track, there is a ratio 1/99 in favour of the student audience track, we'll demote the prospective student audience track from the root to the (root of the) student audience track (i.e. making the prospective student audience track a sub track of the student audience track).

Specifying exactly which user access information needs to be gathered is done using the following ASL information gathering rules[71]:

```
foreach node in {ProspecitiveStudentTrack, StudentTrack}:
  begin
      addTrackingVariable node.amountOfSessions ;
      monitor load on node
      do node.amountOfSessions := node.amountOfSessions + 1 per session
  end
```

[69] Only four audience tracks are represented here; real university web sites may also offer other audience classes (e.g. alumni, visitors, media, etc.)
[70] Other shared requirements, resulting in information to be available for both audience classes, are omitted from this example.
[71] Note that we cannot re-use the information gathering script defined in section 6.1.1.3 (ASL Specification; Promotion), as here the amount of sessions in which a node was accessed is monitored, and not the total amount of accesses. A script similar to section 6.1.1.3 can be defined here, but for illustrative reasons we elect in this example not to use a script.

Intuitively, this for-each rule first declares a tracking variable *amountOfSessions* for the root of both the prospective student and student audience track (i.e. the ProspecitiveStudentTrack and StudentTrack nodes), and subsequently states that their value should be updated once per session when the particular node is loaded.

The actual demotion for this example is specified by the following ASL rule:

```
if ProspectiveStudentTrack.amountOfSessions/
ProspectiveStudentTrack.amountOfSessions < 1/99
then
  begin
        deleteLink(structural, VisitorTrack, ProspectiveStudentTrack);
        addLink(structural, StudentTrack, ProspectiveStudentTrack);
        deleteLink
          (structural, ProspectiveStudentTrack, ConsultAllBachelorsNode) ;
        addLink(navigationAid, ProspectiveStudentTrack, StudentTrack)
  end
```

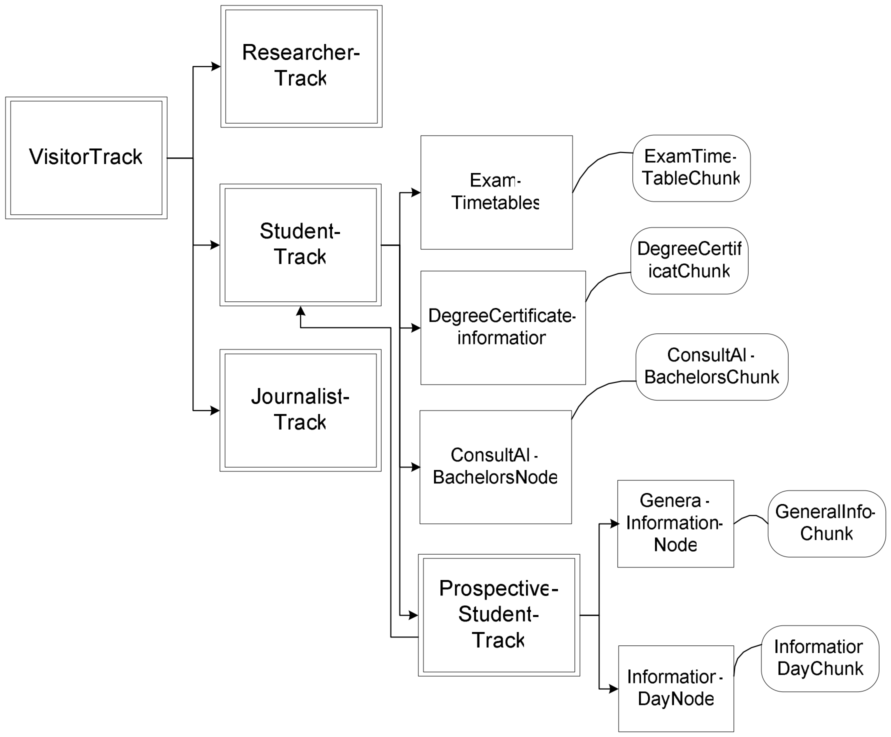


Figure 6-5 Navigation Model for the University Example after Promotion

Intuitively, demotion in our example consists of first unlinking the ProspectiveStudentTrack node from the VisitorTrack node (i.e. the root of the web site), and linking it from StudentTrack node. The prospective student track will then be presented as a sub track of the student track. The duplicated information (all bachelor programs) can be removed from the prospective student audience track (i.e. the sub track) as it is already available in the parent track (i.e. the student track). However, to (still) accommodate prospective students who have bookmarked their track (and thus will not pass through the student track), a link back to the root of the student track is added as a navigational aid for prospective students. The resulting Navigation Model for the university example after demotion is shown in figure 6.5.

6.2 Re-ordering sequential information (gathering)

6.2.1 Principle

Re-ordering a set of linear-linked nodes is aimed at moving bottlenecks (i.e. nodes in the linear order where a lot of users abandon the sequence) to the end of the sequence. Re-ordering of nodes can only be done when there are no pre-requisite relations between the nodes, i.e. it is not vital that the user has visited one node before s/he is allowed to see another[72].

Re-ordering nodes may for example be useful in an information gathering process, where the user is sequentially asked for information (e.g. personal, work related, financial, …) by means of input forms. A bottleneck in the sequence appears when e.g. the user doesn't know what he should enter, he doesn't have the information at hand or he is not willing to provide the information and as a consequence will quit the process. This prevents that the following nodes are visited by the user, and thus prevents the web site owner to gather the information requested in these nodes. Because the order is not important, a re-ordering of the nodes can offer a solution. An example of re-ordering of a sequential information gathering process is presented in the next sub section.

6.2.2 Example

Web sites of commercial organization are commonly interested in gathering as much information as possible from their visitors in order to better accommodate them on their web site and thereby increase the company's success in accomplishing their business goals. Commonly, these web sites use some sort of subscription mechanism, providing registered users with some benefits and extra options, compared to non-registered users. When registering, the user is sequentially asked for a variety of information by means of input forms. In such a case, it is the (implicit) goal of the web site owner to gather as much information as possible.

Users on the other hand are often reluctant to provide information about themselves. They may be willing to provide certain information, but unwilling to give other. When asked for information they do not want to give, some users might decide to abandon the information gathering process (i.e. the registration). If a lot of users don't want to give some particular type of information and subsequently abandon the registration, a bottleneck in the sequential information gathering process occurs, and the information beyond the bottleneck cannot be gathered for those users (although they might have been willing to provide it).

Consider the example of a bookshop, where users are offered the possibility to register. Once registered and logged in, users are offered some extra features (e.g. recommendations, a newsletter, promotions, etc). The customization, such as offering the registered users recommendations, is not the subject of this example. Instead we focus on the registration process itself, and the detection of bottlenecks and possible re-ordering of nodes *for all users*. The registration process for the bookshop is a sequential information gathering process, consisting of four steps. In the first step, the user is asked for a login-name, a password and an email address. In the second step, some personal information, i.e. name, email address, date of birth, sex, height, education, marriage status, children

[72] In WSDM, the task models created during the Task Modelling Phase express, by means of a temporal relation, when a sequence of subtasks can be performed concurrently. As the task models are taken as input to construct the internal (navigation) structure of an audience track, the existence of sequential nodes resulting from concurrent tasks indicate information that do not have prerequisites.

(and some information about them) is asked. The third step inquires for hobbies, personal preferences in music, art, literary genres, housing situation, etc. Finally, the fourth step asks for payment information (e.g. credit card number, expire date, etc) and shipping address (in case the members want to order something). All information gathered in each step is useful for the bookstore (e.g. to adjust the range of products according to the customer profiles based on the gathered information). At design time, the designer does not know if and which information users will be reluctant to give. He might want to find out at which point a lot of users abandon the information gathering process, and if such a point is detected, change the order of the nodes in the sequence to move the bottleneck node towards the end of the sequence.

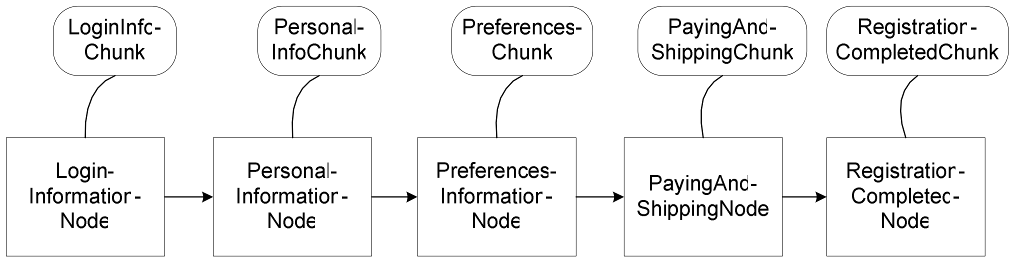


Figure 6-6 Example Information Gathering Process for a Bookstore

ASL provides the designer with the possibility to specify the detection of bottlenecks and subsequently, to specify how to re-order the information gathering process to move the bottleneck to the end of the sequence. How this is done is described in the next sub section.

6.2.3 ASL Specification

Re-ordering of nodes of a sequential information gathering process is based on the existence of a bottleneck. As the nodes are linear linked, calculating the difference between the amounts of accesses of two consecutive nodes indicates how many users abandoned the sequence at the first of the two nodes (i.e. they loaded the first node but never reached the next one; they thus didn't complete the input forms represented on the first node). For the bookstore example, the ASL information gathering rules are specified as follows:

> **let** Sequence **be** {LoginInformationNode, PersonalInformationNode,
> PreferencesInformationNode, PayingAndShippingNode,
> RegistrationCompletedNode};
>
> **call** trackAmountOfAccesses(*Sequence*);

Intuitively, first a set variable (e.g. Sequence) is declared which contains the nodes under consideration. Next, the script defined trackAmountOfAccesses in section 6.1.2.3 (ASL Specification; Promotion) is re-used to attach to each element of Sequence a tracking variable amountOfAccesses, which will be updated each time a user accesses (i.e. loads) the particular element.

Providing a heuristic *when* a bottleneck occurs and thus when nodes in a sequential path may be re-ordered (i.e. the condition for adaptation rules specifying the re-ordering) is the responsibility of the designer. In this example, we decided that a node is a bottleneck if the following condition holds:

- the amount of visitors abandoning the sequence on a particular node is higher than 50% of the total amount of visitors starting the sequence

When this condition holds, a bottleneck has been found, and will be moved towards the end of the path. As an aid, we first define two scripts, one that deletes a node from a path (given as a set) and one that adds a node in a path at a specified position:

```
script deleteNodeFromPath(Set, linkType, node) :
begin
        let linkedFrom be
                project(Set [FILTER ON element: linked?(element, node)], 1);
        let linksTo be
                project(Set [FILTER ON element: linked?(node, element)], 1);
        deleteLink(linkedFrom, node);
        deleteLink(node, linksTo)
        addLink(linkType, linkedFrom, linksTo);
        Set := Set / {node}
end
```

Intuitively, this script takes as input a set (representing a path), a link type and a node which will be deleted from the path. First, two variables are declared: the first one, *linkedFrom*, contains the node from the given path that links to the *node*; the second, *linksTo,* contains the node that is linked by the *node*. Subsequently, the links (*linkedFrom, node)* and *(node, linksTo)* are deleted, and the path (which has now become disconnected due to the elimination of *node*) is reconnected by adding a link (*linkedFrom, linkedTo*). Finally, the *node* is deleted from the given *Set*.

The second useful script inserts a node in a path at a certain specified position:

```
script insertNodeInPath(Set, linkType, node, position) :
begin
        deleteLink(linkType, project(Set, position - 1), project(Set, position));
        addLink(linkType, project(Set, position - 1), node);
        addLink(linkType, node, project(Set, position));
        Set := addElementAtIndex(Set, node, position);
end
```

Intuitively, this script disconnects the path where the *node* has to be added (i.e. the deleteLink operation), and subsequently reconnects the path by linking the *node* in between (i.e. the first addLink operation adds a link to the node, the second addLink operation adds a link from the node back to the path).

The actual adaptation, i.e. re-ordering the nodes when a bottleneck has been detected, is expressed by the following ASL specification:

```
(1) Foreach node1 in Sequence, node2 in Sequence
(2)     If (linked?(node1, node2)) AND
(3)     (node1.amountOfAccesses - node2.amountOfAccesses >
(4)     project(Sequence [MAP on element: element.amountOfAccesses], 1) / 2)
(5)      then
(6)             begin
(7)             call deleteNodeFromPath(Sequence, processLogic, node1);
(8)             call insertNodeInPath(Sequence, processLogic, node1, length(Sequence) – 1)
(9)             end
```

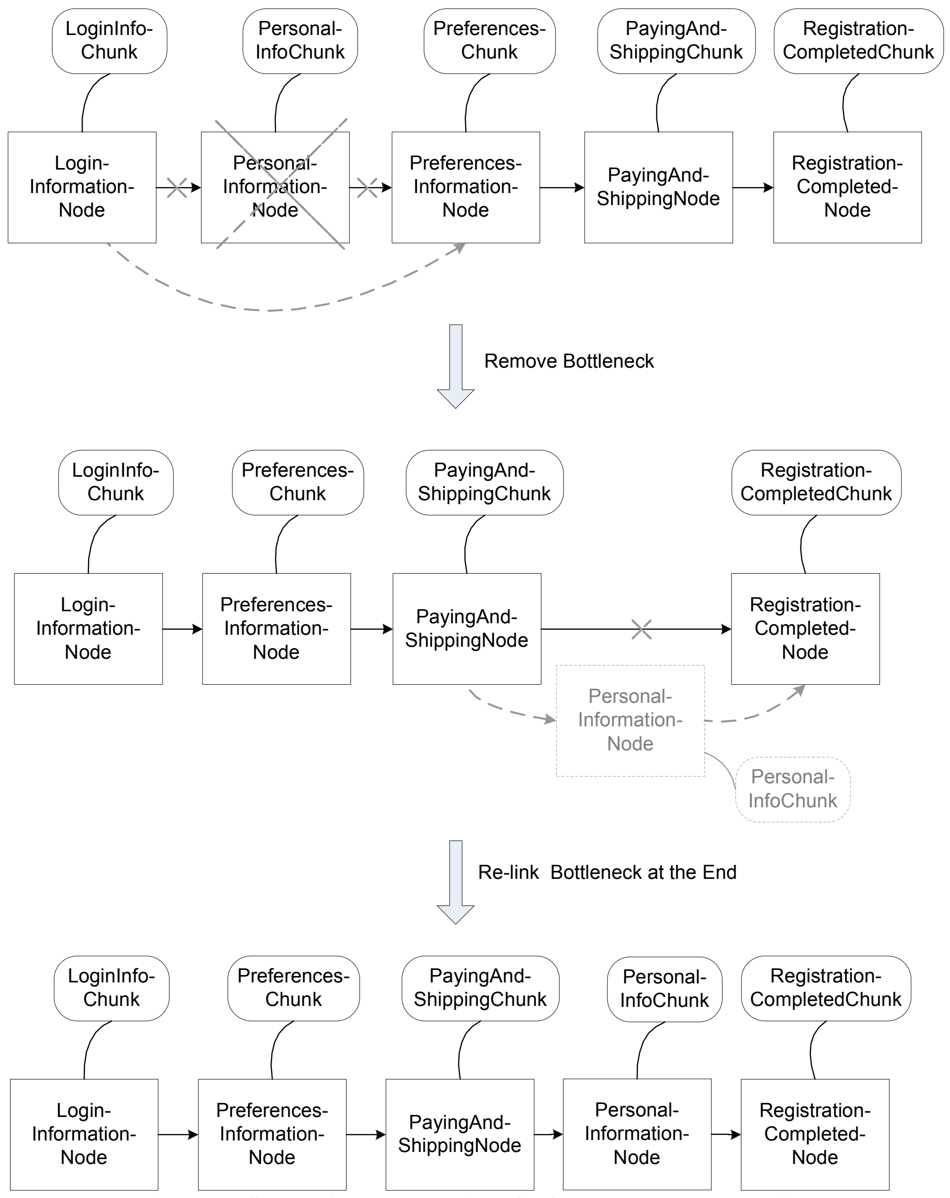

LoginInfo-Chunk
Personal-InfoChunk
Preferences-Chunk
PayingAnd-ShippingChunk
Registration-CompletedChunk
Login-Information-Node
Personal-Information-Node
Preferences-Information-Node
PayingAnd-ShippingNode
Registration-Completed-Node
Remove Bottleneck
Re-link Bottleneck at the End

Figure 6-7 Schematic Representation of a Concrete Node Re-ordering

Intuitively, this rule can be explained as follows:

- **For-each Part (line 1)**: each two nodes in the *Sequence* are iterated over
- **Condition Part (line 2 to 4):** for each two consecutively linked nodes (i.e. *node1* and *node2*, line 2) of the Sequence, the difference between the amount of accesses is calculated (3). This difference denotes the amount of users that have abandoned the information gathering process after having loaded the first node (i.e. they did not complete the forms represented by the first node). If this amount of users abandoning the sequence is higher than 50% (line 3 and 4), then the first node, *node1*, is detected to be a bottleneck.
- **Action part (line 6 to 9):** the adaptation consists of moving the node detected to be a bottleneck to the end of the sequence. This is done by first removing the bottleneck, and subsequently reconnecting it at the end of the path. More specifically, the bottleneck is added

as the penultimate node in the path. Note that the last node indicates that the registration process is finished, and thus should always remain the last node in the path.

To illustrate this rule, consider the situation where the amount of accesses for each node is given as follows:

• LoginInformationNode.amountOfAccesses	= 1000
• PersonalInformationNode.amountOfAccesses	= 950
• PreferencesInformationNode. amountOfAccesses	= 320
• PayingAndShippingNode.amountOfAccesses	= 300
• RegistrationCompletedNode.amountOfAccesses	= 298

The biggest difference between two consecutive nodes occurs between the PersonalInformationNode and the PreferencesInformationNode, indicating that most users abandon the information gathering sequence when they are asked for personal information. As the difference is 630 (i.e. 950 – 320), which is 63% of all users starting the sequence (i.e. 1000), the PersonalInformationNode is detected as a bottleneck and will be moved to the end of the sequence.

Figure <x> schematically shows this re-ordering process. The re-ordering is shown in two phases: first the bottleneck is removed, and subsequently the bottleneck is added at the end of the path. Changes to the original structure caused by adaptation statements are gray and in dashed line.

6.3 Validation of Audience Driven Navigation Structure

Validation and (automatic) correction of a (audience driven) web site design was first briefly described in the literature by the author in [Casteleyn et al, 2004b] and elaborated into more detail and described in the framework of WSDM (with ASL) in [Casteleyn et al, 2005].

In the following sub section, the principle(s) for validation of an audience driven web site design is discussed, followed by an example which is subsequently expressed in ASL.

6.3.1 Principle

6.3.1.1 Introduction

An audience driven design for web sites takes as an explicit starting point of the design the different target users, which are classified in audience classes according to their requirements: users with the same information and functional requirements belong to the same audience class. Subset relations between the requirements of audience classes give rise to a partial order relation between audience classes. Using these partial order relations, the audience classes are classified into an audience class hierarchy, where the top of the hierarchy is the visitor audience class, representing the requirements all visitors share. From the audience class hierarchy the main navigation structure for the web site is derived: for each target audience, a designated audience track is created, containing all and only the information and functionality the designer assessed to be relevant for these users. Concretely, for the visitors this results in links (usually on the homepage), each representing a different navigation path for a different kind of visitor (called audience track).

