A reference architecture for adaptive hypermedia applications

A Reference Architecture for Adaptive Hypermedia Applications

Hongjing Wu

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Wu, Hongjing

A reference architecture for adaptive hypermedia applications / by Hongjing Wu.
Eindhoven: Technische Universiteit Eindhoven, 2002. Proefschrift.

ISBN 90-386-0572-2
NUR 983

Keywords: hypertext / hypermedia / databases / adaptive hypermedia
C.R. Subject Classification (1998): H.5.4, H.5.2, H.5.1, H.2.4, H.3.4

SIKS Dissertation Series No. 2002-13
The research reported in this dissertation has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Cover design: Jan-Willem Luiten
Printed by University Press Facilities, Eindhoven, the Netherlands.

A Reference Architecture for Adaptive Hypermedia Applications

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus, prof.dr. R.A. van Santen,
voor een commissie aangewezen door het College voor Promoties
in het openbaar te verdedigen
op vrijdag 8 november 2002 om 16.00 uur

door

Hongjing Wu

Geboren te Shanghai, China

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. P.M.E. De Bra
en
prof.dr. L. Hardman

Copromotor:
dr.ir. G.J.P.M. Houben

Preface

My research interest in adaptive hypermedia originates from a project on distance learning, in 1998, when I visited the Computer Science Department of the University of Amsterdam. There I met my supervisor, Prof. Dr. Paul De Bra, and found our common interest. He quickly introduced me into the field of adaptive hypermedia by demonstrating his system AHA!.

Devoting myself to this research for four years, I have learned much of the depth and breadth of this research area and gained international research experience. During these years of work with my colleagues, I very much enjoyed the many academic discussions as well as discussions about the many interesting things in our world.

I finished this dissertation with the assistance of various people. I would like to thank all of them here. First of all, I thank my father Zhaoji Wu and my mother Fengying Li, they gave me enormous support for my study in my life. They helped me very much by taking care of my daughter Xiaomeng Liu when I was in my first year of study here. It allowed me more time to deal with the challenges in my life in a foreign country and the new research at hand.

A debt of gratitude is owed to my supervisor Prof. Dr. Paul De Bra. He has guided and helped me with his broad knowledge and great kindness during my research. He has an open mind and a sharp eye for the value of ideas; this was especially important when faced with the challenge of choosing between many possible research directions. I would like to thank all the other committee members, specifically Prof. Dr. Lynda Hardman. She gave me a different view on my dissertation, making it more readable and balanced. I would like to thank my co-promoter, Dr. Ir. Geert-Jan Houben. He contributed much of his time and effort when I was starting my research, and helped me to plan my research agenda well.

Within our group, many colleagues kindly helped me in various ways. I would specifically extend my thanks to Dr. Ad Aerts for showing his enthusiasm towards my research, and helping me clear out some clouds in the early stages of discussions about the behavior of adaptive hypermedia systems, which is the second main part of my dissertation. His voluntary review of my dissertation gave me many suggestions to make my dissertation more balanced. I would also like to thank one of my previous colleagues, Dr. Jan Hidders, for lending me his deep knowledge and understanding of databases, and for his kind help in determining what rule language would be suitable for describing adaptation in adaptive hypermedia systems.

Finally I would like to give special thanks to my husband M.Sc. Erik de Kort. We had many inspiring discussions and he gave me his amazing dimensions of kind help during my Ph.D. research.

Hongjing Wu

Eindhoven, September 2002

Contents

Chapter 1

Introduction

This chapter is focused on presenting the general research agenda. Section 1.1 describes the motivation of my research. Section 1.2 defines the research questions and approaches. Section 1.3 presents an outline of this dissertation.

1.1 Motivation

When an author (or an organization) creates an information source it is intended to serve a large audience. The authoring process is thus always "one to many". Users on the other hand wish to receive exactly the information they need, presented in the way they want it. They wish to experience an information source as if it were created "one to one". In order to create this experience while keeping a "one to many" authoring process we introduce *personalization*. Different users have different interests. A personalized information source will present different information to different users. Users also have different backgrounds and knowledge, also resulting in the need for personalized information; they have different preferences for the presentation, including a preference for different media, for the length or difficulty level of a presentation, for presentation order, etc. The main goal of this dissertation is to provide a conceptual framework for systems that provide personalized information presentations, generated from a single information source.

Traditional text documents can only provide a linear organization of information. It is impossible to change the presentation for different users at all. Once the text document has been designed, it is received by all users in exactly the same form. Hypertext was proposed by Vannevar Bush in 1945. Bush [Bush, 1945] proposed a (then) revolutionary way to access information compared with traditional text documents. The title of this article "As We May Think" suggests that users can organize information in the same way as they think. Hypertext is an information structure in which documents are connected by hyperlinks. Users can jump from one document to another document by following hyperlinks between the documents. Hypertext provides users navigation freedom through an activity called browsing. Users choose their own reading order among pages or documents. Hypertext provides a non-linear organization of information. Nowadays we mostly use the term

hypermedia because we consider that the information may be presented in forms other than just text. Throughout this dissertation we will use the two terms interchangeably.

Hypertext does not really provide personalized information. When a document needs to be adjusted to a user's knowledge level or to his or her background the system may offer links to different versions of the information, but it cannot automatically guide the user to the version that is most suited for him or her.

Another problem with hypertext is that the navigational freedom introduces *navigation* and *orientation* problems, which traditional text does not have. In a book it is easy to know where you are and what you read before thanks to the linear structure. In hypertext it is difficult to remember what you read before because that is not an easily identifiable part of the whole structure. Navigation and orientation thus requires considerable cognitive overhead. When the hyperspace becomes huge, users easily get lost in the hyperspace in the sense that they do not know where they are and where to go to get their desired information. Users need personalized orientation and navigation support in hypertext. The system "remembers" where the user is and can reorient him or her upon request.

Adaptive hypermedia keeps the navigation freedom of hypertext and provides personalization through *adaptive content* and *adaptive navigation support*. (In this dissertation we do not distinguish between "orientation" and "navigation" support.) Adaptive hypermedia systems (AHS) have been under development for over ten years, in many different application areas. Still, until now there was no standard reference model or architecture for AHS. Many AHS offer very similar features, and the same ideas are repeatedly being invented. It is difficult for people to understand and compare different AHS because they are all described in very different ways. For a designer of an adaptive application it is difficult to find out which AHS could provide the adaptation required for the given application. The main goals of this dissertation are to provide a reference model or architecture for AHS, to describe the functionality of an AHS at an abstract level, and to illustrate how an AHS works, using the developed reference model. We validate our reference architecture, by describing the functionality of two well-known existing AHS using the model.

1.2 Research Questions and Approaches

This dissertation provides answers to five research questions:

Question 1: How do hypertext systems (and applications) attempt to personalize the information?
We will look back at the history of hypertext to investigate how hypertext systems solve the personalization problem. We want to find out what limitations hypertext systems have in providing personalization. We need to know what others have tried in order to overcome the limitations in hypertext systems.

When we know what the problems of hypertext systems are, we can ask ourselves the second question:

Question 2: How can adaptive hypermedia systems solve the problems of (non-adaptive) hypermedia systems?
We will look at the history of adaptive hypermedia systems to learn what is the difference between hypertext systems and adaptive hypermedia systems, and how AHS try to solve the problems of hypertext systems.

We want to make AHS easy to understand and provide a way to compare the different AHS. We therefore ask the third question:

Question 3: Can we describe the functionality of AHS at an abstract level? Can we describe adaptive hypermedia systems as extensions to hypermedia systems in general?
We will investigate AHS to find out the main parts included in these systems, and how these parts work together to provide adaptation. Then we can build an abstract conceptual model for AHS. In order to make this a "reference model" we first look at reference models for hypertext. Because AHS are hypertext systems, we do not need to repeat the same work that others have already done for hypertext systems. We can then concentrate on the issues related to personalization or adaptation.