Although the audience driven design philosophy significantly reduces the amount of information the visitor needs to plough through, it also has as drawback that possibly some information needed by the user is not available in the audience track chosen by the user. This is a direct result of the fact that the assessment of requirements by the designer (at design time) is reflected in the final navigation structure of the site, i.e. incorrectly assessed requirements result in information/functionality being in a wrong navigation track. More specifically, some relevant information may be missing in a certain audience track but present in another, while other information present in that audience track may be superfluous (i.e. a requirement was in-correctly assigned to the audience class).

However, for the designer it is not always easy[73] to assess the different requirements of the different target audiences correctly. In the following subsections, we show how to correct for such possible (design) flaws using adaptation. We will show how to specify, at design time, how to detect both missing and superfluous information in a certain audience track. Subsequently, ASL adaptation rules are specified to correct the detected design flaws.

6.3.1.2 Missing and Superfluous Information

As was explained in the introduction, for the designer it may be very difficult, to correctly assess the different requirements of the different audience classes (from now on, the requirements originally assigned to an audience class will be called *native* requirements). On the one hand, missing requirements for a certain audience class result in the fact that the information/functionality related to this requirement is not accessible in the audience track of that audience class. On the other hand, a designer might also assign a certain requirement to an audience class, where actually this requirement is not relevant for that audience class. This results in superfluous information/functionality being presented in the audience track for that particular audience class.

According to the audience driven philosophy, a user should find *all* the information and functionality he needs in his particular audience track. Subsequently, if a user leaves his audience track to look for information elsewhere, then the information he is visiting outside of his audience track (from now on called information related to a *foreign* requirement) might be information that is actually missing in his own audience track. If such information outside the audience track is visited only by a few users of that audience track, those users might just be curious and browsing other tracks or they might just have miss-clicked. However, if a significant amount of users visits information related to a foreign requirement, then we have a good indication that that information is actually relevant for the current audience track as well.

When a user does not leave his audience track, some problems with the information in the track are still possible. Some information in the track might be accessed very few (compared to the other information), suggesting the information does not belong there in the first place. Combined with frequent access to that information originating from another audience track, it could point to the fact that that particular information has been put in the wrong audience track. Note that identifying information that is accessed few, does not necessarily imply that this information is superfluous. Some information might be not relevant to most users, but invaluable for a small amount of users, and thus considered necessary by the designer.

From the scenarios described above, we can determine the following three steps in identifying missing or superfluous information:

1. Determine to which audience class the current user belongs.
2. Determine if and which information the user visits, both within and outside his audience track.
3. Analyze the accumulated data to determine if the information within the audience track is relevant, or if visits to information outside the audience track are significant

The first two steps acquire the necessary information; the third step describes how this information should be used to detect missing or superfluous information.

[73] Target audiences for web sites are often more difficult to access and it is more difficult to perform standard requirements engineering techniques (e.g. questionnaires) compared to classical prospectus users of a standard application.

Determining to which audience track a user belongs depends on how the user enters the web site:

- **Access through the homepage:** due to the particular navigation structure of a web site obtained using an audience driven design, a user entering the web site through the homepage, is offered with a choice of audience tracks (i.e. sub-sites) for the different types of users the web site supports. It is thus safe to assume that, if a user is looking for certain information/functionality, he will choose the navigation track best suited for the role in which he is visiting the web site (hoping to find what he is looking for). Consequently, determining the audience class of this user can be done by monitoring his first clicks (of the session) to know which audience track he enters.
- **Direct access to a page:** if the user starts his session by directly accessing a particular page of the web site, he either bookmarked the page or came via a direct link in an external web site. In both cases, the user is interested in the particular page, so he can be considered as a member of the relevant audience track.

In the second step, the data about which information a user visits is stored. As we want to keep track of which information is visited outside a particular audience track, and relate this information to the frequency of visits inside the track, we cannot just store the (total) number of accesses to every piece of information. Instead, we need to store the number of visits to each piece of information relative to each audience class. This data can be conveniently stored in a matrix, which we will call the *information access matrix*. Rows of the matrix represent the different pieces of information (i.e. the *nodes* in WSDM, which groups object chunks to be represented on the same page), while the columns represent the different audience classes. Each entry (i,j) in the matrix represents how many times the particular node(*i*) has been accessed by a member of the audience class *j*.

In pseudo code, the algorithm to populate the matrix is as follows:

```
WHEN session.start THEN
    Determine Audience Class j
    UNTIL session.end DO
        FOREACH node(i) visited
        DO M(i,j)++
        END
    END
```

Summarized, whenever a browsing session starts, the audience class of the user is determined. According to that audience class, the amount of accesses to each node (both inside and outside the audience track) is updated, by increasing the value of the correct cell in the matrix.

Over time[74], the matrix contains a good summary of the amount of accesses to the different nodes (containing information) for each audience class. The next sub section describes how the information access matrix can be used to identify missing or superfluous information in an audience track.

Identifying Missing or Superfluous Information

To identify missing or superfluous information, known statistical techniques are applied on the Information Access Matrix. How this is done exactly is shown in the next subsections.

Missing Information

[74] Here, we do not go into deeper detail on *when* (the first) evaluation of the matrix can be performed. Suffices to say a representative amount of visitors should have performed browsing session(s).

To determine if the amount of accesses from a certain audience class to foreign information is significant, we can rely on known statistical techniques. In statistics, the problem of determining if a certain value fits well in a given (relatively small)[75] sample data set is solved using basic statistical measures for central tendency (e.g. mean, median, trimmed mean, ..) and standard measures of spread (e.g. variance, standard deviation, median absolute deviation, ..).

Central tendency is a measure of the middle of a data set. The two best know central tendencies are the mean value, and the median. As the distribution of our dataset is unpredictable and we do not want our calculations influenced by (a few) extreme values (e.g. nodes that are highly popular), we prefer a measure that is robust (i.e. not influenced by extremes in our data set). With this in mind, the median is the best choice for further calculations. As a measure of spread the median absolute deviation (MAD) is chosen. This measure is less influenced by outliers (compared to other measures) and has robustness of validity (i.e. the underlying distribution doesn't influence reliability too much). The following formula is used to calculate absolute median deviation:

- $MAD = median\ (|x_i - x_m|)$

  where x_i is each of the values of the data set, and x_m is the median of the data set.

As the spread denotes how far, on average, the values of the dataset are from the middle value, we can conclude that most of the values of the dataset lie within the distance of the spread from the middle value (for more exact estimates of how much elements of the data set lie within this interval, we refer to [DeGroot and Schervish, 2001]). Consequently, external values (i.e. values outside the given dataset) *fit well* in the given dataset, if they lie within that range.

Applied to the problem of finding information outside the audience track of a particular audience class that is actually relevant for that particular audience class, we adopt the following method:

For a given audience class/track:

1. Calculate median and MAD for the set of number of accesses to information resulted from *native* requirements and calculate the threshold (median – MAD)
2. For all *foreign* nodes, verify if the amount of accesses is greater than the calculated threshold. If this is the case, we have an indication that that particular information is relevant to the audience class under investigation

Note that only the lower limit of the range is used as information resulting from a foreign requirement track that is accessed more than (median + MAD) is off course also relevant for the current audience class.

Superfluous Information

As for identifying missing information, we also rely on known statistical techniques to determine when information is superfluous, i.e. when it is significantly less accessed compared to other information (within the same audience track). In statistics, the problem of finding an outlier in a data set is generally considered hard, especially when the dataset is small, and accurate information about it (e.g. distribution) is missing. Consequently, most existing tests are not usable here, because they only work on large datasets or assume a normally distributed dataset (e.g. Rosner's Test [Rosner, 1975], Dixon's test [Dixon, 1950], Grubbs's test [Grubbs, 1969] or others [Barnett and Lewis, 1994] [DeGroot and Schervish, 2001] [Hawkins, 1980] [Stefansky, 1972]).

However, for our purposes, detecting significantly low values (but not per se outliers as they are defined in statistics) in our dataset is already sufficient. To identify these low values, we will use a

[75] For large datasets, more advanced methods are available (e.g. clustering techniques).

double strategy: look for values which lie both far from their (greater) neighbor, and also lie far from the middle value of the dataset.

To determine which value lies far from its neighbor, we take the ordered dataset, and calculate the distances between each 2 consecutive values. These distances give us a new dataset, for which we calculate mean and standard deviation[76] (abbreviated as *std*). Calculating the interval [(mean–std) (mean+std)] gives us the range in which most of the distances (i.e. 50% for normally distributed data) lie. Distances above the (mean+std) are thus relatively big, compared to the other distances, and we have identified two points which are relatively far from each other.

To determine which point lies far from the middle value, we apply the same technique as in the previous section: calculate the median and MAD for the given dataset. Values below the threshold (median – MAD) can be labeled as being far from the middle value.

6.3.1.3 Correcting Missing and Superfluous Information

Having identified missing or superfluous information in a certain audience track, the structure of the web site can be adapted (automatically) to reflect the detected deficiencies.

Missing Information

There are several ways to correct an audience driven navigation model for missing information in a certain audience track. The most obvious way is to add a link to the missing information in the root of the audience track in which is it missing (using a navigation aid link). In this way, it is immediately available to the particular audience class. In particular cases, the designer might also decide to add a link to the information at a more specific location in the audience track. Another possibility consists of duplicating the missing information and makes it available within the particular audience track in which it is missing (i.e. by adding a new node and connecting the duplicated information to this node). In this way, the conceptual navigation structure is effectively changed, but the audience driven design philosophy (i.e. the user should not be able to unwittingly ‘jump’ to another audience track) is respected. In case the same foreign information is detected to be missing for several audience classes, the particular information can be make available from the root of the web site. Finally, the navigation structure may be totally re-arranged. For example when two audience classes each contain information that is missing in the other, these two audience classes may be merged.

Superfluous Information

As mentioned in section 6.3.1.2, information identified as superfluous in a certain audience track does not necessarily need to be removed. Although visited only few times, it might still be valuable for (a small amount of) visitors. For example, information on “how to get there” at a university lab’s web site will presumably have relatively few visits, compared to other information (e.g. information on publications etc), as only a small part of the visitors of the site will need to visit the lab physically. However, the information on how to arrive to the lab is certainly necessary. Thus, the detection of superfluous information is rather considered as an alert to the web master, than something that requires (automatic) adaptation. If the web master indeed decides the information is not needed in that track, he can remove it.

[76] As this time, we want to detect high distances, we use mean and standard deviation, as they are more affected by the presence of extreme low or high values.

6.3.2 Example

To illustrate validation of an audience driven design, consider the example of (a simplified version of) the NASA web site[77] in figure 6.8. Not to overload the figure, only the Media&Press and the Educator audience tracks were elaborated. As can be seen from the navigation model, each audience track presents the user with some navigation choices. In most cases, following a particular link will bring the user to an overview list (e.g. "Press Kits"), presenting a list of the available items. When selecting a particular item, detailed information on this item is displayed (e.g. "Press Kit Detail"). Some nodes linked from the audience track root do not contain an overview list. Instead, upon following the link, the relevant information is immediately shown (e.g. "Press Contacts").

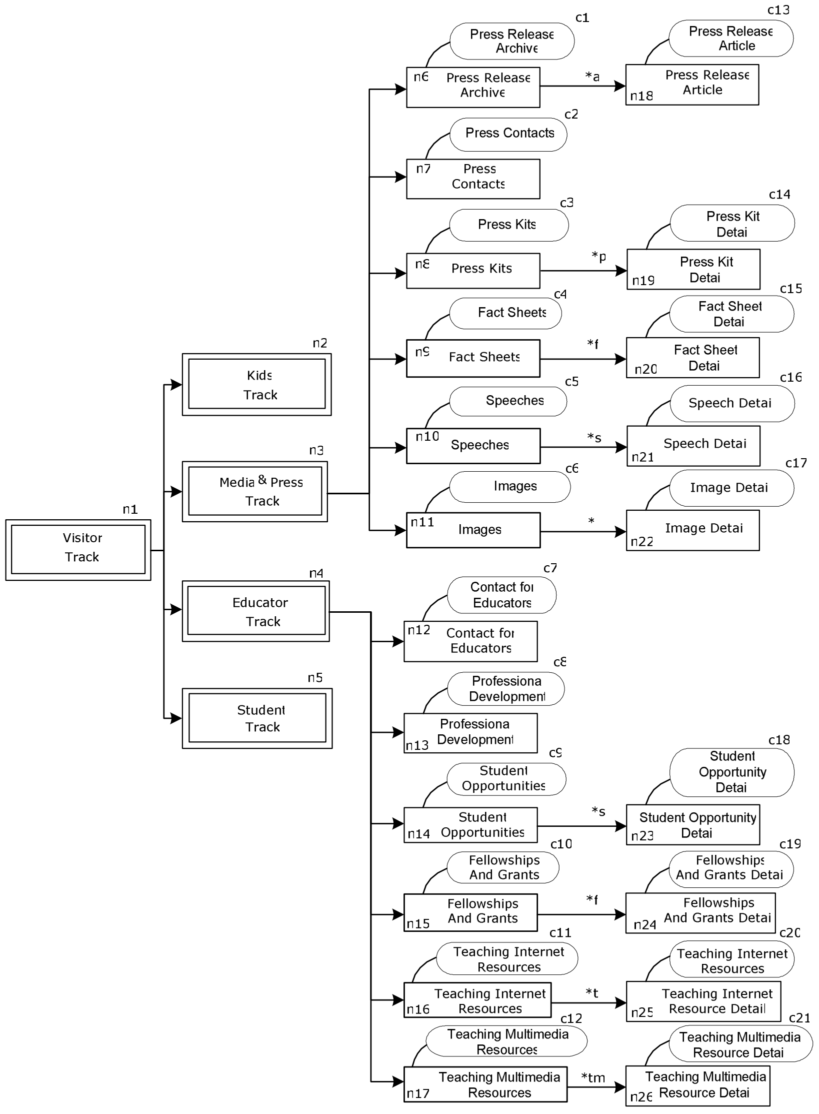

n1 Visitor Track
n2 Kids Track
n3 Media & Press Track
n4 Educator Track
n5 Student Track
n6 Press Release Archive
c1 Press Release Archive
n7 Press Contacts
c2 Press Contacts
n8 Press Kits
c3 Press Kits
n9 Fact Sheets
c4 Fact Sheets
n10 Speeches
c5 Speeches
n11 Images
c6 Images
n12 Contact for Educators
c7 Contact for Educators
n13 Professional Development
c8 Professional Development
n14 Student Opportunities
c9 Student Opportunities
n15 Fellowships And Grants
c10 Fellowships And Grants
n16 Teaching Internet Resources
c11 Teaching Internet Resources
n17 Teaching Multimedia Resources
c12 Teaching Multimedia Resources
*a
n18 Press Release Article
c13 Press Release Article
*p
n19 Press Kit Detail
c14 Press Kit Detail
*f
n20 Fact Sheet Detail
c15 Fact Sheet Detail
*s
n21 Speech Detail
c16 Speech Detail
*
n22 Image Detail
c17 Image Detail
*s
n23 Student Opportunity Detail
c18 Student Opportunity Detail
*f
n24 Fellowships And Grants Detail
c19 Fellowships And Grants Detail
*t
n25 Teaching Internet Resource Detail
c20 Teaching Internet Resources
*tm
n26 Teaching Multimedia Resource Detail
c21 Teaching Multimedia Resource Detail

Figure 6-8 Simplified Navigation Model for (part of) the NASA Web site

[77] http://www.nasa.gov/

An information access matrix for these tracks is shown in figure 6.9[78]. The columns represent the two audience classes considered in this example (Media & Press and Educators) and the rows represent the nodes for which we want to detect if their associated information (i.e. object chunk) is missing or superfluous in another audience track. Note that, for simplicity of the example, we are only considering the nodes immediately linked from the root of each audience track; we do not consider the detailed information represented by nodes n18, n19, n20, n21 and n22, n23, n24, n25 and n26[79].

		Media & Press	Educators
n6	Press Release Archive	**52**	**40**
n7	Press Contacts	**49**	**4**
n8	Press Kits	**31**	**10**
n9	Fact Sheets	**16**	**5**
n10	Speeches	**40**	**5**
n11	Images	**38**	**12**
n12	Contacts for educators	**0**	**56**
n13	Professional development	**5**	**50**
n14	Student opportunities	**1**	**30**
n15	Fellowships and grants	**0**	**10**
n16	Teaching Internet Resources	**3**	**20**
n17	Teaching Multimedia Resources	**2**	**15**

Figure 6-9 Information Access Matrix

For example, cell (1,1) shows that members of the audience track "Media & Press" have accessed 52 times "Press Release Archive" node (i.e. n6). Note that the first six nodes in the matrix are native for the Media & Press track (and foreign nodes for the Educator track), and the last six ones are native to the Educator track (and foreign to the Media & Press Track).

Missing Information

Let's now consider the NASA web site example and apply the technique described above to identify possible missing information in the Educators track. To do so, we consider the native nodes of the Educators audience track (n12 … n17), and determine if the amount of access to foreign nodes (n6 .. n11) is significant compared to the amount of access to native nodes. First, we calculate median and MAD for the native nodes as described above:

Data set (ordered): 10 15 20 30 50 56
Median: 25
MAD: 12.5

We are now ready to calculate the threshold that is the lower limit to decide if foreign information is relevant for the current audience track. In this example, the threshold is median – MAD = 25 – 12.5 = 12.5. The information of the *Media & Press* track with a number of accesses greater than that threshold is detected to be missing for the Educator track. This is the case for the *Press Release Archive* node (40 accesses), located in the *Media & Press* audience track. The adaptive action to correct this (design) flaw is specified in the next section, when stating the ASL specification for the NASA example.

Superfluous Information

[78] Fictitious data was used to populate the matrix in order to show the principles discussed.
[79] Note that users which entered the site through the homepage have to pass these (more) general overview nodes before they can access any detailed node. Gathering information for the overview nodes is thus also representative for the usage of the more detailed nodes.

Let's again consider the NASA web site example and apply the technique described above to identify possible superfluous information in the Media & Press track. To do so, we consider the nodes of the Media & Press audience track (n6 .. n11) , and determine if accesses to some of these node are significantly low (compared to the other native nodes).

Data set (ordered): 16 31 38 40 49 52

Median: 39
MAD: 9
Lower limit: 39 – 9 = 30

Data set of distances: 15 7 2 9 3
Mean: 7.2
Standard deviation: 4.7
Upper limit: 7.2 + 4.7 = 11.9

As the distance between the first and the second element of the original dataset is 15, which is larger than the threshold 11.9, we conclude that the first element lies significantly *far* from the next element. As this first element, namely 16 in the original dataset, lies below the threshold of 30, we can also conclude that it lies *far* from the middle. With both tests positive, we conclude that the value 16 is indeed significantly low compared to the other values: the information related to the requirement 'Fact Sheets' is detected as (possibly) superfluous for the audience class Media & Press.