A general, abstract model of AHS does not yet describe how adaptation is exactly performed in AHS. It does not describe the behavioral semantics of AHS, because the behavioral semantics of AHS are system or implementation dependent. To be able to describe how AHS exactly work we ask ourselves the fourth question:

Question 4: Can we define behavioral semantics for AHS and analyze how AHS work exactly?
We need to analyze the way in which AHS execute transactions in order to decide what our main problems are. We expect (and will find this to be true in Chapter 4) that systems that provide easy yet powerful ways for authors to describe the desired adaptation will have certain "decidability" problems. We need to provide ways to solve these problems in such a way that authors can avoid them with the help of system-tools and so that end-users are never confronted with these problems.

Finally we want to validate that our model is indeed a reference model. Therefore we ask the fifth question:

Question 5: Can we easily describe the adaptation functionality of some well known existing AHS in our model?
We select two representative AHS, InterBook [Brusilovsky et al., 1996b] and AHA! [De Bra et al., 2000] to validate our reference model.

1.3 Outline of the Dissertation

Chapter 2 answers research Question 1 and Question 2. It recalls the history of hypertext systems to show that navigation support was considered necessary already in early hypertext systems to lower the cognitive overhead when users are browsing in a huge hyperspace. The chapter also introduces the Dexter reference model for hypertext systems. The Dexter model contributed to hypertext research by its three layer structure and by

concentrating on the Storage Layer consisting of information nodes and a link structure. Adaptive hypermedia systems (AHS) aim to improve the hypertext application by making it personalized. They provide adaptive presentation and adaptive navigation support. We give an overview of adaptive hypermedia systems by summarizing the old review and the new review by Brusilovsky [Brusilovsky, 1996, 2001].

Chapter 3 answers research Question 3. It proposes a Dexter-based reference model for Adaptive Hypermedia Applications: AHAM. AHAM describes adaptation of AHS at an abstract level. It advocates a clear separation between a domain model (DM), user model (UM) and adaptation model (AM) in an AHS. It is based on methods and techniques commonly used in AHS, and intends to provide a reference model for AHS to facilitate system designers to develop new AHS, authors to write their adaptive hypermedia (AH) applications and users to use AHS. This chapter summarizes the advanced adaptation features of AHAM.

Chapter 4 answers research Question 4. It describes design issues for a general-purpose adaptation engine (AE). All adaptive hypermedia research projects focus on providing adaptation. Most of them have very simple rules, so termination, confluence or other behavioral aspects of AHS are very often neglected. Termination means that the system provides an adaptation result in a finite number of steps, and confluence means the system provides deterministic adaptation results. Termination and confluence have been well studied in active databases. Based on the results from active databases, this chapter focuses on a rule language, called AHAM-CA, and describes its termination and confluence properties.

Chapter 5 and Chapter 6 answer research Question 5. To validate our reference architecture for adaptive hypermedia applications, Chapter 5 uses AHAM-CA to describe a well known adaptive hypermedia system: InterBook, designed by Brusilovsky et al [Brusilovsky et al., 1996b].

Chapter 6 uses AHAM-CA to describe another existing well known adaptive hypermedia system: AHA!, designed by De Bra et al [De Bra et al., 2000], as another validation of our reference architecture for adaptive hypermedia applications.

Finally, in Chapter 7, we give a summary of the main results and indicate some directions for future research.

Chapter 2

From Hypertext to Adaptive Hypermedia

This chapter focuses on presenting the background of the research described in this dissertation. Section 2.1 gives an overview of the starting background of the research by introducing the initial area of interest: hypertext. Section 2.2 gives an overview of models used for hypertext, with an elaborate description of the well-known Dexter model for hypertext. Section 2.3 describes the state of the art in adaptive hypertext systems. It summarizes the main parts of adaptive hypermedia applications. Section 2.4 summarizes the research background and discusses the answers to research questions 1 and 2 from Chapter 1

2.1 Hypertext

In this section we first give a definition of *hypertext*, and then show some ideas on how to solve navigation problems through a review of the history of (representative) hypertext systems.

2.1.1 Definition of hypertext

Traditional text documents provide a conceptually linear organization of information. For a growing number of applications, this is not sufficient. It is hard to determine the most appropriate reading order for some conceptual information that does not have an intrinsic sequential order, or for which there are several possible reading orders.

Hypertext is non-sequential by definition. There is no single order that determines the sequence in which the text is to be read. It provides a means for readers to actively explore rather than passively absorb a body of information. Hypertext consists of interlinked pieces of text (or other types of information). Each unit of information is called a node. Whatever the grain size of these nodes, each of them may have pointers to other units, and these pointers are called links.

Figure 2.1 shows that the entire hypertext structure forms a network of nodes and links. Readers move through this network in an activity called *browsing* or *navigating*, rather than just "reading", to emphasize that users must actively determine the order in which they read the nodes.

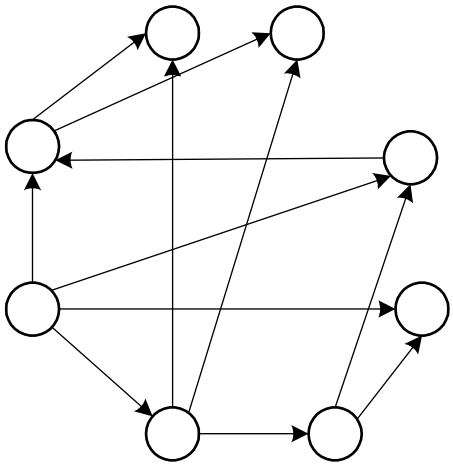

Figure 2.1: An example of hypertext structure

User interfaces of hypertext systems give users (limited) freedom to navigate through the hyperspace. (Hyperspace is a term used for the structure formed by the nodes and links.) Users only need to e. g. "click" on the links to go anywhere they want, and in the meantime users get a hyperdocument instead of a plain text document. Hypertext systems provide users a little or a large amount of navigational freedom, depending on how "rich" the link structure is.

Hypertext systems, however, generate navigation and orientation problems especially when the size of the hyperspace is very large. Users face the "lost in hyperspace" risk when browsing without a navigation guide based on information domain knowledge. Secondly, hypertext systems do not consider individuals. Users with different goals get the same information, and the same user continues to get the same information when his/her goal changes. Hypertext systems have inherent navigation and orientation problems when they offer users freedom in navigation. To overcome these problems, hypertext systems often provide navigation support tools to aid users. These navigation tools are so important that Frank Halasz [Halasz, 1988] from Xerox PARC has put forward the view that a true hypertext system should include an explicit representation of the network structure in its user interface. But in many systems that network is only present inside the computer. At any given time the user sees only the "current" node and links leading out of that node; it is up to the user's imagination to picture how the entire network is structured.

Very few hypertext systems provide a graphical representation of the entire hypertext on a computer screen, because a hypertext typically contains a large number of nodes. Halasz wanted to give the user a dynamic overview of the structure of this network in

NoteCards [Halasz et al., 1987], which displayed the structure in detail only for the local neighborhood surrounding the user's current location. (When a complete overview of a hypertext structure is given in which the local neighborhood is shown in detail and the rest is graphically summarized in some way then this is called a "fish-eye view" [Furnas, 1986].)

Many definitions of hypertext exist. An interesting definition is based on the "look and feel" of a hypertext user interface. It should make users *feel* that they can move freely through the information, according to their own needs. A hypertext system should also generate only a low overhead with respect to using the computer's resources. This means short response times so that the text is on the screen as soon as the user asks for it. Low overhead also contributes to reducing the cognitive load on the user when navigating: users do not have to spend their time wondering what the computer does or how to get it to do what they want.

Graphical overviews and fast response times are not enough to guarantee low cognitive overhead. We give an (informal) definition of hypertext that includes a higher-level structure than just nodes and links, to enable the system to base its features for lowering the cognitive overhead on more than just the navigational structure. We consider that the information structure of a hypertext consists of two types of elements:

- a set of *concepts*, each representing a unit of information. A concept can be a page or a composite concept that consists of other composite concepts or pages.
- a set of *concept relationships*, each connecting a series of concepts in some way. Physical links (hyperlinks) between pages are a type of concept relationship. Other types of relationships can be used to describe any kind of relationship among concepts. The meaning of each type of concept relationships depends on authors and system designers.