6.3.3 ASL Specification

Detecting missing or superfluous nodes is based on the information access matrix (described in section 6.3.1.2). This matrix contains amount of accesses to the nodes considered, stored relative to the audience class of the current user. ASL allows to specify the information access matrix, using two information gathering rules; one to specify tracking variables for access to foreign nodes, and one for access to native nodes. To make the rules more readable, and to be able to re-use them, we specify two scripts implementing these two information gathering rules. The ASL information gathering script to specify tracking variables to store accesses to native nodes is as follows:

```
       script gatherNativeNodeAccesses(Set) :
(1)    forEach node in Set:
         begin
(2)          addTrackingVariable node.nativeAccesses ;
(3)          monitor load on node if
             currentAudienceClass() = audienceClass(node) do
(4)          node.nativeAccesses := node.nativeAccesses + 1
         end
```

Intuitively, for each node considered (1), a tracking variable is declared (2) to track the amount of accesses (3). However, only users that are a member of the audience class[80] (condition part of if) of the audience track in which the node is contained trigger an update of the monitor (3). In this way, the amount of accesses for users browsing in their designated audience track is stored.

As accesses to foreign nodes need to be stored relatively to each audience class, the script specifying the information gathering rule to store information for foreign nodes is slightly more complicated:

[80] Section 6.3.1.2 describes how the audience class of the current user can be determined.

```
     script gatherForeignNodeAccesses(Set) :
(1)  forEach audience in AudienceClasses
(2)    forEach node in Set [ FILTER on element:
(3)                          audienceClass(element) != audience]:
       begin
(4)        addTrackingVariable node[audience].foreignAccesses ;
(5)        monitor load on node if currentAudienceClass() = audience do
(6)        node[audience].foreignAccesses := node[audience].foreignAccesses + 1
       end
```

Intuitively, for each foreign node (line 2), a series of tracking variables is added, one for each audience class (line 1) except for the audience class in which' audience track the node is contained (this is done using the FILTER in line 2 and 3). Each tracking variable is only updated if the current user (i.e. the user generating the load event that causes to monitor-rule to be triggered) is a member of the audience class represented by the *audience* parameter (i.e. between [ ]) of the tracking variable (line 6).

Information gathering can now be done simply by calling the two scripts. For readability reasons, the set of nodes to be tested (i.e. n6 … n17) is defined first as a variable:

```
let ConsideredNodes be {n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16, n17};
call gatherNativeNodeAccesses(ConsideredNodes) ;
call gatherForeignNodeAccesses(ConsideredNodes) ;
```

To correct the flaw of (detected) missing information (i.e. nodes) in a certain audience track, in this example we elect to provide the relevant audience class with a navigation aid link to the detected node, by linking it from the root of the particular audience track. The ASL rules specifying this adaptive behavior are as follows[81]:

```
(1) forEach audience in AudienceClasses
    begin

(2)   let NativeNodes be ConsideredNodes[FILTER on element :
(3)                          audienceClass(element) = audience];
(4)   let ForeignNodes be ConsideredNodes[FILTER on element :
(5)                          audienceClass(element) != audience];
(6)   let threshold be median(NativeNodes[MAP on element: element.nativeAccesses]) –
                 mad(NativeNodes[MAP on element: element.nativeAccesses]);

(7)   forEach node in foreignNodes:
(8)     if node[audience].foreignAccesses > threshold
(9)     then addLink(NavigationAid, root(audienceTrack(audience), node)
    end
```

Intuitively, the rule iterates over all audience classes (1). For each audience class, the total set of nodes considered is split up into a set of native nodes (2) (3) and foreign nodes (4) (5) relative to the (current) audience class. The threshold above which foreign nodes will be considered missing is calculated next (6). Subsequently, the for-each rule (7) checks if the foreign accesses for each foreign node (relative to the current audience class) is above the calculated threshold (8). If this is the case,

[81] Although possible, we do not define a script here as it concerns only an example adaptation strategy upon detecting missing information ; other strategies can easily be imagined.

the corrective action is given in (9): adding a navigation aid link from the root of the current audience track to the particular node.

As was discussed in section 6.3.1.3 (Correcting Missing and Superfluous Information), automatically deleting nodes detected to be superfluous might result in vital information being removed from the web site[82]. Therefore, here we only give an alert to the web site administrator who can decide if he wants to remove the particular node, or not. The ASL rules specifying this adaptive behavior are as follows:

```
(1) forEach audience in AudienceClasses
     begin
(2)   let NativeNodes be
(3)    ConsideredNodes[FILTER on element: audienceClass(element) =
(4)                          audience; SORT on element: element.nativeAccesses];
(5)   let Distances be (nativeNodes / {project(nativeNodes, 1)}
                    [MAP on element: element.nativeAccesses] –
(6)         (nativeNodes / { project(nativeNodes, length(NativeNodes))}
                    [MAP on element: element.nativeAccesses];
(7)   let index be 0;

(8)   forEach node in NativeNodes :
(9)   index := index + 1;
(10)      if node.nativeAccesses <
(11)       median(nativeNodes [MAP on element: element.nativeAccesses]) –
           mad(nativeNodes [MAP on element: element.nativeAccesses]) AND
(12)       project(distances, index) > mean(nativeNodes [MAP on element: element.nativeAccess])
           + std(nativeNodes [MAP on element: element.nativeAccess]);
(13)       then alert("Node detected to be superfluous in the audience track: ", audience, " : ", node)
       end
```

Intuitively, the rule iterates over all audience classes (1) and first calculates for each audience class the native nodes (of the nodes considered) (2) (3), and sorts them according to the amount of native accesses (e.g. the value of the *nativeAccess* tracking variable) (4). Subsequently, the distances between consecutive nodes in this set are calculated (5) (6) and stored (as a set) in the variable Distances. In line (7), an index variable, to be used to reference the correct distance in the distances set, is declared and initialized to 0. Subsequently, for each native node (8), the amount of native accesses (10) is checked to be below the calculated threshold (11) and the distance to its (bigger) neighbor is checked to be above the calculated threshold (12). If both conditions hold, the node is detected to be superfluous, and an alert to the web site administrator is given (13).

[82] Note that it would possible to consider only nodes of which the web designer decided they can possibly be (automatically) deleted when detected to be superfluous. The ASL rule to specify such behavior would be slightly more complex, as the difference should be made between nodes considered for removal and nodes considered to calculate the condition-threshold.

Chapter 7: Validation

“Science is the refusal to believe on the basis of hope”
(Carrie P. Snow)

The previous chapter presented three possible (optimization) adaptation strategies, and showed how to specify them using the Adaptation Specification Language. Each adaptation strategy was intuitively explained, and an example was given.

In this chapter, we explain in detail how the work described in the previous chapters has been evaluated:

- **WSDM validation:** a prototype tool to generate from the WSDM design models the actual web site was implemented. In the prototype tool, the WSDM design models are described using OWL, and instantiated in OWL instance files. The actual data (i.e. the data source) also consists of OWL instance files. Generation of html code is done using XSLT and XPath.
- **Adaptation Specification Language validation:** a prototype evaluator for the Adaptation Specification Language has been implemented using Java. A relational database was used to store (and retrieve) the user access information. The evaluator accesses the relational database, and transforms the OWL instance files (representing the WSDM design models) to reflect the adaptive changes to the web site design according to the ASL specification.
- **Adaptation Strategies validation:** The adaptation strategies promotion, and validation and correction of the audience driven design were implemented using ASL, and applied on a case study (using the ASL evaluator discussed in the previous bullet). Results of the experiment are reported and interpreted.

The remainder of this chapter is structured as follows. Section 7.1 explains the implementation of the prototype tool that generates an implementation from the WSDM design models. Section 7.2 discusses the implementation of the Adaptation Specification Language prototype evaluator. Section 7.3 presents the experiment. In the experiment, promotion and design validation adaptation strategies are applied on the web site for the university laboratory of the author realized using WSDM. Finally, results are discussed.

7.1 WSDM Validation

To validate the WSDM method and more particularly the ability to fully automatically generate an actual implementation (i.e. a web site) starting from the various design models, a prototype tool has been built. The prototype was realized using Semantic Web technology[83] to describe the different WSDM design models, as well as to realize the transformation pipeline as described in Chapter 3 (WSDM Informal), section 3.6 (Implementation Generation).

The next two sub sections go into deeper detail on respectively the representation of the WSDM design models as an OWL ontology (called the WSDM Ontology), and the implementation generation from instantiations of this ontology.

7.1.1 WSDM Ontology

The different WSDM design models, as presented in Chapter 3 and formalized in Chapter 4, have been represented in the prototype by means of an ontology using OWL[84]. This ontology is called the WSDM Ontology. Although, for the purpose of this dissertation, the WSDM models could have been represented in any format[85], the OWL representation format was selected for the following reasons:

- **Reusability:** One of the well-known advantages of ontologies concerns the improvement of

[83] This offers several advantages, such as possibility for (automatic) annotation or integration with other design methods. However, an elaboration on these advantages is outside the scope of this thesis.
[84] http://wise.vub.ac.be/ontologies/WSDMOntology.owl
[85] E.g. using plain XML

interoperability between systems. Defining the meta-model of the design models explicit (by means of an ontology), makes interoperability between different design methods easier, and allows us to reuse design models, even if they were specified in another design methods.

- **Semantic (Content) Annotations:** the true strength of the WSDM ontology lies in the fact that existing domain ontologies can be used in (parts of) the WSDM design process (e.g. to model object chunks), and an explicit link between the web site implementation and the domain ontology used, can be maintained. This paves the path for a web engineering approach that supports semantically annotated web applications.
- **Semantic (Structural) Annotations:** Using the information in the WSDM ontology, annotations concerning the structure of the web site (e.g. composition of pages by different page elements, origin of data, intended meaning of page object) can be generated. In this respect, annotations for visually impaired users have been realized using the WSDM Ontology [Plessers et al, 2005]
- **W3C Recommendation:** OWL is a W3C recommendation, and likely to become the W3C standard to describe ontologies.

The WSDM Ontology consists of a series of sub-ontologies, one for each design phase of WSDM (i.e. mission statement, audience modelling, conceptual modelling and implementation modelling). Each of these sub-ontologies can be instantiated to series of OWL instance files, which together describe a complete web site design. From this web site design, an actual web site (in the chosen implementation platform) can be generated. How this is done is described in the next subsection.

7.1.2 WSDM Implementation Generation

To validate the expressiveness (i.e. is the information comprised in a WSDM design sufficient to represent an actual web site) and applicability (i.e. can a web site be generated from a WSDM design) of the WSDM Ontology, a prototype tool to generate the actual implementation has been implemented. The implementation architecture, a five-step transformation pipeline (see Chapter 3, section 3.6 Implementation Generation), has been realized using XSLT and xPath. For our proof-of-concept implementation, XHTML was chosen as the target implementation language, using (xhtml-) tables as positioning mechanism for page objects. Cascading Style Sheets were used for layout purposes. More concretely, the transformation pipeline takes as input OWL instance files containing the object chunks, navigational model, site structure design, template design and page design as inputs. It consists of five steps:

- **High Level Transformation:** is responsible for the transformation of the high-level template and presentation concepts into the basic set of primitive presentation concepts. Although the high-level presentation concepts are already described in the WSDM Ontology, the high level transformation was not yet implemented in the prototype. Instead, the current WSDM designs were made only using primitive presentation and template concepts, thus not requiring this transformation.
- **Model Integration:** integrates the navigation model, page structure model, template and page model into one single model. This transformation was realized using XSLT, taking as an input the different OWL instance files representing the respective WSDM design models, and outputting one single OWL instance file, containing the full WSDM design for the target web site.
- **Implementation Mapping:** the (partial) mapping of the integrated WSDM design model to the actual implementation platform, XHTML, was done using XSLT. References to data (i.e. references to lexical object types of object chunks in the presentation design) are not

processed in this transformation but are dealt with in the next step.

- **Data Source Mapping:** references to lexical object types in object chunks are mapped on their data source, OWL instance files. Using XSLT, the references to object chunks were transformed to executable XPath[86] queries over the OWL instance files. The output of this step is an XSLT transformation.
- **Query Execution:** executing the output of the previous step, the XSLT transformation, the actual web site is created (i.e. the queries created in the previous phase are executed). The prototype initially supported static web sites[87], but was extended to support dynamic web sites (see next section).

Note that this versatile transformation pipeline offers a lot of flexibility. E.g. it is possible to substitute one single mapping (step) without the need to change the other mappings. In this way, one aspect of the generated web site can be changed. For example, to change our choice of xhtml tables for positioning purposes to Cascade Style Sheets, only the Implementation Mapping needs to be replaced.

7.2 Adaptation Specification Language Validation

The implementation generation as described in the previous section supports the generation of static web sites and potentially dynamic web sites. However, solely using XSLT to generate a web site, without any support of a server side scripting or implementation environment, limits the possibilities to store user access information and to adaptively take actions based on this information. Therefore, a generic server side architecture has been set up, aimed to serve the web site based on the relevant underlying WSDM design models, while also allowing to add programming logic. In this case, the programming logic consists of storage of relevant information about the user accesses on the web site and transformations of the relevant design models based on ASL rules.

An overview of the architecture can be found in figure 7.1. Before going into detail on the technologies used to implement this architecture, let’s first informally describe the implementation architecture:

- **User:** a user visiting the web site, i.e. generating http requests and receiving http responses.
- **Web site Engine**: the central component of the architecture, responsible for both receiving http requests from the client and in response returning an actual webpage (i.e. the http response). The http request specifies the requested page, which the Web site Engine successively composes based on the information contained in the relevant **WSDM Design Models**. I.e. it derives from the design models the nodes, links, object chunks and relevant presentation design for the requested page, and then constructs this page, which is delivered to the user. The engine also generates events, which are passed to the **Active Event Queue**. These events are obtained either by analysis of the http request (i.e. link information) or throughout the page assembly process (i.e. the load events of the various design elements).
- **Active Event Queue:** obtains events from the **Web site Engine** and on the basis of these events, triggers the appropriate **ASL Rules** according to the Adaptation Policy of the web site. These rules are evaluated by the **ASL Evaluator**, and (possibly) give rise to some adaptive action. In most cases, the action will simply be an update in the **database**, containing user

[86] See http://www.w3.org/TR/xpath.
[87] Due to the limitation of XSLT 1.0 to output only one single file, manual separation of the different xhtml pages was required. The W3C reports XSLT 2.0 (under preparation) will include support for outputting several files.

access information. However, the scheduler is also responsible to trigger rules which possibly will adaptively change one or more **WSDM design models**.

- **ASL Evaluator:** evaluates **ASL rules** passed by the **Active Event Queue,** which possibly give rise to adaptive changes in the **WSDM design models**, or perform updates in the **database** of user access information.
- **ASL rules:** contain specifications of adaptive changes to the **WSDM design models**, and updates to the **database** according to the user access information.
- **User Accesses Database:** contains the user access information to the various elements of the **WSDM design models**.

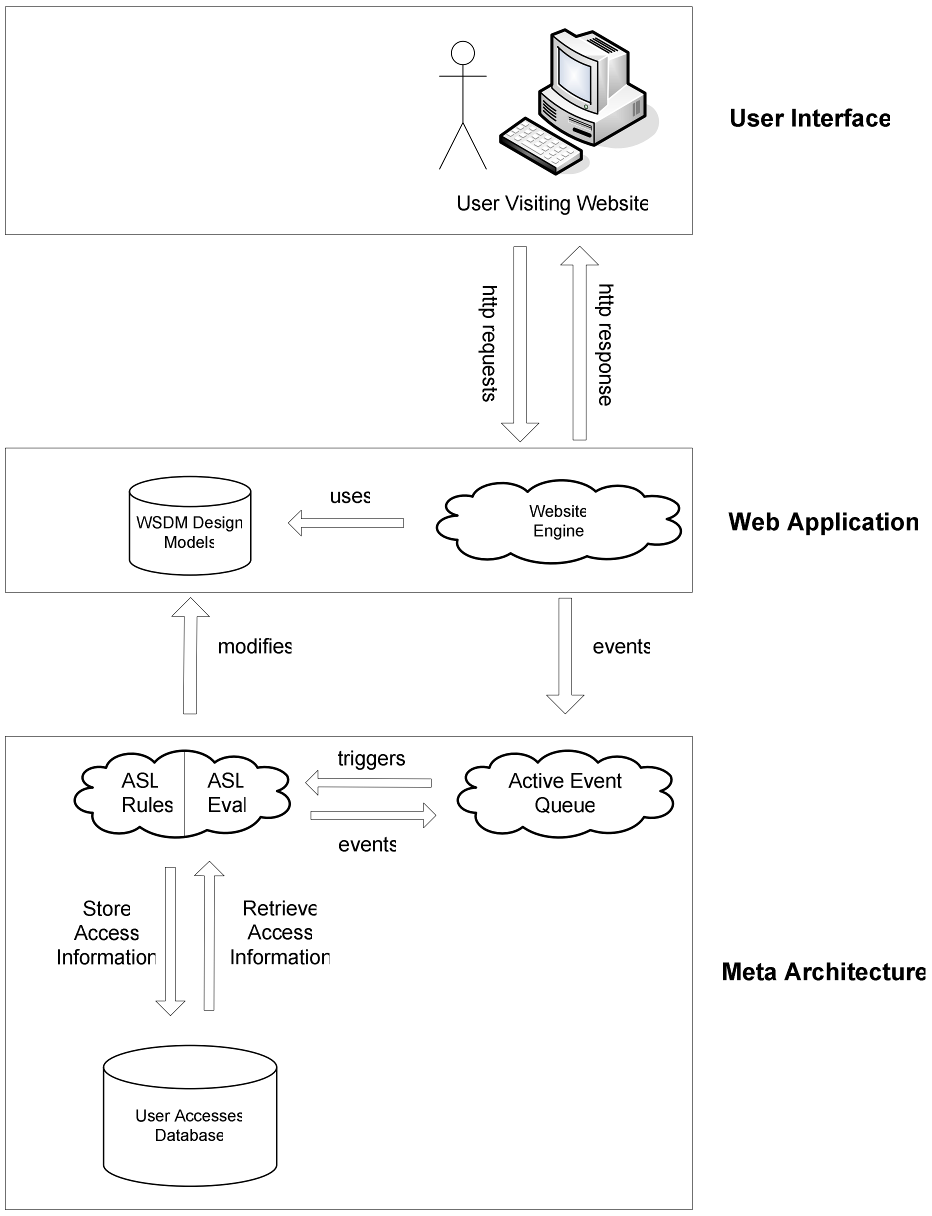


Figure 7-1 WSDM Implementation Architecture supporting ASL

As can be observed from the figure, this architecture can be considered three layered:

- the *client side layer* (user interface), a standard web browser
- the web *application layer*, responsible for assembling the correct web page according to the request received from the user
- the *meta architecture layer*, responsible for monitoring user accesses and adapting WSDM models according to the specified adaptation policy.