This informal definition is compatible with the formal definition used in the Dexter model [Halasz and Schwartz, 1990] which is used extensively in this dissertation. It not only covers the network of nodes and links as the Dexter model does, but also covers the semantic network among concepts which will play an important role in providing content adaptation and adaptive navigation support in adaptive hypermedia systems.

Many hypertext systems tried different ways to provide navigation support. Those navigation support tools were limited because they did not use semantic relationships between nodes, only navigational links. In fact there exist certain semantic relationships between the nodes in many hypertext applications, but they have been simply dropped from the definition of the system by only using links to represent a possible way leading out from the nodes. Most hypertext applications either ignore the differences between types of links or relationships, or simply assume a simple one-to-one mapping: semantic links in the document are directly mapped to navigational hyperlinks in the presentation. Some systems actually show different link types in different ways, but few systems allow the definition of semantic relationships that are not tied to hyperlinks. Hypertext applications may only make sense to users if they at least provide some navigation support, using the

semantics of the links between nodes, so that users can easily understand their navigation environment. In that respect World Wide Web, at least in its initial form, uses a very simple model. The only navigation support is that links to previously visited pages are purple and links to still unread pages are blue. More recent Web standards have a richer link model with link classes and at least the potential to provide better navigation support.

We believe that the semantics of different connections, called concept relationships, are the key to providing navigation support in hypertext applications. From a user's point of view, the concept relationships give cognitive information in navigation support. From the system's point of view, the concept relationships can play an important role in generating lower cognitive overhead by using them to automatically provide adaptation, adaptive content (sometimes also called adaptive presentation) and adaptive navigation support. We will see in the rest of this dissertation that concept relationships are one of the bases for adaptation, making hypertext applications personalized.

2.1.2 History of hypertext

We look back to the history of hypertext and show different hypertext systems each using their own way to deal with navigation and orientation problems. From a historical perspective hypertext research is strongly influenced by the three pioneers of the field: Bush, Engelbart and Nelson. They described the ambitious goals of the field, Bush in 1945 and the others in the early sixties. But hypertext systems have been actually developed and commercially used since the late sixties. The Hypertext Editing System (designed by Ted Nelson and Andy van Dam), a predecessor of Brown University's *Intermedia* system, was used in the late sixties to produce the documentation for the Apollo missions. In the eighties, Intermedia was effectively used in an educational environment to teach a course on cell biology and one on English literature. Engelbart's Augment/NLS was marketed as a commercial system by McDonnell Douglas. Carnegie-Mellon's ZOG system was used as an information management system on a nuclear-powered aircraft carrier and later developed into a commercial system called KMS (Knowledge Management System [Akscyn et al., 1988]). The year 1987 is the turning point in the history of hypertext in the sense that several new hypertext systems were introduced, and that the first ACM Hypertext conference was organized. Following is a list of representative hypertext systems from before 1987. (The information below is mostly gathered from [Nielsen, 1990] and from the on-line course 2L690 at TU/e.)

- 1945 Vannevar Bush proposed Memex in his article "As We May Think" [Bush, 1945]. Memex was designed as a mechanical hypertext device that used microfilm. It was never built.

- 1965 Ted Nelson introduced the Xanadu distributed system concept and coined the term hypertext [Nelson, 1965]. (It took until 1999 to actually build Xanadu.)

- 1967 Andries van Dam developed the Hypertext Editing System at Brown University [van Dam, 1987], followed by the introduction of FRESS in 1968.

- 1968 Doug Engelbart gave a demo of NLS, the oN Line System, as an experimental tool to store specifications, plans, designs, programs, documentation, reports, etc. [Engelbart, 1963]

- 1975 A team at CMU, headed by Robertson, developed the ZOG system, which later became KMS [Akscyn et al., 1988]. ZOG was designed to be used on standard character terminals. Each segment of a ZOG database was called a frame, and consisted of a title, a description, a line with standard ZOG commands, and a set of menu items, called selections, leading to other frames. The structure was always hierarchical, though some cross-reference links could be included. The purely text-based interface with relatively small frames easily led to user-disorientation. Donald McCracken and Robert Akscyn, two major developers of the ZOG system at CMU, started the company Knowledge Systems in 1981, and produced KMS, the Knowledge Management System. KMS used the same frame-based interface as ZOG, with the same danger for disorientation. The developers claimed that the possibility for user disorientation should be greatly reduced by the fact that you could move very quickly among frames and thus become reoriented with very little effort. We know of no empirical evidence for (or against) this claim. However, given our own experience with Web browsing we believe that this claim is false; we believe that navigation and orientation support is needed to avoid disorientation. If ZOG and KMS caused little disorientation it was probably because the structures were normally hierarchical.

- 1978 A team at MIT, headed by Andrew Lippman, developed the Aspen Movie Map, the first true example of a multimedia application including videodisk [Lippman, 1980]. The Aspen Movie Map was a surrogate travel application that allowed the user to take a simulated "drive" through the city of Aspen on a computer screen. It used two monitors for its interface. One provided users with an immersing view of the city and made them feel as if they entered into the environment. The other monitor was placed horizontally and provided users with a street map. The user could point to a spot on the map and jump directly to it instead of having to navigate through the streets. In this way the Aspen Movie Map provided both navigation and orientation support.

- 1985 Janet Walker [Walker, 1987] developed the Symbolics Document Examiner, claimed to be the first hypertext system used by "real" customers. The user-interface was kept as simple as possible. A traditional hierarchical structure, using chapters and sections, was chosen so users would recognize the book metaphor, and not have to learn the network based navigation which is typical for hypertext. High resolution displays were used in order to make reading from the screen as comfortable as reading from paper. The user could add bookmarks, to make returning to specific items easier. A survey among 24 users showed that almost all of them preferred the hypertext version over the printed manual. Still, these users were all engineers, who might have been more motivated to use high technology solutions than ordinary users would be.

- 1985 Several other hypertext systems were announced, including NoteCards [Halasz et al., 1987] from Xerox, and Intermedia [Meyrowitz, 1986] from Brown University. In NoteCards each node was a single notecard, which could contain an arbitrary amount of information. As in Guide and Intermedia (see later), scrolling was used to display different parts of the node. When following a link the destination node could be displayed in a new window, possibly overlapping the old window. Users could open as many windows as they wanted, thus generating a messy desktop. Links were typed connections between cards. The type was a label chosen by the creator of the link, and could indicate the kind of relation between the source and the destination node. The browser card showed a graphical overview of the link structure of the hyperdocument. The most advanced hypertext system developed at Brown University is the Intermedia system. Intermedia offered two kinds of overviews, to help the user find his or her way through a large document:
  - Overview nodes: these nodes displayed a fixed part of the document's structure. They can best be compared to a menu-like node, showing which major components were reachable from this node. A good example of an overview node would be the table of contents of this dissertation. Overview nodes were constructed manually using a special drawing package. Any layout was possible, but by convention the name of the central topic was put in the middle, surrounded by the related parts of the document.
  - The web view: this was an automatically constructed graphical overview of the link-structure of the entire hyperdocument.
- 1986 OWL introduced Guide for the Macintosh, the first widely available hypertext system, based on the Unix Guide system [Brown, 1987], developed by Peter Brown at the University of Kent. Most hypertext systems use pagination when following a link, meaning that the currently displayed node is replaced by the destination of a link. In Guide the main link-mechanism was based on replacement, meaning that when following a link the current node would break open, making room for the destination node. The anchor of the link was replaced by the contents of the destination node. One could close the destination node, which means that it was once again replaced by the text of the anchor. Replacement means that the structure of the hyperdocument must be strictly hierarchical. Each time the user "followed" a link the source node of the link remained a visible context in which the destination node was shown. This should have been very effective as orientation support, but it limited the possible presentations. In addition to replacement anchors, Guide supported pop-ups for small annotations, and so called jumps, which behaved like the follow-link operation in most hypertext systems. (They cause the current node to be replaced completely by the destination node.) The jumps provided a way to create non-hierarchical links, albeit without the orientation support offered by the replacement link mechanism.
- 1987 Apple started delivering HyperCard [Wesley, 1989] for free with every Macintosh. HyperCard was, however, not originally designed to be a hypertext system,