This architecture, serving dynamic web sites with ASL support, was implemented using the following technologies:

- **Web site Engine:** the web site engine was implemented using a (single) Java servlet, running on an Apache Jakarta Tomcat web server.
- **Page Generation:** the actual generation of a web page, starting from the WSDM design models, was done using the XSLT transformations described in the previous section.
- **Active Event Queue:** the active event queue was only partly implemented in the current prototype. All rules involving tracking variables (i.e. declaration, updates) were implemented using Java. Using standard java events, generated by the web site engine, initialization, load, click, sessionEnd and sessionStart events are supported. However, time events and events involving (values of) tracking variables (i.e. conditions), are not yet supported in the current prototype. Adaptation Strategies triggered by either a time event or a condition were thus initiated manually.
- **ASL Evaluator**: the ASL evaluator was implemented using Java 1.5. Starting from the BNF grammar for ASL, an ASL parser was generated using the Java ANTLR parser generator. This parser was used to parse ASL specifications, resulting in Abstract Syntax Trees. Successively, an ASL evaluator was written, which maps Abstract Syntax Trees to manipulations on the WSDM Design Models, or updates in the database (see following).
- **WSDM Design Models Manipulation:** manipulations of the WSDM Design Models, which are represented as OWL file, were done using Java on an internal graph structure. This internal structure was created using the Java Xerces XSML Parser 1.4.4 to parse the OWL file representing the WSDM Design Models. All basic model transformation operations (see Chapter 5 section 5.2, Basic Information storage operations) were implemented on the internal structure, and the control flow operation defined by ASL were implemented on the internal model using standard Java control flow constructs.
- **ASL rules**: ASL rules are, evidently, described in ASL.
- **Database**: a MySQL relational database was used, with standard SQL for database creation and querying.

Note that the implemented prototype does not fully support all features of WSDM or ASL. However, it does provide a proof of concept, and was used to implement the case study, discussed in the next section. Furthermore, the ASL implementation was validated by performing some adaptation strategy experiments with the case study.

7.3 Case Study

In this section, the case study used to validate both the WSDM and the Adaptation Specification Language implementation is presented. In the following subsection, the subject of the case study, and the relevant WSDM design models are provided. The next section will show how the case study was used to perform experiments with adaptation strategies.

7.3.1 Subject

The subject of the case study is a web site for WISE, the university laboratory of the author at the Vrije Universiteit Brussel. The web site has been modelled using WSDM, and implemented as discussed in section 7.1 (WSDM Validation) and 7.2 (Adaptation Specification Language Validation). In the following section, the (audience driven) WSDM design for the WISE web site is presented. Conform Chapter 4 (WSDM Formalization), only the relevant models are provided, the techniques used to derive these models are not relevant and thus not discussed in this dissertation.

7.3.2 WSDM design

Mission Statement

The mission statement in WSDM expresses *the purpose*, *subject* and *targeted users* of the web site. The purpose of the case study web site, the WISE web site, is to provide all information related to the WISE research laboratory; the mission statement is thus formulated as follows:

"The subject of the web site is the research laboratory WISE from the Computer Science department of the Vrije Universiteit Brussel. It is the purpose of the WISE web site to provide both researchers and students with all relevant information about the research group, such as general information, research activities (publications, research projects, research topics, …) and teaching activities (courses, theses, …)."

Audience Modelling

In the Audience Modelling phase, the different types of visitors are analyzed and their requirements and characteristics stated. Based upon their requirements, the targeted users are classified in audience classes.

Based on the Mission Statement formulated in the previous subsection, the following target audiences can be derived for the WISE web site: *student* and *researcher*. For each target audience, all information and functional[88] requirements are listed:

- *Student*
    - General information about WISE
    - Overview of WISE members
    - Courses overview
    - Detailed course information
    - Thesis proposals
    - Thesis students at WISE
    - Past theses performed at WISE

- *Researcher*
    - General information about WISE
    - Overview of WISE members
    - Overview of research topics of WISE
    - Detailed research topic information
    - Overview of research projects
    - Detailed information on research projects

[88] In the WISE web site case study, there are no functional requirements.

- Publication overview
- Detailed information on publications
- Information on visiting researchers

Students and researchers have some common requirements, i.e. general information about WISE, overview of WISE members. These requirements can therefore be assigned to the Visitor audience class, the audience class representing the requirements all users share. Conform the audience subclass relations as described in Chapter 4 (WSDM Formalization), section 4.2 (Audience Modelling), both the *Student* and the *Researcher* audience class are audience subclasses of the *Visitor* audience class. The following audience hierarchy is derived:

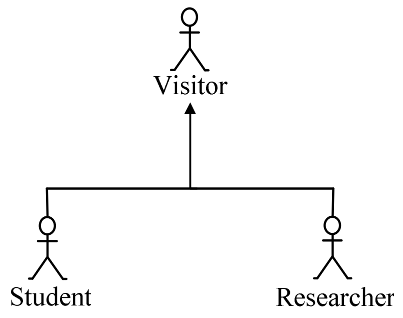


Figure 7-2 Audience Class Hierarchy for the WISE web site

For each audience, WSM prescribes to list relevant characteristics, which can later be taken into account when devising the navigation and presentation design. For the WISE site case study, the characteristics are as follows:

- *Student*
  - Between 18 and 24 years old
  - Fluent in Dutch or English
  - Motivated to use the web site (as it contains information needed for their studies)

- *Researcher*
  - Used to use English to communicate about research
  - Domain experts (concerning their research field)

Conceptual Modelling

During the *conceptual design* phase, the tasks of the different audience classes are analyzed in detail, and the exact information / functionality requirements needed are modeled in a formal way (i.e. task & information modelling). Subsequently, using the audience class hierarchy of the previous phase, the main navigation structure (i.e. the audience tracks) of the web site is derived. Using the task models, the available information/functionality is organized within each audience track and the (internal) navigation structure is decided.

In the next two sub sections, the *task and information* and the *navigation modelling* for the WISE web site case study are provided:

Task and Information Modelling

As was discussed in Chapter 3 (WSDM Informal), the methods and techniques used in WSDM to derive the various design models are not the subject of this dissertation and thus not discussed.

Therefore, we do not discuss the task modelling phase for the WISE web site case study, nor how these task models are used to derive the internal navigation structure for each audience track.

The information modelling, the activity of constructing an object chunk for each requirement, is presented here on the basis of two representative object chunks: ShowCourses and ShowCourseDetail. The first object chunk, ShowCourses, gives a general overview of all courses WISE are involved in in some way (i.e. both WISE courses and non-WISE courses, differentiated by their type). This object chunk is represented in figure 7.3. The second object chunk, ShowCourseDetail, shows the details of a course (e.g. responsible professor, assistants, practical information, course material, etc). This object chunk is represented in figure 7.4. How the user can browse from the overview of courses to a specific course is represented in the Navigation Model (see next sub section).

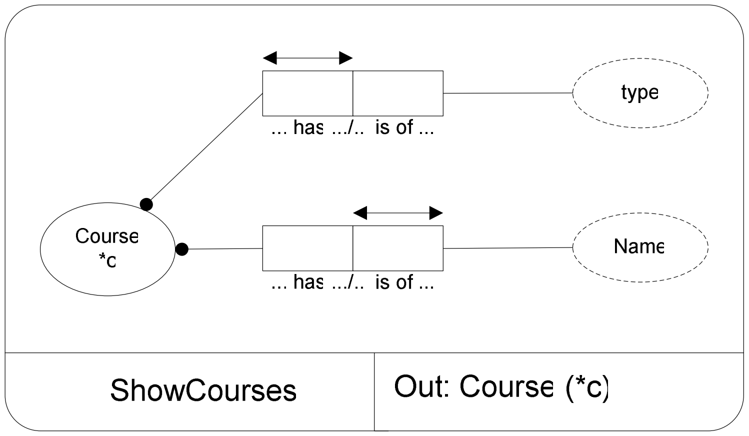


Figure 7-3 Object Chunk "ShowCourses"

Other object chunks of the WISE web site case study are described in a similar manner:

- **Overview type of object chunks**: object chunks that provide an overview of particular items, similar to the showCourses object chunk (i.e. ShowMembers, ShowOngoingProjects, ShowPastProject, ShowResearchTopics, ShowResearchProjects, ShowVisitors, ShowNewS&T, ShowCurrentS&T, ShowPastS&T)
- **Detailed type of object chunks:** object chunks that, where required, provide detailed information about one particular item, similar to the ShowCourseDetail object chunk (i.e. ShowHomepage, ShowOngoingProjectDetail, ShowPastProjectDetail)
- **Remaining types of object chunks:** object chunks that represent some (general) information (i.e. Coordinates, General Information, GettingThere)

The overview type of object chunks may have an output-parameter (i.e. a reference uniquely identifying the particular lexical object instance), which can be passed along a link (see next sub section). Detailed type of object chunks require an input-parameter, identifying the particular item for which detailed information need to be showed. The remaining types of object chunks do not require any input or output parameters, as they represent stand-alone information.

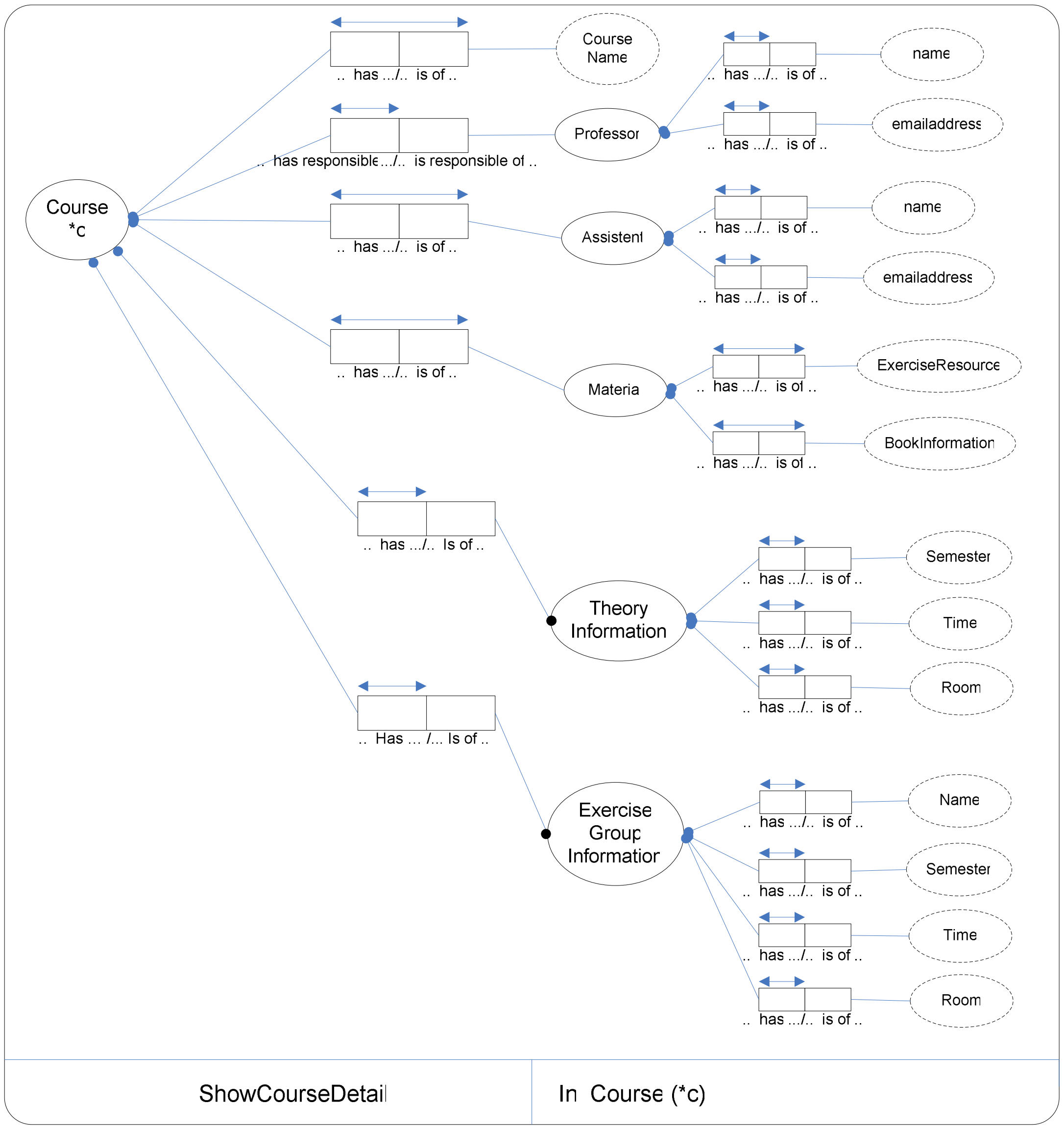


Figure 7-4 Object Chunk "ShowCourseDetail"

Navigation Modelling

During Navigation Modelling, the conceptual structure of the web site is devised, and it is decided how the different visitors (of the different audience classes) will be able to (conceptually) navigate through (the information/functionality presented on) the site. This conceptual navigation structure, called the Navigation Model, is represented by means of nodes (conceptual navigation entities, representing some information that logically belongs together), with zero or more object chunks connected to them (representing the information contained in a node). Note that the actual page structure is not yet decided at this point.

Figure 7.5 shows a general overview of the Navigation Model for the WISE web site case study. Not to overload the figure, the student and researcher audience track are not elaborated in this figure, but presented in separate figures. This general navigation structure is derived from the Audience Class

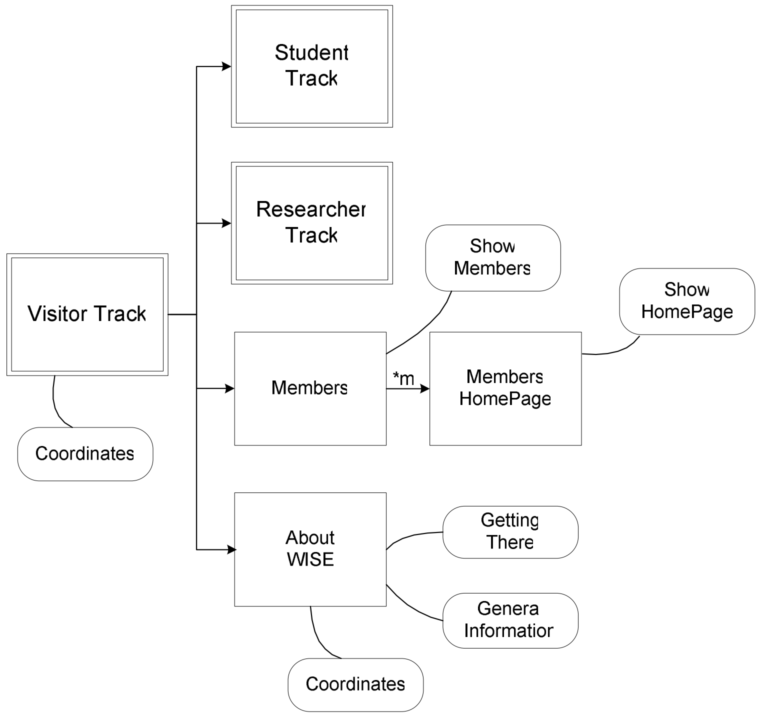


Figure 7-5 Overall Navigation Model for the WISE Web site

Hierarchy, as described in Chapter 3 (WSDM Informal) section 1.3 (Audience Modelling), and complemented with nodes representing information required by all users (i.e. object chunks resulting from requirements of the Visitor audience class). Consistent with the notation introduced in Chapter 3 section 3.4.2 (Navigational Design), nodes are represented by rectangles, (root of) audience tracks by double rectangles, object chunks by rounded rectangles, and links by arrows between nodes.

The detailed navigation model for the researcher and student audience track is given respectively in figure 7.6 and 7.7. Recapitulating on the ShowCourses and the ShowCourseDetail object chunks described in the previous subsection, we see in the Student audience track how the output parameter from the ShowCourses object chunk is passed to the ShowCourseDetail object chunk, along the link between WISEVUBCourses and the Course node. The link between those two nodes denotes that is will be possible to navigate from each course in the overview to detailed information about this course; the parameter ensures the course details are presented of the course selected by the user. Note that, although the detailed information about a course is represented by one object chunk, connected to one node, the ShowCourseDetail chunk may give rise to multiple instances of the object chunk (i.e. one for each course) and thus also to several instances of the conceptual node Course.

Implementation Modelling

The goal of the implementation modelling is to add to the conceptual design models, in an abstract way, the necessary details needed to generate the actual implementation. Concretely, the page structure is devised (deciding which nodes to represent on one page), and the look and feel of the web site (i.e. templates) and the layout of each page (i.e. page presentation) is described.

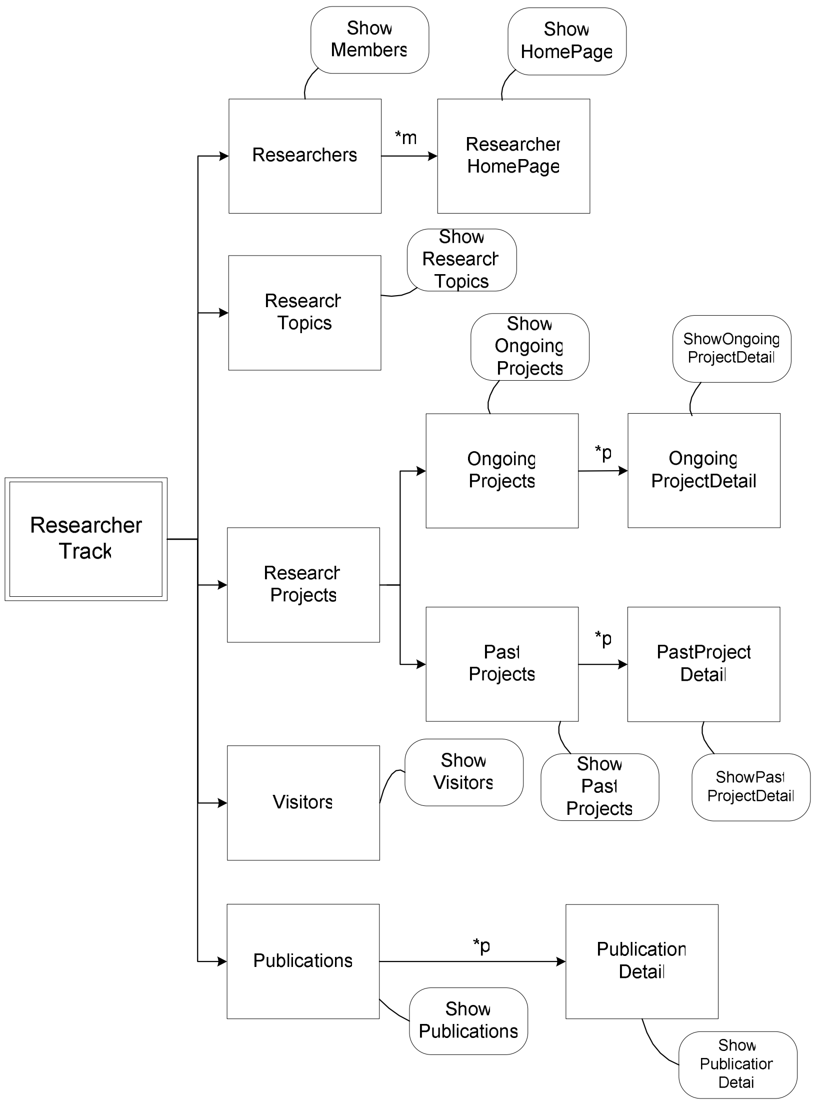


Figure 7-6 Detailed Navigation Model for the Researcher Audience Track

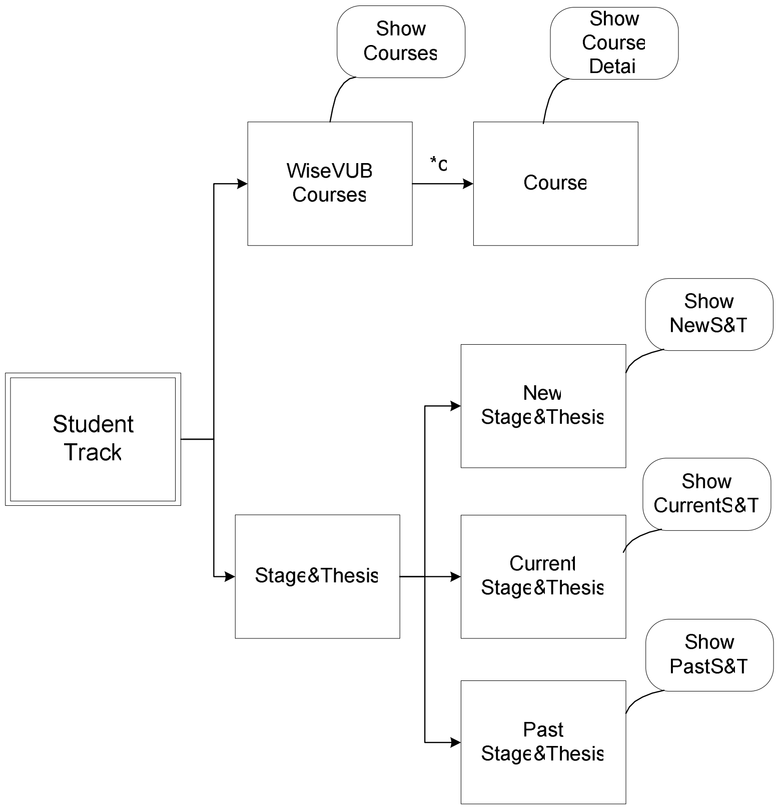


Figure 7-7 Detailed Navigation Model for the Student Audience Track

In the next three subsections, respectively the site structure, the template and the page presentation design are presented.

Site Structure Design

For the WISE web site case study, the site structure model is very simple: every node is represented on a separate (abstract) page. Considering the WISEVUBCourses and Course nodes,
each node is placed on a separate abstract page. In the generated web site, this will results in a (concrete) page presenting an overview of all courses, and a separate (concrete) page for each course. Note that the designer does not necessarily need to represent each node on a separate (abstract) page. For example, grouping the WISEVUBCourse and Course node on one abstract page, would result in one (concrete) page containing an overview of all courses with (in-page) links to detailed information for each course.

Template Design

Templates describe the shared (presentation) part for the different *types* of pages. For the WISE web site case study, we have defined three templates:

- **Homepage Template:** used for the homepage of the WISE web site case study, i.e. the abstract page containing the Visitor Track node[89].

[89] In some cases, a Homepage template can be used multiple times, therefore, it is considered good practice to make a Homepage template. For example, when a web site is offered in more then one language (possibly

- **Menu Template**: used for the abstract pages (menu pages) that present a collection of links to other pages. More specific, this template is used for the abstract pages containing the following nodes: Student Track, Researcher Track, WISEVUBCourses, Stage&Thesis, OngoingProjects, Past Projects, Publications
- **Information Template**: used for abstract pages that present information of some kind. More specific, this template is used for the abstract pages containing the following nodes: Members, MembersHomePage, AboutWISE, Course, NewStage&Thesis, CurrentStage&Thesis, PastStage&Thesis, ReseachTopics, ResearchProjects, Visitors, OngoingProjectDetail, PastProjectDetail, Researchers, ResearcherHomePage

As an example, the Information Template is schematically given in figure 7.8. As the formal model of Chapter 4 (WSDM Formalization) does not go into detail on high level presentation concepts (see Chapter 3, section 3.5.2.3, Page Design), we describe templates here only in terms of primitive presentation concepts. The actual dimensions of the different (template) grids used are scaled down for the sake of the figure, and links contained in the template are not represented in the figure. To be complete, the images in the right top corner are augmented with (navigation aid) links to respectively the root of the wise web site, the department web site, the faculty web site and the VUB web site.

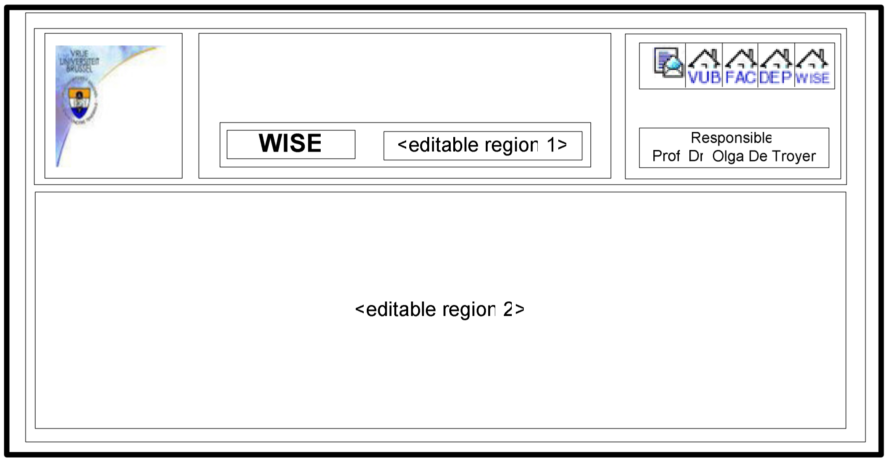


Figure 7-8 Information Template for the WISE Web site Case Study

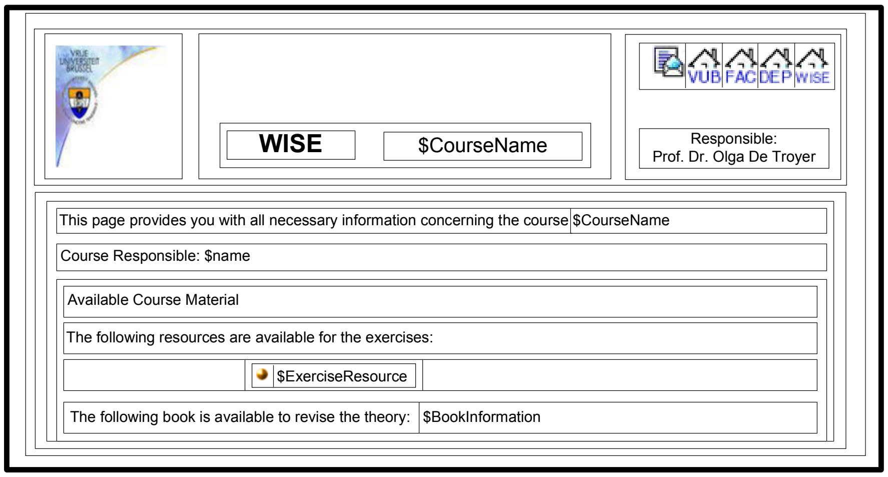
Page Presentation Design

The purpose of the Page Design is to describe exactly how the data, originating from the (instances of the) object chunks, will be positioned and laid out on a page. For each (abstract) page, in the Page Presentation Design the modeler should start from a Page Template, and fill the editable regions with grids representing the data modeled in the object chunks connected to the nodes contained by that particular page.

As an example for the WISE web site case study, consider (a simplified version of) the Page Presentation Design for the abstract page representing the Course node. As it concerns a page presenting information, the Information Template is used as a starting point. References to (lexical)

containing different information), the Homepage template may be re-used for each (language) version of the homepage.

object types of object chunks are denoted using a starting "$", followed by its name[90]. Not to overload the figure, only a few representative examples are given; the other object types can be done in a similar way.



WISE
$CourseName
VUB
FAC
DEP
WISE
Responsible:
Prof. Dr. Olga De Troyer
This page provides you with all necessary information concerning the course
$CourseName
Course Responsible: $name
Available Course Material
The following resources are available for the exercises:
$ExerciseResource
The following book is available to revise the theory:
$BookInformation

Figure 7-9 Simplified Page Presentation Design for the Course Page

Other abstract pages can be constructed in a similar way. Typically, the Page Presentation Design will be performed using a graphical tool, thereby shielding the designer from the laborious task of manually specifying the presentation grids.

Implementation Generation

The WISE web site case study was implemented as a dynamic web site, with support for the adaptation as described in the previous section.

7.3.3 Adaptation Strategies

Using the WISE web site case study, further on referred to as the experimental WISE web site, two of the adaptation strategies described in Chapter 6, promotion and validation of an audience driven design, have been applied and evaluated. These experiments, their results and an interpretation of these results are given in the next two sub sections.

7.3.3.1 Promotion

Promotion was specified for all nodes representing specific information (i.e. not on overview pages, such as the member's page; and not on 'hub' pages, such as the root of each audience track[91]) in the student audience track, and on all hub (or overview) pages in the researcher audience track.

Setup

[90] Note that this is a simplification in the graphical presentation used; in the formal presentation model described in Chapter 4, references to (uniquely identified) lexical object types are used, thereby avoiding the problem of lexical object types with the same name (e.g. name of professor and assistant).

[91] Indeed, if we would compare the amount of accesses to a page containing detailed information (e.g. a particular course) with the amount of accesses to a 'hub' page (e.g. the page giving an overview of all courses), naturally, the hub page would have more accesses, as a lot of visitors just pass by.

An Adaptation Policy specifying information gathering from the initialization of the web site, and after two months performing promotion was defined. ASL specification to gather the necessary information to perform promotion described above is given as follows (note re-use of the *trackAmountOfAccesses*-script, defined in Chapter 6, section 6.1.1.3) :

```
when initialization do

    begin
    let ConsideredStudentNodes be (ALL Course) UNION {NewStage&Thesis,
                                       CurrentStage&Thesis, PastStage&Thesis} ;

    let ConsideredResearcherNodes be { Publications, Visitors, ResearchTopics,
                                          OngoingProjects, PastProjects, Researchers } ;

    call trackAmountOfAccesses(ConsideredStudentNodes);
    call trackAmountOfAccesses(ConsideredResearcherNodes)
    end
```

The actual adaptation consists of adding a link on the home page (i.e. the Visitor Track node) to the most popular item of each audience track. As the amount of navigation links on the homepage is limited (i.e. there are four navigation choices from the homepage), our choice to add two more navigation (aid) links, directly accommodating a significant amount of users of both the student and the researcher audience class, will not overload the page and is therefore acceptable. The ASL specification is as follows:

```
when 2 months from now do
    begin

    let studentMax be
      max(ConsideredStudentNodes [MAP on element: element.amountOfAccesses]);
    let researcherMax be
      max(ConsideredResearcherNodes [MAP on element: element.amountOfAccesses]);

    forEach node in ConsideredStudentNodes:
       if  node.amountOfAccesses = studentMax
      then addLink (navigationAid, VisitorTrack, node) as StudentPromotionLink;

    forEach node in ConsideredResearcherNodes:
      if  node.amountOfAccesses = researcherMax
      then addLink (navigationAid, VisitorTrack, node) as ResearcherPromotionLink

    end
```

After initialization, the experimental WISE web site has run for two month, gathering information as specified by the information gathering rules. During those two months, the experimental WISE web site was visited 1490 times[92], with an average of 24 visitors a day. There were significantly more visits to the student audience track, compared to the researcher audience track. The most popular node of the student audience track was the Course node showing information on the course "Inleiding

[92] Note that the original WISE web site was also still available. Visitors thus had the choice to use the original or experimental site.

Databanken" (279 visits), and in the researcher audience track the Publications page (46 visits). Both these nodes were thus promoted to the homepage of the site.

Results

After promotion of the "Inleiding Databanken" Course node, and the "Publications" node, all tracking variables were reset, and the gathering of information was restarted. Ad hoc evaluation could have been done after a certain timeframe, but instead, we have used ASL to give the designer an overview of the effectiveness of his specified adaptation rules. Therefore, we specified, in the web sites adaptation policy, an additional strategy to monitor the effectiveness of the performed adaptation. To evaluate how many times the adaptively added links were used to reach the two popular nodes, the following strategy is specified:

```
(1)     script resetAmountOfAccesses(Set) :
(2)             forEach element in Set :
(3)                     element.amountOfAccesses := 0

(4)     script trackAmountOfClicks(Set) :
(5)        forEach link in Set :
(6)             begin
(7)             addTrackingVariable link. amountOfClicks;
(8)             monitor click on link do link.amountOfClicks := link. amountOfClicks + 1
(9)             end


(10)    when 2 months 1 day from now do
(11)    begin

(12)    call resetAmountOfAccesses(ConsideredStudentNodes);
(13)    call resetAmountOfAccesses(ConsideredResearcherNodes);

(14)    call trackAmountOfClicks({StudentPromotionLink, ResearcherPromotionLink})

(15)    end
```

We start off with the two scripts. The first script (line 1 – 3) resets all *amountOfAccesses* tracking variables for a certain set; the second script (line 4 – 9) attaches *amountOfClicks* tracking variables to the element of the given set (of links) (line 7), and updates the tracking variable when a click event on a link occurs (line 8). The actual (adaptation) strategy starts on line 10, where we denote that after 2 months and 1 day (line 10), the amount of accesses for all nodes considered for promotion (i.e. nodes from both the student and the researcher audience track) is reset (line 12 – 13)[93], and tracking of amount of clicks for both links that were added by promotion is specified.

Successively, the following rule was specified to alert the web designer how many times each adaptively added link was used, after the web site ran for 1 month with the promotion links:

```
        when 3 months from now do
        forEach link in {StudentPromotionLink, ResearcherPromotionLink}:
```

[93] Note that load events on these nodes are still monitored, and tracking of amount of accesses thus continues.

alert("The total amount of accesses to the node ", target(link), " during the last month was ", target(link).*amountOfAccesses*, " times, of which ", link.amountOfClicks, " (= ", 100 * link.amountOfClicks / target(link).amountOfAccesses, " %) accessed the node by using the adaptive link.")

After one month, the following results were obtained:

- The total amount of accesses to the node "Inleiding Databanken" during the last month was 174 times, of which 122 (= 70 %) accessed the node by using the adaptive link.
- The total amount of accesses to the node "Publications" during the last month was 44 times, of which 25 (= 57 %) accessed the node by using the adaptive link.

Discussion

As a significant amount of users clicks on the adaptively added links, we can consider those links a useful addition to the existing web site navigation structure, effectively reducing the amount of clicks *a lot of* the users need to make to find the information relevant for them. By only adding links to the most popular pages, we have found a good compromise between accommodating a relative large group of users yet not overloading all users with unnecessary links[94].

Further analysis of user accesses to one of the popular nodes (i.e. the "Inleiding Databanken" or "Publications" node), but not using one of the adaptive links reveals that 26 of a total of 71 users (37%) arrived to the node through an external link (e.g. they arrived through a search engine, bookmarked the page or were linked by an external site). These users do not use the sites navigation structure (i.e. they start their session immediately on the particular node) and can thus not be considered as failing to use the adaptive links. 45 of the 71 users not using the adaptive link to arrive to one of the popular nodes (63%) do start their session on the homepage and (still) use the sites navigation structure to arrive to their respective destination. This leads us to conclude that 166 out of 192 users (86%) that use the site's navigation structure and visit one of the promoted nodes, use the adaptive link added by promotion to do so.

Finally, further (manual) analysis was done in order to compare our results to the results reported in [Perkowitz and Etzioni, 1997a] by Perkowitz and Etzioni, who did some manual experiments with page promotion. They reported that about 25% of visitors used at least one link created by promotion[95]. The results obtained in our experimental WISE web site shows that during the evaluation period of one month there were 548 sessions performed on this. In 113 of these 548 sessions at least one of the two adaptive links was used, equalling 20,6% of all sessions. This result is thus somewhat in correspondence with [Perkowitz and Etzioni, 1997a].

7.3.3.2 Audience Driven design validation

To validate the audience driven design for the experimental WISE web site, identification of missing information was specified for both the student and the researcher audience track. As for promotion, only nodes representing specific information are considered in the validation process.

Setup

[94] Indeed, the amount of clicks needed to access all pages could be dramatically reduced by simply flattening the navigation hierarchy to one level, and presenting all users with one huge list of links to all available information on the homepage. However, it is widely agreed that this does not increase the usability of the web site, and in fact increases the time needed for the user to find the *correct* link.
[95] Exact numbers of how many promotion links were added, were not reported.

An Adaptation Policy specifying information gathering from the initialization of the web site, and after two months performing audience driven design validation to detect missing information was specified. For the ASL specification to gather the necessary information to detect missing information in each audience track, we re-use the information gathering script (specified in ASL) defined in Chapter 6, section 6.3.3). We also define a variable ConsideredNodes to hold the nodes that will be considered in the algorithm:

```
when initialization do

    begin
    let ConsideredNodes be (ALL Course) UNION {NewStage&Thesis, PastStage&Thesis,
    CurrentStage&Thesis, Publications, Visitors, ResearchTopics, OngoingProjects, PastProjects,
    Researchers};

    call gatherNativeNodeAccesses(ConsideredNodes) ;
    call gatherForeignNodeAccesses(ConsideredNodes)
    end
```

As our goal here is to evaluate if our algorithm to detect missing information is feasible, and if it can indeed by logically explained why certain information is detected to be missing, for the actual adaptation strategy it suffices to alert the web designer if and which nodes were found to be missing in one of the two audience track (i.e. Researchers and Students):

```
(1)  when 2 months from now do
(2)  forEach audience in AudienceClasses
     begin

(3)    let NativeNodes be ConsideredNodes[FILTER on element :
(4)                          audienceClass(element) = audience];
(5)    let ForeignNodes be ConsideredNodes[FILTER on element :
(6)                          audienceClass(element) != audience];
(7)    let threshold be median(NativeNodes[MAP on element: element.nativeAccesses]) –
                   mad(NativeNodes[MAP on element: element.nativeAccesses]);

(8)    forEach node in foreignNodes:
(9)      if node[audience].foreignAccesses > threshold
(10)     then alert("The node: ", node, " of the audience track ", audienceTrack(node),
                  " was detected to be missing for the audience class ", audience)
(11)  end
```

The for-each rule on line 2 is in fact very similar to the one specifying the example adaptation strategy for missing information in Chapter 6, section 6.3.3 (ASL Implementation): for each audience class (line 2), the native and foreign nodes are calculated (line 3 – 4, 5 – 6) and the threshold is calculated (line 7). Subsequently, for each foreign node (line 8), if it has been detected missing for a certain audience class (in that audience class's audience track; line 9), an alert to the web designer is given (line 10).

The experimental WISE web site subsequently ran for two months. The results obtained are described in the next sub section.

Results

A diagrammatic representation of the results is given below in the form of the audience class matrices for the "Researcher" and "Student" audience class. The first column (i.e. "Node") denotes the node, the second and third column (i.e. "Student" and "Researcher") denotes the amount of visits to the particular node by a member of respectively the student or researcher audience class.