according to its designer, Bill Atkinson. It was intended as a tool for developing prototypes of user-interfaces. Typically all cards in a HyperCard stack have a button to go to the next card and to the previous card. This lets you page through the stack. Stacks are therefore mostly useful to provide a paging mechanism for nodes that are too large to fit onto a single card. However, in general there need not be a linear order in which to read the cards. The stack is atypical for hypertext applications, because it suggests a linear structure, while hypertext promotes the use of much richer structures. Creating hypertext using HyperCard is more like programming than like authoring. Therefore, despite the fact that HyperCard was (and is) free, many hypertext documents have been developed using the more recent commercial hypertext system Storyspace which is available for both the Macintosh and the IBM-PC with Windows.

In 1987 the the first ACM Conference on Hypertext was held. Since around that time the new developments are too numerous to name all of them. World Wide Web, invented by Tim Berners Lee at CERN in 1989, demonstrated at the ACM Hypertext Conference in 1991, and made popular by Marc Andreessen and Eric Bina from NCSA in 1992 by means of the Mosaic for X browser, has really pushed hypertext into the mass market. Unfortunately it has brought a very limited hypertext system to the mass market, with a very simple link structure. (Initially there were no typed links, and the only navigation support was that the blue links became purple when they led to a previously visited page.)

In the ACM Hypertext'91 conference proceedings Wright [Wright, 1991] discussed some issues in the evaluation of hypertext applications. She stated that, when evaluating hypertext, navigation support needed to be included. She addressed reducing cognitive overhead in navigation as being an important issue for hypertext to be useful. We give a brief overview of navigation support (aiming to lower cognitive overhead) in the hypertext systems published at ACM Hypertext Conferences since 1990.

- 1990: Bruza [Bruza, 1990] used hyperindices as a means for supporting effective search in hypertext. He showed how the hyperindex can be constructed using the structural properties of index expressions. These hyperindices are hypertexts themselves, but consisting of concepts, not pages. From concepts the user could navigate to associated pages and back through operations called *beam down* and *beam up*.
  Wilson [Wilson, 1990] showed the need to automate the conversion of traditional legal documents into an integrated hypertext database. He discussed the types of links the system uses to cater for: 1. linear and hierarchical structure; 2. directed graphs; 3. annotational or associative links; 4. index or concept links. He illustrated that these links can create different virtual structures for the document collection to give flexibility of access and navigation. DeYoung [DeYoung, 1990] showed an idea to identify the different underlying structure of the ways specific sets of data are related, and then to use the structure to provide navigation support.

- 1991: Lai and Manber [Lai and Manber, 1991] showed an additional tool to navigation: flying. The variability of the links and structure that they provided enabled

flexible flipping in many different orders controlled by the reader. The key to any flipping was speed. The goal was not to digest the contents of the pages (like in ZOG and KMS), but rather to gain some insight to features such as organization, size, depth, level, detail and so on. Flying could also be used to move fast from one place to another in the hypertext following a certain order for traversal.

- 1992: Quint and Vatton [Quint and Vatton, 1992] provided two-level navigation in the Grif system, a structured document editor based on the generic structure concept. It combined elements of hypertext and structured documents. The notion of logical structure encompassed both hierarchical structures (as is usual in structured documents) and non-hierarchical links (as is usual in hypertext).

- 1993: Noik [Noik, 1993] addressed the importance to provide an overview for global context while users go to local details in Fisheye views. A Fisheye view displays information at several levels of abstraction simultaneously.
  Chang [Chang, 1993] stated disorientation in hypertext could be reduced by limiting the number of links, and by attaching attributes to links that could be used to group, sort and filter them. Links, however, still could be overwhelming if they did not reflect the experiences and biases of the user even if links could be organized and filtered. Chang proposed a user-centered approach for automatic link generation by exploiting existing hierarchical hyperdocument structures, or the pattern of user-created links in HeiNet.