For the considered nodes of the Student audience class, the audience class matrix is as follows:

Node	Student	Researcher
Course Communicatievaardigheden voor Informatici	8	0
Course Design methods for Internet-based Systems	111	0
Course Inleiding tot de Informatica	205	0
Course Stage/Onderzoeksproject en Rapportering	10	0
Course User Aspects of Software Systems	17	0
Course Inleiding tot databanken	277	2
CurrentStage&Thesis	19	0
PastStage&Thesis	24	1
NewStage&Thesis	37	1

Figure 7-10 Audience Class Matrix for Students

For the considered nodes of the Researcher audience class, the audience class matrix is as follows:

Node	Student	Researcher
Publications	6	40
Researchers	11	53
Visitors	4	5
ResearchTopics	6	16
Pastprojects	0	4
Ongoingprojects	1	7

Figure 7-11 Audience Class Matrix for Researchers

We are now ready to calculate the threshold for both audience classes to detect missing information. We do so by using the amount of accesses to native nodes:

- **Student audience class** (first table, student column)**:**
  Median: 24
  MAD: 15
  Threshold: 24 – 15 = 9
  Now examining the foreign clicks by the Student audience class (second table, student column), we observe that there is one node that has an amount of clicks higher then the threshold: "Researchers". This node is thus detected to be missing from the Student Audience Track.

- **Researcher audience class** (second table, researcher column)**:**
  Median: 10
  MAD: 5,5
  Lower limit: 10 – 5.5 = 4.5
  Now examining the foreign clicks by the Researcher audience class (first table, researcher column), we observe that there is no missing information in the researcher audience track.

The web designer thus obtained the following feedback from the adaptation strategy as it was defined in the previous sub section:

- The node ‘Researchers’ of the audience track ‘Researcher Audience Track’ was detected to be missing for the audience class ‘Student’

This result is discussed in the next sub section:

Discussion

Let’s first consider the two tables given in the previous subsection, and make some general observations. Notice first off all that, in absolute numbers, there were a lot more students visiting the considered nodes compared to researchers (i.e. 736 total visits for students, 125 total visits for researchers, and thus also less foreign visits for researchers (i.e. 28 for researchers compared to 4 for researchers). However, in percentages, researchers and students more or less equally visited foreign nodes: 3,8% for students compared to 3,2% for researchers.

Another observation is that in both audience tracks, the visits to the different nodes are partitioned: one group of nodes is visited a lot and a second group is visited less. For each audience class, there is also a group in between. More particularly, for the student audience class courses ‘Inleiding tot databanken’ and ‘Inleiding tot de Informatica’ are most frequently visited. These two nodes account for respectively 277 and 205 visits, totalling 68% of the total amount of student nodes visited by members of the student audience class. The middle group consists of only one node, the ‘Design methods for Internet-based Systems’ course node with 111 visits, totalling 15,6%. The group of least visited student nodes are the course nodes ‘Communicatievaardigheden voor Informatici’ and ‘User Aspects of Software Systems’ and the ‘CurrentStage&Thesis’, ‘PastStage&Thesis’ and NewStage&Thesis nodes (16,2%). For the researcher audience class the nodes ‘Publications’ and ‘Researchers’ are most visited (74,4%); the node ‘Research Topics’ forms the middle group (12.8%); the nodes ‘Visitors’, ‘Pastprojects’ and ‘Ongoingprojects’ are least visited (12,8 %). In the calculation of the threshold to detect missing information, the choice for median as central tendency and median absolute deviation as a measure of spread are thus well motivated, as both are robust and thus less influenced by outliers and extreme values (i.e. in our case, the group of nodes visited most). Using *mean* as central tendency for example would result in a higher influence of the most frequently visited group of nodes (i.e. the central tendency would go up) causing the threshold to rise. However, both for the student and the researcher audience class, the group of nodes most frequently visited only consist of a small amount of the total amount of nodes considered. Further research and experiments should be performed, combining statistical analysis with user feedback, to determine which threshold gives the most satisfying results (possibly different thresholds in different cases). A double strategy may also be considered, where automatic adaptation is performed based on a more rigid threshold and an alert to the web designer is given based on a more relaxed threshold.

Considering the fact that the experimental WISE web site has been designed by experts and thus the audience driven navigation structure was well thought through, it is not surprising that only one missing node was found. The ‘researchers’ node, identified to be missing in the student audience track, can be explained by the fact that students consulting stage and thesis related information (in the student audience track) are also interested in (information about) the researchers of WISE.

Chapter 8: Conclusion

"The whole problem with the world is that fools and fanatics are always so certain of themselves, but wiser people so full of doubts."
(Bertrand Russell)

In the previous chapter, the validation of the work presented in this dissertation was described. A prototype implementation to generate the actual web site from the WSDM design models was explained. This implementation was extended to support adaptive behaviour as described in this dissertation and an implementation architecture was proposed. Subsequently, a case study has been presented and results on two adaptation strategies performed in the context of this case study were presented.

In this final chapter, we reflect on the results of this dissertation. We start by summarizing the work presented (section 8.1). Next, we present the main contributions and achievements, thereby referring to the problem statement as described in the introduction (section 8.2). We then list the limitations of the work (section 8.3). Finally (in section 8.4), we present possible future work.

8.1 Summary

In this dissertation, an extension to a web design method is presented. The extension allows design time specification of runtime adaptation based on web site usage information. The aim of the adaptation is to provide *optimization* (i.e. improvement of the web site for *all* users), as opposed to *personalization* (i.e. adjustment of the web site for the current user).

The work is presented in the framework of WSDM, an audience driven web site design method. To provide a solid foundation for our work, the relevant design models of WSDM were first formalized. As an extension to this formalization, the Web site Overlay Model was defined to support the storage (at runtime) of (web site usage) information. Based on the WSDM formalization and the Web site Overlay Model, basic information storage and model transformation operations were defined. The basic information storage operations allow population of the Web site Overlay Model. The basic model transformation operations allow manipulation of the relevant web site design models (i.e. the navigation model and the page design model) to allow for adaptation. In this way, in the design models, links, nodes and pages can be added or deleted, chunks can be connected or disconnected from nodes and node can be removed or added to pages. These operations provide the possible basic structural and navigational changes to a web site design. Using these extensions, it is possible to provide a mechanism that allows a web designer to specify about which (design) element information needs to be stored, when and how to update this information and when and how to transform the navigation and page design models. This mechanism is a high level rule based adaptation specification language ASL.

A part of this language deals with the population and modification of the Web site Overlay Model (e.g. adding tracking variables to design elements, updating the value of tracking variables) and with the invocation of basic design model transformations. This part of the language can be one-to-one mapped to the corresponding operations defined on the formal model. The language furthermore offers the designer the possibility to specify:

- **Adaptation Strategies and Policies**: Adaptation Strategies specify *which* adaptive behaviour needs to be performed; Adaptation Policies specify *when* the Adaptation Strategies need to be performed. Specification of *which* adaptive behaviour needs to be performed is done by means of rules (see further). Specification of *when* adaptation strategies need to be performed is done by means of events (see next point).
- **Events**: events will trigger some adaptation strategy. Events can be user generated (e.g. clicking a link, visiting a node, page, …) or system generated (elapse of a time interval, tracking variables attaining a certain value, …).

- **Rules**: rules are the basic ingredients to specify Adaptation Strategies. Rules can be used to specify iteration, conditional execution of an action, add tracking variables, declare variables, assign values to (tracking) variables or perform pre-defined operations.
- **Control flow**: control flow operations (e.g. loop-constructs, if-then) allow the designer to use some programming logic in an Adaptation Strategy.
- **Expressions and set-expressions**: expressions allow the designer to express calculations, using tracking variable values and arbitrary values (e.g. thresholds). For this purpose, some basic arithmetic and statistical operations are provided. Set expressions allow the designer to express calculations on sets. Standard set theory operators (e.g. division, union, …) are provided, some set creators (e.g. chunksFromNode, …) and some set operations (i.e. mapping, filtering, sorting) are also provided.

In summary, the Adaptation Specification Language allows the designer to specify, at design time, which adaptation (on the design models) can be performed at runtime. To illustrate the power of the Adaptation Specification Language, three useful Adaptation Strategies were specified:

- **Promotion and Demotion:** Promotion of a node makes the node easier to find for a user by moving it closer to the root (e.g. homepage) of the web site. Demotion on the other hand, reduces navigation choices by moving a node further away from the root (e.g. homepage) of the web site. Promotion and demotion is based on the popularity/un-popularity of the node (i.e. a high or low amount of accesses).
- **Re-ordering sequential information (gathering):** re-ordering a set of sequentially linked nodes is aimed at moving bottlenecks (i.e. node(s) in the sequence where *a lot of* users abandon the sequence) towards the end of the sequence. This technique can only be applied when there is no dependency between the nodes in the sequence, e.g. a user doesn't need information of one node before he is able to understand the next.
- **Validation of Audience Driven Design:** validation of an audience driven design exploits its typical navigation structure. In an audience driven design, for each group of users[96], called an *audience class*, a dedicated navigation path, called an *audience track*, is constructed in the navigation model. In their audience track, the users of the corresponding audience class can find all and only the information and functionality relevant for them. Validation of an audience driven navigation structure is done by using statistical techniques to identify missing or superfluous information. Once missing information is identified, it can be added to the audience track in which it is missing. If superfluous information is detected, the information can either be removed, or an alert to the web designer can be given. The designer can then decide to remove the information if indeed he perceives it to be un-necessary, or leave it in place if he considers the information essential for that audience track.

Each of these Adaptation Strategies was illustrated with an example, and the implementation using ASL was given.

Finally, to validate the principles and applicability of our approach, a prototype implementation was made, and an experiment was set up. The implementation showed that it is possible to generate an actual web site, using a WSDM design. Furthermore, two of the mentioned adaptation strategies, namely promotion and detection of missing information, were implemented and validated. The results of experiment show that the promotion links that were adaptively added proved to be usable for the visitors. Concretely, two nodes, one for each audience track, were promoted to the root of the web site (i.e. the homepage). One month after the promotion links were added, analysis of collected web site usage information showed that for the first promoted node, 70% of the users utilised the promotion

[96] Users are grouped together based on the same requirements.

link; for the second promoted node, 57% of the visitors used the promotion link. Using the algorithm for validating the audience driven approach, one node was detected to be missing. Although somewhat unexpected, the existence of the missing node could be intuitively explained.

8.2 Contributions and Achievements

In this section we summarize the contributions and achievements that are the result of this dissertation.

In our opinion, the most significant contribution to the web engineering research field is the introduction of flexible design support for (runtime) adaptive behaviour in the context of *optimization,* based on the usage of the web site. It was claimed, and demonstrated, that such support can be used:

1. **To anticipate and react on runtime information during design:** illustrated by specifying the adaptive behaviour of promoting the most popular nodes of each audience class to the root of the web site (i.e. the homepage)
2. **To automatically evaluate and select among design alternatives:** illustrated by specifying, in ASL, the demotion of a node, resulting in possible relocation of an existing audience track as a audience sub track of another audience track
3. **To detect and correct design flaws in web site design:** illustrated by specifying, in ASL, an adaptation strategy to identify and correct for missing and superfluous information in an audience driven design
4. **To better tailor the web site to satisfy business requirements:** illustrated by specifying, in ASL, re-ordering of a sequential information gathering process if a bottleneck was detected

Another valuable contribution is the specification of three adaptation strategies, which clearly demonstrate the applicability and usefulness of design time adaptation specifications based on web site usage information and also illustrate the power of ASL.

Also worth highlighting is the fact that the way the adaptation is specified in this dissertation provides a complete decoupling of the adaptation aspects from the regular (non-adaptive) web design aspects. This clear separation of design concerns enables the designer to consider adaptation separately, thereby reducing the complexity of specifying the complete system. Adaptation could even be specified at a later stage, after the web site has already been deployed. This does not necessarily need to be done by the same web designer; an ASL expert could for example devise suitable adaptation strategies. Furthermore, as the specification of adaptation is not intertwined with the regular web site design, it allows easy re-use of adaptation strategies for different web sites.

To realise the proposed approach, some other important research artefacts were produced. The most important one is the formalization of WSDM. Not only did it provide a solid foundation for the work presented in this dissertation, it is also usable to formally found any future WSDM extension. For example, extending WSDM with evolution support could be based on this formalization. A second important artefact is the prototype implementation for the WSDM approach and the Adaptation Specification Language evaluator. Both were built in such a way that they can be easily re-used for other prototype implementations regarding WSDM.

8.3 Limitations

In every major research effort, such as this dissertation, boundaries to the work have to be set and limitations accepted. The following boundaries and limitations apply to this dissertation:

- **WSDM Specific:** The presented work was done in the framework of WSDM, and is thus specific to it. More in particular, the Adaptation Specification Language acts upon WSDM design models, and can thus not be re-used, as it is, for other design methods. However, as discussed in Chapter 2, section 2.3 (Web Site Design Methods), most web design methods take a similar approach to web design. As WSDM, they distinguish between a data modelling, navigation modelling and presentation modelling phase. Since the work described in this dissertation uses exactly these models, the general principles on which this work is founded are thus well applicable for other approaches as well. More concretely, it is feasible to modify the Adaptation Specification Language (slightly) to provide the basic operations to transform the internal models of other design methods.

- **Limitations of the formal specification:** The WSDM formalization was done only to an extent sufficient to found the research on adaptation on. Therefore, some phases and design outputs of WSDM are not included in the formal specification given in this dissertation. Task models, object chunk parameters, functionality and high level presentation concepts were not formalized.
- **Limitations of the adaptive behaviour:** Adaptive behaviour as described in this dissertation only concerns adaptation of structure (i.e. manipulation of object chunks, nodes, pages) and navigation (i.e. removing/adding/moving links). Content adaptation (i.e. adding/removing object types in object chunks, modifying lexical object type values) or presentation adaptation (i.e. adjusting look & feel and layout) is not considered because it was not the focus of this work.
- **Limitations of the validation and the experiments:** in the validation chapter, a case study was set up and some experiments were performed. From this validation, several conclusions could be drawn:
    1. It is possible to generate, from a WSDM design, the actual implementation of the web site
    2. The Adaptation Specification Language as it was described in this dissertation, can be implemented. Furthermore, the case study showed that it is feasible to adapt, at runtime, the web site using design time specifications.
    3. The results from the experiments furthermore suggested that the performed adaptations are useful for the visitors.

  Although satisfactory results were obtained from the experiments, it should be noted that the subject of the case study, a university research lab web site, is rather small scaled both in web site-size as in amount of visitors. Further experiments on larger-scaled web sites, spawning a longer time period, would strengthen the obtained results. It should also be noted that an implementation and experiments, even large-scaled, can never be a formal proof of applicability and correctness of any approach.

8.4 Future Work

The work presented in this dissertation provides ample opportunity for further work. Some concern straight-forward elaborations or extensions to the current approach; others are more challenging: they combine existing research area's or/and may even open up new research (sub) areas.

One straight forward elaboration concerns the experiments. As already mentioned in section 8.3 (Limitations), further large-scale experiments would strengthen the results of this dissertation. It is our conjecture that, with more web site usage information available, results will only improve. In fact, large-scale experiments would enable us to use more advanced statistical techniques, which are better suited for handling large datasets. Other methods to calculate thresholds (for example for detecting superfluous information when validating an audience driven design) can also be applied, and different results can be compared in order to fine-tune the particular adaptation strategies. Further experiments could also include usability studies to further examine the web site's usability and the user satisfaction after adaptive changes were made.

Another straight forward elaboration concerns the information gathering specifications**:** although some programming logic can be expressed when specifying information gathering rules (e.g. gather access information only for a certain group of users; accesses by other users are ignored), there is room for improvement. For example, navigation history within a session (i.e. from which other page did the visitor originate when arriving at a particular page) could be made explicitly available to information gathering rules. It needs to be mentioned however, that navigation history information does not always reveal actual paths followed by the users. In particular, a visitor may request multiple pages, yet consult them in a different order then they were requested (thus suggesting a false path).

Another small scale elaboration consists of support for session-based adaptation. Adaptation as it is discussed in this dissertation is aimed at accommodating *all* users (i.e. optimization). Session-based information, which is currently not easily identifiable, could be used for example to present (the current user with) recommendations. In fact, this endeavour has already been undertaken by a thesis student under the supervision of the author [Coolbrandt, 2005]. It turned out that only limited additions were required: the current sessionID was made explicitly available in ASL and could subsequently be used to 'tag' tracking variables (in a similar way as tracking variables were made relative to an audience class in the audience driven validation adaptation strategy). The second adjustment was the introduction of session-relative adaptive actions (i.e. adaptive actions now destructively change the WSDM design models; session-relative adaptive action do not modify the (global) WSDM design models, yet they (temporarily) adjust them only for the current session). Using these slight modifications to the Adaptation Specification Language, the K-Nearest Neighbour and naive Bayes classifier based link recommendations (based on [Mitchell, 1997] and [Rish, 2001]) were successfully specified.

A major extension to the work described in this dissertation includes support for personalization. In the vein of the previous paragraphs, personalization as it is described in Chapter 2 (Background and Related Work) (i.e. adapting the web site for the current user, based on his user model), could be implemented. This would require session-based adaptive actions (as described above) and support for a user model. This user model would probably need to be more fine-grained in storing relevant information compared to the information currently stored in the Web site Overlay Model. E.g. it would make sense not to consider the object chunks atomic entities, as it was done in the current approach. This would enable us to store knowledge of the user for individual concepts. Furthermore, content-adaptation (i.e. on object chunk level) would improve the quality of possible personalization adaptation.

It is the fact that today, and certainly in the future, Web Engineering goes beyond the traditional discipline of Information Systems (i.e. storing, recording and disseminating information, e.g. [Hirschheim et al, 1995]) using the World Wide Web as a medium. While Web Engineering has benefited from Software Engineering and Information Systems research and techniques, this dissertation reaches out to the discipline of Programming Language Engineering by introducing ASL, a rule-based adaptation language, to complement typical Software Engineering and Information Systems techniques. Further cross-fertilization is a challenge for both research communities, and would contribute to the quality and results obtained in both. Applied to the work of this dissertation, the author sees opportunities for the application of declarative and aspect oriented programming (and modelling). First of all, it should be noted that ASL is a first attempt to incorporated programming technology to support adaptation in web engineering. As for any programming language, improvements both in form and expressiveness are foreseeable in future versions of ASL. Possibly the most promising advance would be the use of the declarative programming paradigm. As specified now, the Adaptation Specification Language is mainly imperative, i.e. the programming logic is totally specified by the designer when implementing an adaptation strategy. Both the condition triggering adaptation (i.e. the "condition" of an ECA rule), and the actual adaptation (i.e. the "action" of an ECA rule) are imperatively specified. Although ASL benefits from the strengths of imperative programming (total control over the programming logic), it also suffers from its drawbacks (everything needs to be 'hard-coded'). In particular, specifying complex generic browsing patterns on the navigation model is difficult and sometimes even not possible. Consider for example the related child pattern, described by the author in [Casteleyn et al, 2004a]. This pattern occurs when (links to) a group of related items (i.e. pages) are offered from one single parent page, yet the related items are not linked themselves (i.e. there is no 'next' link). If the visitor wants to visit all items, he is forced to repeatedly go back to the parent page to gain access to the next item. Detecting the occurrence of such patterns is yet impossible in ASL. Declarative programming could provide a solution here. Declarative programming describes *what* needs to be detected/done, but leaves *how* it should be done to the interpreter. When using a declarative programming paradigm patterns can be described in a more generic way, i.e. the actual occurrence of the pattern needs not to be 'hard-coded'. Applied to describing conditions in ASL, a declarative approach could not only make it easier to describe complex patterns, it could also improve the (current) expressiveness. A declarative description of a browsing pattern might also be a more intuitive. This however, is a matter of personal taste and experience, and the question whether or not specification of conditions using a declarative approach would be easier to grasp for a web designer should be examined. On the downside, we should mention a possible decrease in performance. However, given the fact that adaptation strategies are not performed often and can be done independently of the running web application (i.e. in parallel), this is an acceptable loss. When it comes to the action part (i.e. the actual adaptation), it is doubtful if ASL would benefit from a declarative approach. Due to the lack of mechanisms to store global state, and thus changes to it, declarative programming typically performs poorly in describing state-based applications. In particular, the transformations on the navigation and site structure models, which comprise the core adaptation as described in this dissertation, cannot be easily expressed using declarative programming language. Here, the imperative approach ASL currently used is certainly more intuitive. In conclusion, we can state that next generation ASL-like languages could benefit from a mixed approach, using declarative programming techniques to specify conditions and browsing pattern detection (and possible easy some language constructs, such as the for-each), and an imperative approach to specify the actual adaptation (i.e. the model transformations).

As already mentioned in the foregoing discussion, adaptation in ASL is now to some extend 'hard-coded': the designer needs to specify, imperatively, when adaptation needs to be performed, how it needs to be performed and where it needs to be performed (i.e. on which design elements browsing patterns can be detected). To do so, the designer uses his implicit knowledge and experience in web design and his explicit knowledge of the navigational model. A major challenge lies in generalizing and automating the process of adaptation. This requires the description of adaptation strategies in a more abstract way (e.g., independent of the specific navigational model under consideration), and more importantly, a strategy to decide when and where to apply which adaptation strategies. Possibly, different adaptation strategies would be applicable/appropriate for different (kind of) websites, or for different website organizations (i.e. data driven, organization driven, …). A deep understanding on website organization, and site navigation structures, is thus required. Unfortunately, there is very few work available (guidelines, principles) on how to devise an efficient, easy to grasp site organization, and which site organization fits best with a particular kind of website. Gathering such knowledge, and making it explicit is a primary requirement to automate selection and application of adaptation strategies. Moreover, the implicit knowledge of the role of different design elements (e.g. a certain page can for example serve as an information provider, or as a hub that does not contain any information itself but simply re-directs the visitor) would need to be made explicit as well. Incorporation of meta-information when performing adaptation could prove useful for this purpose. Based on this meta-information, adaptation strategies could exploit the knowledge now implicitly present in the designers mind to decide where to apply which adaptation strategy. This could also ease automation.

Another research result from the Programming Technology community from which Web Engineering in general, and adaptation in particular, could benefit is Aspect Oriented Programming [Kiczales et al, 1997]. Much as in Web Engineering, where it is the aim to let the designer focus on one design concern at a time (i.e. data/functionality modelling, navigation, presentation, implementation), Aspect Oriented Programming aims to identify and separate (cross-cutting) design concerns to allow a better separation of concerns, thereby increasing the ease and reducing complexity of software development. Therefore, Aspect Oriented Modelling and Programming techniques could be applied to further pursue separation of concerns in the web design cycle and to cleanly separate adaptation concerns from the regular web design, as it was done in this dissertation. The AOP hyperslice approach [Ossher and Tarr, 2001], where each identified concern is specified in a separate 'slice' and the different slices are subsequently (automatically) woven together, seems best applicable. In addition, the use of Aspect Oriented techniques could also allow to easily add adaptation to websites that were not designed for this. In this way, Web Engineering can benefit from the existing AOP expertise in general, and existing hyperslice implementations (i.e. HyperJ [Ossher and Tarr, 2001]) and weaving algorithms in particular.

The work described in this dissertation also lends itself perfectly to combine with other research areas. Some challenging interesting long-term future work can be formulated doing so.

Combining adaptation with evolution is a promising and feasible research avenue. As already mention in Chapter 1 (Introduction), the basic model transformation operations, able to modify WSDM design models, can be useful in other contexts. One of this is the domain of ontology evolution concerned with the evolutionary nature of ontologies, both at concept and at instance level. As the WSDM design models are defined as an OWL ontology (called the WSDMOntology)[97], and a particular WSDM web site design is an instantiation of the WSDMOntology, they may act as the subject for ontology evolution techniques. With this in mind, we can consider the basic model transformation operations defined in the context of ASL, merely as initiators of evolution at the instance level of the WSDMOntology. If design changes are made using the basic model transformation (either manually by the designer when iterating over the design, or automatically due to execution of adaptation strategies), the evolution of the changes could be recorded. This would effectively allow to track the evolution of the web site (i.e. both manual and automatic changes). Contemplating this situation with regard to adaptation, we could say that tracking the evolution of adaptation adds history to the (performed) adaptation. This yields ample opportunities. Making the history available from within ASL could allow straight-forward reversal of adaptation strategies, or even selective reversal of some adaptive actions. Exploiting the adaptation history, reasoning about the kind of adaptation made or the evolution of (performed) adaptation, are open research areas.

Combining the basics of the Adaptation Specification Language for Quality analysis is another open issue. In ASL, the designer is given the possibility to specify exactly of which design elements relevant usage information needs to be tracked. Using some programming logic, he can fine-tune the conditions under which information is tracked (e.g. only for a specific audience class) and as explained earlier in this section, information gathering specification could be improved (e.g. using declarative programming). As also described earlier in this section, the mechanism to detect user

[97] See Chapter 7, Validation

access patterns is also susceptible for improvement. Combining these two improvements, ASL could be usable as the basis for a quality evaluation framework concerning structure and navigation. When bringing in the evolution aspect, described in the previous bullet, evolution of the web site could be evaluated within the quality evaluation framework. Both the web site itself, its evolution through adaptation, and the evolution of the performed adaptation itself, could be evaluated within such a framework. Rigid evaluation criteria, and possible conclusions to be drawn from evaluation of adaptation and the evolution of adaptation, are challenging open research questions.

Appendix 1: ASL BNF Specification

```
<adaptationPolicy> ::=
        (<script> ';')*
        <adaptationSpecification> ';' (<adaptationSpecification> ';')*

<adaptationSpecification> ::= 'when' <trigger> 'do' <adaptationStrategy>

<trigger> ::= <timeEvent>
<trigger> ::= <systemEvent>
<trigger> ::= <userEvent> ['on' <reference>]
<trigger> ::= <condition>

<userEvent> ::= 'click' | 'sessionStart' | 'sessionEnd' | 'load'

<systemEvent> ::= 'initialization'

<timeEvent>::= <dateSpecifier> | ['every'] <timeExpression> ['from now']

<dateSpecifier>       ::= <number> <month> <number>

<month>               ::= 'January'| 'February' | 'March' | 'April' | 'May' | 'June' | 'July' |
                          'September' | 'October' | 'November' | 'December'

<timeExpression>      ::= <timePrimitive> <timePrimitive>*

<timePrimitive>       ::= <number> <timeUnit>
<timePrimitive>       ::= <day>
<timePrimitive>       ::= <month>

<timeUnit>            ::= 'seconds' | 'minutes' | 'hours' | 'days' | 'weeks' | 'months' |
                          'years'

<day>                 ::= 'Monday' | 'Tuesday' | 'Wednesday' | 'Thursday' | 'Friday' |
                          'Saturday' | 'Sunday'

<adaptationStrategy> ::= <ruleSequence> | <rule>
<ruleSequence> ::= 'begin' <rule>  (';' <rule>)* 'end'

<rule> ::= <foreachRule> | <simpleRule>

<foreachRule> ::= 'forEach' <setIterator> <setIterator>* ':'  <adaptationStrategy>
<setIterator> ::= <reference> 'in' <setExpression>
```

<simpleRule> ::= <ifStatement> | <trackingVariableStatement> | <letStatement> | <assignment> | <setLetStatement> | <setAssignment> | <nativeOperation> | <callStatement>

<ifStatement> ::= ‘**if**’ <condition> ‘**then**’ <adaptationStrategy>

<letStatement> ::= ‘**let**’ <variable> [‘**be**’ <expression>]

<setLetStatement> ::= ‘**let**’ <setVariable> [‘**be**’ <setExpression>]

<addTrackingVariable> ::= **‘addTrackingVariable** <trackingVariable>
[‘**initialized on’ <**expression>]

<monitorStatement> ::= ‘**monitor**’ <userEvent> [‘**on**’ <reference>]
[‘**if**’ <condition>] ‘**do**’ <trackingVariableAssignmentSequence>

<trackingVariableAssignmentSequence> ::=
<trackingVariableAssignment> |
‘**begin**’ <trackingVariableAssignment> (‘**;**’ <trackingVariableAssignment >)* ‘**end**’

<trackingVariableAssignment> ::= <trackingVariable> ‘**:=**’ <expression>

<assignment> ::= < trackingVariableAssignment>
<assignment> ::= <variable> ‘**:=**’ <expression>

<setAssignment> ::= <setVariable> ‘:=’ <setExpression>

<script> ::= ‘**script**’ <reference> ‘(’ [<parameter> (‘**,**’ <parameter>)*] ’)’ ‘**:**’
<adaptationStrategy>

<parameter> ::= <variable> | <setVariable>

<callStatement> ::= **‘call’** <reference> ‘(‘ [((<expression> | <setExpression>)’,’
(<expression> | <setExpression>)*)] ‘)’

<condition> ::= <expression>

<expression> ::= <comparand> (<comparator> <comparand>)*

<comparand> ::= <term> (<adder> <term>)*

<term> ::= <factor> (<multiplier> <factor>)*

<factor> ::= <primitive>
<factor> ::= <not> <primitive>

<primitive> ::= <number>
<primitive> ::= <boolean>
<primitive> ::= <reference>
<primitive> ::= <variable>

```
<primitive>       ::= <trackingVariable>
<primitive>       ::= <nativeFunction>
<primitive>       ::= <statisticalOperator>
<primitive>       ::= ‘(‘<expression>‘)’

<comparator>   ::= ‘<’ | ’=’ | ‘>’ | ‘<=’ | ‘>=’ | ‘!=’
<adder>           ::=  ‘OR’ | ‘+’ | ‘-‘
<multiplier>    ::= ‘AND’ | ‘/’ | ‘*’
<not>              ::=  ‘NOT’ | -

<trackingVariable> ::=  <reference> [ ‘[’<reference> (‘,’ <reference>)* ‘]’ ] ’.’ < reference >

<variable>       ::= <reference>
<setVariable>   ::= <capitalizedLetter> (<letter> | <digit>)*
<reference>     ::= (<nonCapitalizedLetter> | <digit>)* <letter> (<letter> | <digit>)*

<nativeSet> ::= ‘Nodes’ | ‘Chunks’ | ‘Links’ | ’Connections‘ | ’Pages’ |
                       ‘AudienceClasses’

<setCreator> ::=
           ‘nodesInAudienceTrack(‘<expression>’)’ |
           ‘chunksInAudienceTrack(‘<expression >’)’ |
           ‘chunksFromNode(‘<expression >’)’ |
           ‘nodesLinkedTo(‘<expression>, <expression >’)’ |
           ‘nodesLinkedFrom(‘<expression>, <expression >’)’ |
           ‘nodesFromChunk(‘<expression >’)’ |
           ‘ALL’ < expression > |
           ‘addElement(‘<setExpression>, <expression>’)’
           ‘addElementAtIndex(‘<setExpression>, <expression>, <expression> ’)’

<setOperation> ::= <setSorter> | <setFilter> | <setMap>

<setSorter> ::=  ‘SORT ON’ <reference> ‘:’ <trackingVariable>
<setFilter>  ::=  ‘FILTER ON’ <reference> ‘:’ <condition>
<setMap>    ::=  ‘MAP ON’ <reference> ‘:’ <expression>

<setExpression>           ::=  <setComparand> (<setComparator> <setComparand>)*

<setComparand>          ::= <setTerm> (<setAdder> <setTerm>)*

<setTerm>                  ::= <setPrimitive> (<setMultiplier> <setPrimitive>)*

<setPrimitive>             ::= <set>
<setPrimitive>             ::= ‘(‘<setExpression>‘)’
<setPrimitive>             ::= <setExpression> ‘[‘<setOperation> (‘;’ <setOperation>)*’]’

<set>    ::=         <nativeSet> |
                       <setCreator> |
                        ‘{‘ <expression>  [‘,’ <expression>]* ‘}’
```

```
<setComparator> ::= ’=’
<setAdder>      ::=  ‘DIFFERENCE’ | ‘+’ | ‘-‘
<setMultiplier> ::= ‘UNION’ | ‘INTERSECT’ | ‘*’ | ‘/’

<nativeOperation> ::=
                ‘deleteNode (‘<expression>’)’ |
                ‘addNode (‘<expression>’)’ |

                ‘connect (‘<expression>’, ‘<expression>’)’ |
                ‘disconnect (‘<expression>’, ‘<expression>’)’ |

                ‘deleteLink (‘<expression>’, ‘<expression>’ , ‘<expression>’)’ |
                ‘addLink (‘<expression>’, ‘<expression>’, ‘<expression>’)’
                        [as <reference>] |

                ‘addPage(‘<expression>’)’ |
                ‘deleteNodeFromPage(‘<expression>’, ‘<expression>’)’ |
                ‘addNodeToPage(‘<expression>’, ‘<expression>’)’ |

                 ‘alert(’(<expression> | <string> )* ‘)’

<nativeFunction> ::=
                ‘root(‘<expression>’)’   |   ‘audienceClass(‘<expression>’)   |
                ‘audienceTrack(‘<expression>’)   |   ‘currentAudienceClass()‘   |
                ‘inAudienceTrack?   (‘<expression>’,’   <expression>’)’   |
                ‘linked?(‘<expression>’,‘   <expression>’,’   <expression>’)’   |
                ‘project(‘<setExpression>’,’  <expression>’)’  |  ‘target(‘<expression>’)’  |
                ‘source(‘<expression>’)’ | ‘linktype(‘<expression>’)’

<statisticalOperators> ::=
                ‘average(’< setExpression >’)’ |
                ‘max(’<setExpression>’)’ |
                ‘min(’<setExpression>’)’ |
                ‘range(’< setExpression >’)’ |
                ‘mad(’<setExpression>’)’ |
                ‘median(’< setExpression >’)’ |
                ‘middle(’< setExpression >’)’ |
                ‘stdev(’<setExpression>’)’ |

                ‘length(‘<setExpression>’)’ |

<boolean>       ::=     ‘false’ | ‘true’

<string>        ::= ‘”’ (<letter> | <digit>)* <letter> (<letter> | <digit>)* ’”’

<letter> ::= < nonCapitalizedLetter> | <capitalizedLetter>

<nonCapitalizedLetter> ::= ‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘g’ | ‘h’ | ‘i’ | ‘j’ | ‘k’ | ‘l’ | ‘m’ | ‘n’ |
‘o’ | ‘p’ | ‘q’ | ‘r’ | ‘s’ | ‘t’ | ‘u’ | ‘v’ | ‘w’ | ‘x’ | ‘y’ | ‘z’
```

<capitalizedLetter> ::= ‘**A**’ | ‘**B**’ | ‘**C**’ | ‘**D**’ | ‘**E**’ | ‘**F**’ | ‘**G**’ | ‘**H**’ | ‘**I**’ | ‘**J**’ | ‘**K**’ | ‘**L**’ | ‘**M**’ | ‘**N**’ | ‘**O**’ | ‘**P**’ | ‘**Q**’ | ‘**R**’ | ‘**S**’ | ‘**T**’ | ‘**U**’ | ‘**V**’ | ‘**W**’ | ‘**X**’ | ‘**Y**’ | ‘**Z**’

<number> ::= <digit> <digit>*
<digit**>** ::= ‘**0**’ | ‘**1**’ | ‘**2**’ | ‘**3**’ | ‘**4**’ | ‘**5**’ | ‘**6**’ | ‘**7**’ | ‘**8**’ | ‘

References

Abrams, M. (1998). *World Wide Web - Beyond the Basics*, Prentice Hall, ISBN 0-13-954785-1

Backus, J. W., Wegstein, J. H., van Wijngaarden, A., Woodger, M., Bauer, F. L., Green, J., Katz, C, McCarthy, J., Perlis, A. J., Rutishauser, H., Samelson, K. and Vauquois, B. (1960). Revised Report on the Algorithmic Language ALGOL 60. In *Communications of the ACM*, Vol. 3 No.5, pages 299-314, ACM Press, ISSN 0001-0782

Barnett, V. and Lewis, T. (1994). *Outliers in Statistical Data (3rd edition).* John Wiley & Sons, ISBN 0471930946

Berners-Lee, T., Hendler and J., Lassila, O. (2001).The Semantic Web. In *Scientific American - May 2001*, pages 34-43

Bieber, M. (2000). Hypertext. In *Encyclopedia of Computer Science (4th Edition)*, pages 799-805, Nature Publishing Group

Brown, P.J. (1987). Turning ideas into products: The Guide system. In *Proceedings of the ACM Hypertext'87 Conference*, pages 33-40, ISBN 0-89791-340-x

Brusilovsky, P. (1996). Methods and techniques of adaptive hypermedia. In *User Modeling and User-Adapted Interaction*, 6 (2-3), pages 87-129, Springer Science+Business Media B.V, ISSN 0924-1868

Brusilovsky, P. (2001). Adaptive Hypermedia. In *User Modeling and User Adapted Interaction*, 11 (1/2), pages 87-110, Kluwer Academic Publishers

Brusilovsky, P. and Maybury, M. T. (2002). From adaptive hypermedia to adaptive Web. In *Communications of the ACM* **45** (5), pages 31-33, ACM Press, ISSN 0001-0782

Bush, V. (1945). As We May Think. In *The Atlantic Monthly, Volume 176. No. 1*, pages 101-108,

Cailiau, R. and Ashman, H. (1999). Hypertext in the Web — a history. In *ACM Computing Surveys (CSUR), Volume 31, Issue 4es*, Article No. 35, ACM Press, ISSN 0360-0300

Carr, B. and Goldstein, I. (1977). Overlays: A theory of modelling for computer aided instruction. In Technical Report AI Memo 406, MIT, Cambridge

Casteleyn, S., Garrigós, I. and De Troyer, O. (2005). Automatic Runtime Validation and Correction of the Navigational Design of Web Sites. In *Web Technologies Research and Development - APWeb 2005*, pages 453-463, Springer-Verlag, ISBN 3-540-25207-X

Casteleyn, S., Garrigós, I. and Plessers, P. (2004). Pattern Definition to Refine Navigation Structure in Hypermedia/Web Applications. *In Proceedings of the IADIS International Conference WWW/Internet 2004 (ICWI2004), Volume II*, pages 1199-1203, IADIS PRESS, ISBN 972-99353-0-0

Casteleyn, S., Garrigós, I. and De Troyer, O. (2004). Using Adaptive Techniques to Validate and Correct an Audience Driven Design of Web Sites. In *Proceedings of the ICWE 2004 Conference, Lecture Notes in Computer Science 3140*, pages 55-59, Springer, ISBN 3-540-22511-0

Casteleyn, S., De Troyer, O. and Brockmans, S. (2003). Design Time Support for Adaptive Behaviour in Web Sites. In *Proceedings of the 18th ACM Symposium on Applied Computing*, pages 1222 - 1228, ACM, ISBN 1-58113-624-2

Casteleyn, S. and De Troyer, O. (2002). Exploiting Link Types during the Web Site Design Process to Enhance Usability of Web Sites. In *Proceedings of the Second International Workshop on Web-Oriented Software Technologies, IWWOST 2002*

Casteleyn, S. and De Troyer, O. (2001). Structuring Web Sites Using Audience Class Hierarchies. In *Conceptual Modeling for New Information Systems Technologies, ER 2001 Workshops, HUMACS, DASWIS, ECOMO, and DAMA, Lecture Notes in Computer Science , Vol. 2465*, Springer-Verlag, ISBN 3-540-44-122-0

Chen, P.(1975). The entity-relationship model—toward a unified view of data. In *ACM Transactions on Database Systems (TODS) Volume 1 , Issue 1*, pages 9 - 36, ACM Press, ISSN 0362-5915

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S. and Matera, M.(2002). *Designing Data-Intensive Web Applications.* Morgan Kaufmann Publishers Inc., ISBN 1558608435

Ceri, S., Fraternali, P., Bongio, A. and Maurino, A. (2000). Modeling data entry and operations in WebML. In *Lecture Notes In Computer Science; Vol. 1997, Selected papers from the Third International Workshop WebDB 2000 on The World Wide Web and Databases*, pages 201-214, Springer-Verlag, ISBN 3-540-41826-1

Ceri, S., Fraternali, P., Maurino, A. and Paraboschi, S. (1999). One-To-One Personalization of Data-Intensive Web Sites. In *WebDB Workshop*

Ceri, S., Daniel, F. and Matera, M. (2003). Extending WebML for Modeling Multi-Channel Context-Aware Web Applications. In *Proc. of the WISE - MMIS'03 Workshop (Mobile Multi-channel Information Systems)*, pages 225-233, IEEE Press, ISBN 0-7695-2103-7

Ceri, S., Piero Fraternali, P. and Stefano, S. (1999). Data-Driven, One-To-One Web Site Generation for Data-Intensive Applications. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 615-626, Morgan Kaufmann, ISBN 1-55860-615-7

Coolbrandt, S. (2005). Design Support for Session-based Adaptation in WSDM: Link Recommendations, Master thesis, Vrije Universiteit Brussel

Cristea A.I. and Kinshuk, K. (2003). Considerations on LAOS, LAG and their Integration in MOT, In *ED-MEDIA'03 Volume 2003, Issue 1*, AACE Digital Library, pages 511-518

Dayal, U. (1988). Active Database management systems. In *Proceedings of the Third International Conference on Data and Knowledge Bases*, pages 150, 169, Publ. Morgan Kaufmann, ISBN 0-93413-95-8

De Bra, P., Aerts, A., Berden, B., De Lange, B., Rousseau, B., Santic, T., Smits, D. and Stash, N. (2003). AHA! The Adaptive Hypermedia Architecture. In *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, pp. 81-84, ISBN:1-58113-704-4

De Bra, P. (1999). Design Issues in Adaptive Hypermedia Application Development. In *Proceedings of the Second Workshop on Adaptive Systems and User Modeling on the World Wide Web*, pages 29-39

De Bra, P., Brusilovsky and P., Houben, G.-J. (1999). Adaptive Hypermedia: From Systems to Framework. In *ACM Computing Surveys, 31 (4es)*, ACM Press, ISSN 0360-0300

De Bra, P., Houben, G.J. and Kornatzky, Y. (1992). An Extensible Data Model for Hyperdocuments. In *4th ACM Conference on Hypertext*, pages 222-231, ACM Press, ISBN 0-89791-547-X

De Bra, P., Houben, G.J. and Wu, H., (1999). AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. In *Proceedings of the ACM Conference on Hypertext and Hypermedia*, pages 147-156, ACM Press, ISBN 1-58113-064-3

DeGroot, M.H. and Schervish, M.J. (2001). *Probability and Statistics (3rd edition)*. Addison Wesley, ISBN 0201524880

De Troyer, O. and Casteleyn, S. (2004). Designing Localized Web Sites. In *Proceedings of the 5th International Conference on Web Information Systems Engineering (WISE2004)*, pages 547-558, Springer-Verlag, ISBN 3-540-23894-8

De Troyer, O. and Casteleyn, S. (2003). Exploiting Link Types during the Conceptual Design of Web Sites. In *International Journal of Web Engineering Technology, Vol 1, No. 1*, pages 17-40, Inderscience, ISSN 1476-1289

De Troyer, O. and Casteleyn, S. (2003). Modeling Complex Processes for Web Applications using WSDM. In *Proceedings of the Third International Workshop on Web-Oriented Software Technologies* (held in conjunction with ICWE2003), IWWOST2003
(also on http://www.dsic.upv.es/~west/iwwost03/articles.htm)

De Troyer, O. and Casteleyn, S. (2001). The Conference Review System with WSDM. In *First International Workshop on Web-Oriented Software Technology, IWWOST'01*, online on http://www.dsic.upv.es/~west2001/iwwost01/

De Troyer, O., Plessers and P., Casteleyn, S. (2003). Conceptual View Integration for Audience Driven Web Design. In *CD-ROM Proceedings of the WWW2003 Conference, IW3C2* (also http://www2003.org/cdrom/html/poster/)

De Troyer, O., Plessers, P. and Casteleyn, S. (2003). Solving Semantic Conflicts in Adience Driven Web Design. In *Proceedings of the WWW/Internet 2003 Conference (ICWI 2003), Volume I*, pages 443 - 450, IADIS Press, ISBN 972-98947-1-X

De Troyer, O. and Leune, C (1998). WSDM: A User-Centered Design Method for Web Sites. In *Computer Networks and ISDN systems Volume 30, Proceedings of the 7th International World Wide Web Conference*, pages 85-94, Elsevier

Dieterich, H., Malinowski, U., Kühme, T., Schneider-Hufschmidt, M. (1994). State of the Art in Adaptive User Interfaces. In *Adaptive User Interfaces: Principles and Practice*, pages 13 - 48, Schneider-Hufschmidt & Al.

Dix, A., Finlay, J., Abowd, G. and Beale, R. (1998). *Human-Computer Interaction (second edition)*, Publ. Prentice Hall Europe, ISBN 0-13-239864-8

Dixon, W. J. (1950). Analysis of extreme values. In *Ann. Math. Stat., 21*, pages 488-506

Facca, F. M., Ceri S., Armani, J. and Demaldé, V. (2005). Building Reactive Web Applications, In *Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1058-1059, ACM Press, ISBN 1-59593-051-5

Fiala, Z., Frasincar, F., Hinz, M., Houben, G.J., Barna, P. and Meissner, K.(2004). Engineering the presentation layer of adaptable web information systems. In *Web Engineering 4th International Conference, ICWE 2004, volume 3140 of Lecture Notes in Computer Science*, pages 459-472, Springer, ISBN 3-540-22511-0

Frasincar, F., Barna, P., Houben, G.-J. and Fiala Z. (2004). Adaptation and reuse in designing web information systems. In *International Conference on Information Technology: Coding and Computing (ITCC'04)*, pages 387-291, IEEE Computer Society

Frasincar, F. and Houben, G.-J. (2002). Hypermedia presentation adaptation on the semantic web. In *Adaptive Hypermedia and Adaptive Web-Based Systems, Second International Conference, AH 2002, volume 2347 of Lecture Notes in Computer Science*, pages 133-142. Springer, ISBN 3-540-43737-1

Frasincar, F., Houben, G.-J. and Vdovjak R. (2002). Specification framework for engineering adaptive web applications. In *The Eleventh International World Wide Web Conference*, Web Engineering Track (CDROM Proceedings)

Furuta, R., Stotts, P.D. (1989). Separating Hypertext Content from Structure in Trellis. In *Proceedings of the UK Hypertext conference*, Intellect Ltd., pages 205-213, ISBN 1-871516-08-0

Güell, N., Schwabe, D. and Vilain, P. (2000). Modeling Interactions and Navigation in Web Applications. In *ER (Workshops)*, pages 115-127, ISBN 3-540-41073-2

Garrigós, I., Casteleyn, S. and Gómez, J. (2005). A Structured Approach to Personalize Web sites using the OO-H Personalization Framework. In *Web Technologies Research and Development - APWeb 2005*, pages 695-706, Springer-Verlag, ISBN 3-540-25207-X

Garrigós, I., Gómez, J. and Cachero, C. (2003). Modeling Dynamic Personalization in Web Applications. In *Third International Conference on Web Engineering (ICWE'03) - LNCS 2722*, pages 472-475, Springer, ISBN 3-540-40522-4

Gómez, J., Cachero, C. and Pastor, O. (2001). Conceptual Modeling of Device-Independent Web Applications. In *IEEE Multimedia Special Issue on Web Engineering*, pages 26-39, IEEE Computer Society Press, ISSN 1070-986X

Gómez, J., Cachero, C. and Pastor, O. (2000). Extending an Object-Oriented Conceptual Modeling Approach to Web Application Design. In *Proceedings of the 12th International Conference on Advanced Information Systems Engineering - LNCS 1789*, pages 79-93, Springer-Verlag, ISBN:3-540-67630-9

Grubbs, F. (1969). Procedures for Detecting Outlying Observations in Samples, In *Technometrics, Vol 11, No 1*, pages 1-21

Halasz, F.G. (1987). Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. In *Proceedings of the ACM Hypertext'87 Conference*, pages 345-365, ISBN 0-89791-340-x

Halasz, F. and Schwartz, M. (1990). The Dexter Reference Model. In *Proceedings of the NIST Hypertext Standardization Workshop*, pages 95-133

Halasz, F. and Schwartz, M., (1994). The Dexter Reference Model. In *Communications of the ACM, Vol 37, issue 2 (February)*, pages 30- 39, ACM Press, ISSN 0001-0782

Halpin, T.A. (2001). *Information Modeling and Relational Databases*, Morgan Kaufmann Publishers, ISBN 1-55860-672-6

Hardman, L., Bulterman, D.C.A. and Guido van Rossum (1994). The Amsterdam hypermedia model: adding time and context to the Dexter model. In *Communications of the ACM Volume 3 , Issue 2*, pages 50 - 62, ACM Press, ISSN 0001-0782

Hawkins D. (1980). *Identification of Outliers*, Chapman and Hall, ISBN 041221900X

Hilbert, D.M., Redmiles, D.F. (2000). Extracting usability information from user interface events. In *ACM Comput. Surv. 32(4)*, pages 384-421

Hirschheim, R. & Klein, H. K. & Lyytinen, K (1995). *Information systems development and data modeling: conceptual and philosophical foundations*. Cambridge University Press, ISBN 0-521-37369-7

Houben, G.-J., Frasincar, F., Barna, P. and Vdovjak, R. (2004). Engineering the presentation layer of adaptable web information systems. In *Web Engineering 4th International Conference, ICWE 2004, volume 3140 of Lecture Notes in Computer Science*, pages 60-73, Springer, ISBN 3-540-22511-0.

Isakowitz, T., Stohr, E. A. and Balasubramanian, P. (1995). RMM: A Methodology for Structured Hypermedia Design. In *Communications of the ACM 38(8)*, pages 34-44, ACM Press, ISSN 0001-0782

Ivory, M. and Hearst, M. (2001). The state of the art in automating usability evaluation. In *ACM Computing Surveys, 33(4)*, pages 470-516, ACM Press , ISSN 0360-0300

Jacobson, I., Booch, G. and Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison Wesley, ISBN 0201571692

Jin, Y., Xu, S., Decker, S., Wiederhold, G. (2002). Managing Web Sites with OntoWebber. In *Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology*, pages 766 – 768, ISBN 3-540-43324-4

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J., Irwin, J. (1997). Aspect-Oriented Programming. In *ECOOP'97 – Object-Oriented Programming, 11th European Conference*, pages 220 – 242, ISBN 3-540-63089-9

Klapsing, R. and Neumann, G. (2000). Applying the resource description framework to web engineering. In *First International Conference on Electronic Commerce and Web Technologies (EC-Web 2000)*, pages 229 – 238, ISBN 3-540-67981-2

Kobsa, A., Koenemann, J. and Pohl, W. (2001). Personalised hypermedia presentation techniques for improving online customer relationships Source. In *The Knowledge Engineering Review archive Volume 16 , Issue 2 (March 2001)*, pages 111 – 155, Cambridge University Press, ISSN: 0269-8889

Koch, N. (2001). *Software Engineering for Adaptive Hypermedia Systems, Reference Model, Modeling Techniques and Development Process*, PhD Thesis, Verlag UNI-DRUCK, ISBN 3-87821-318-2

Koch, N., Kraus, A. and Hennicker, R. (2001). The Authoring Process of the UML-based Web Engineering Approach. In *Proceedings of the 1st International Workshop on Web-Oriented Software Technology.*

Lei, Y., Motta, E., Domingue, J. (2005). OntoWeaver: an Ontology-based Approach to the Design of Data-intensive Web Sites. In *Journal of Web Engineering, Volume 4, number 2 (June 2004)*, pages 244 - 262

McDaid, J. (1991). Breaking Frames: Hyper-Mass Media. In *Hypertext/Hypermedia Handbook*, New York, pages 445-458, Intertext Publications/McGraw-Hill Publishing Company, ISBN 0-07-016622-6.

Mérida, D., Fabregat, R. and Marzo, J.L. (2002). SHAAD: Adaptable, Adaptive and Dynamic Hypermedia System for content delivery. In *Proceedings of Workshop on Adaptive Systems for Web Based Education (WASWE2002)*

Mitchell, T. M.. (1997). *Machine Learning*. McGraw-Hill Science/Engineering/Math., ISBN 0070428077

Murugesan, S., Deshpande, Y., Hansen, S., and Ginige, A. (2001). Web engineering: A new discipline for development of web-based systems. In *Web Engineering, Software Engineering and Web Application Development, volume 2016 of Lecture Notes in Computer Science*, pages 3–13, Springer, ISBN 3-540-42130-0.

Nelson, T.H. (1965). Complex information processing: a file structure for the complex, the changing and the indeterminate. In *Proceedings of the 1965 20th National Conference*, pages 84-100, ACM Press

Nielsen, J. (1995). *Multimedia and Hypertext: The Internet and Beyond*, Morgan Kaufmann, ISBN 0-12-518408-5

Nielsen, J. (1992) Finding usability problems through heuristic evaluation. In *Proceedings of the SIGCHI conference on Human Factors and Computer Systems*, pages 373–380, ISBN:0-89791-513-5

Ossher, H., Tarr, P. Hyper/JTM: Multi-Dimensional Separation of Concerns for JavaTM. In *Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001*, pages 821-822, ISBN 0-7695-1050-7

Pastor, O., Fons, J., and Pelechano, V. (2003). Oows: A method to develop web applications from web-oriented conceptual models. In *International Workshop on Web-Oriented Software Technology (IWWOST 2003)*, pages 65 - 70

Paterno, M., Mancini, C. and Meniconi, S. (1997). ConcurTaskTrees: a Diagrammatic Notation for Specifying Task Models. In *Proceedings of INTERACT 97*, pages 362-366, Chapman & Hall, ISBN 0-412-80950-8

Paterno, F. (1999). *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, ISBN 1-85233-155-0

Perkowitz, M. and Etzioni, O. (1997). Adaptive web sites: an AI challenge. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 16-23, Morgan Kaufmann

Perkowitz, M. and Etzioni O (1997). Adaptive Web Sites: Automatically Learning from User Access Patterns (poster). In *Proceedings of WWW6 Conference*

Plessers, P., Casteleyn, S., Yesilada, Y., De Troyer, O., Stevens, R., Harper, S. and Goble, C. (2005). Accessibility: A Web Engineering Approach. In *Proceedings of the 14th International World Wide Web Conference (WWW2005)*, pages 353-362, ACM Press, ISBN 1-59593-046-9

Rish, I. (2001). An empirical study of the naive Bayes classifier". In *IJCAI Workshop on Empirical Methods in Artificial Intelligence*

Rossi, G., Schwabe, D. and Guimarães, R. (2001). Designing personalized web applications. In *WWW 2001*, pages 275-284, ISBN 1-58113-348-0

Rosner, B. (1975). On the detection of many outliers. In *Technometrics 17*, pp. 221-227

Scharl, A. (2001). A Classification of Web Adaptivity: Tailoring Content and Navigational Systems of Advanced Web Applications. In *Web Engineering, Software Engineering and Web Application Development*, pages 156-169, ISBN 3-540-42130-0

Schwabe, D. and Moura, S. (2003). Interface development for hypermedia applications in the semantic web. In *Proceedings of the WebMedia & LA-Web 2004 Joint Conference 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress - Volume 00,* pages 106-113, IEEE Computer Society, ISBN 0-7695-2237-8

Schwabe, D. and Rossi, G. (1995). The Object-Oriented Hypermedia Design Model. In *Communications of the ACM 38(8),* pages 45-46, ACM Press, ISSN 0001-0782

Schwabe, D. and Rossi, G. (1998). An object oriented approach to web-based applications design. In *Theory and Practice of Object Systems, 4(4)*, pages 207-225, John Wiley & Sons, Inc , ISSN 1074-3227

Schwabe, G. Szundy, de S. Moura, and F. Lima (2004). Design and implementation of semantic web applications. In *WWW 2004 Workshop on Application Design, Development and Implementation Issues in the Semantic Web*

Shneiderman, B. (1987). User interface design for the Hyperties electronic encyclopedia. In *Proceedings of the ACM Hypertext'87 Conference*, pages 189-194, ISBN 0-89791-340-x

Shneiderman, B. and Kearsley G., (1989). Hypertext Hands-On!: An Introduction to a New Way of Organizing and Accessing Information, Addison Wesley, ISBN 0-201-15171-5

Stefansky, W. (1972). Rejecting Outliers in Factorial Designs, In *Technometrics Vol 14*, pages 469-479

Stotts, P.D. and Furuta, R. (1989). Petri-Net-Based Hypertext: Document Structure with Browsing Semantics. In *ACM Transactions on Information Systems* 7(1), pages 3-29, ACM Press, ISSN 1046-8188

Van Dam, A. (1988): Hypertext '87: Keynote Address. In *Commun. ACM 31(7)*, pages 887-895

Vdovjak, R., Frasincar, F., Houben, G.-J. and Barna, P. (2003). Engineering semantic web information systems in hera. In *Journal of Web Engineering,* 2(12), pages 3-26, Rinton Press, ISSN 1540-9589

Van Deursen, A., Klint, P. and Visser, J. (2000). Domain-Specific Languages: An Annotated Bibliography. In *SIGPLAN Notices 35(6),* pages 26-36, ACM Press, ISBN 1-58113-255-7

Walker, J.H. (1987). Document Examiner: Delivery interface for hypertext documents. In *Proceedings of the ACM Hypertext'87 Conference*, pages 307-323, ISBN 0-89791-340-x

Warmer J. and Kleppe A. (1998). *The Object Constraint Language Precise Modeling with UML.* Addison-Wesley, ISBN 0201379406

Wintraecken, J.J.V.R. (1990). *The NIAM Information Analysis Method: Theory and Practice*. Kluwer Academic Publishers, Dordrecht, The Netherlands, ISBN 079230263X

Yankelovich, N., Smith, K.E., Garrett, N. and Meyrowitz, N. (1988). The concept and the construction of a seamless information environment. In *IEEE Computer* 21, 1, pages 81-96

Yesilada, Y., Harper, S., Goble, G., and Stevens, R. (2004). Screen Readers Cannot See (Ontology Based Semantic Annotation for Visually Impaired Web Travellers). In *Web Engineering - 4th International Conference, ICWE 2004*, pages 445-458, Springer, ISBN 3-540-22511-0