- 1994: Simon and Erdmann [Simon and Erdmann, 1994] provided responsive manuals to the current situation in SIROG. A responsive manual consists of a "standard" hypertext-based operational manual and a task description. It monitored the changing situation and based on this was able to point to relevant information.
  Mukherjea et al. [Mukherjea et al., 1994] talked about clustering related nodes of an overview diagram to reduce its complexity and size. This was because although overview diagrams are useful for helping the user to navigate in a hypertext system, for any real-world system these become too complicated and large to be really useful. Since the nodes could be related to each other in different ways, depending on the situation different clustered views would be useful. They provided navigational views based on these abstract views. They proposed a 3-dimensional approach for visualizing these abstracted views. But they did not have a user model to store the user needs or goals; their system interactively asked for clustering conditions to select related nodes in the overview diagram.

- 1996: Weiss [Weiss et al., 1996] described in HyPursuit a hierarchical network search engine that clustered hypertext documents to structure a given information space for browsing and search activities. HyPursuit's abstract functions summarized cluster contents to support scalable query processing. Its content-link clustering algorithm was based on the semantic information embedded in hyperlink structures and document contents.

2.1.3 Hypermedia

Hypermedia is a combination of "hypertext" and "multimedia". In [van Ossenbruggen, 2001] Jacco van Ossenbruggen gives a clear description of three types of hypermedia systems according to different aspects in different combinations.

In the first group, hypermedia system means "multiple media hypertext". These systems are developed as hypertext systems, they are based on the node/link model of hypertext, and are extended to handle media data types. The addition of multimedia in these systems does not change the underlying data and process models. Researchers in this area often use the terms "hypertext" and "hypermedia" as being interchangeable. For example, for the Dexter model this combination can be easily explained: the addition of new media types in Dexter's *Within-Component Layer* does not affect the basic hypertext data structures modeled by the *Storage Layer*, nor does it affect the navigation-based interaction process modeled by the *Runtime Layer*. This dissertation focuses on providing adaptation in hypermedia systems that fit in this group.

In the second group, hypermedia systems provide "interlinked multimedia". They have their roots in multimedia rather than hypertext. They are multimedia systems, and are extended to provide navigation-based interaction in addition to the more traditional—VCR-style—interaction mechanism. These systems support the specification of synchronization constraints on their constituent media items, and have built-in support for defining multimedia layout. Linking facilities in these systems are not as sophisticated as those provided by their hypertext counterparts, because the hypertext node/link model is subordinate to the spatio-temporal composition model.

In the third group, hypermedia systems are a full integration of hypertext and multimedia. These system are described as "non-linear multimedia", but not yet widely supported.

From now on we use the term hypermedia systems to represent the combination that fits in the first group.

2.2 Reference Models for Hypertext Systems

A hypertext system is a complex piece of software, consisting of several parts which serve a very different purpose. Campbell and Goodman proposed a division of a hypertext application [Campbell and Goodman, 1988] in the following way:

- the Presentation Level or user interface
- the Hypertext Abstract Machine, serving nodes and links
- the Database level, providing efficient storage and network access

Such a separation of concerns forms the basis of all *reference models* for hypertext.

A *reference model* for hypertext systems describes the possible conceptual elements and the functionality of hypertext systems, in an abstract (implementation independent) way. There are several reference models for hypertext systems. We name five models that are sometimes refered to in literature:

- The HAM or Hypertext Abstract Machine, as described by Campbell and Goodman [Campbell and Goodman, 1988].
- The Trellis model, a reference model by Furuta and Stotts [Furuta and Stotts, 1990].
- The Dexter model, a reference model by Halasz and Schwartz [Halasz and Schwartz, 1990], written in the specification language Z [Spivey, 1989].
- A Formal Model by B. Lange, a reference model written in the specification language VDM [Lange, 1990].
- The Tower Model, a more general object-oriented model by De Bra, Houben and Kornatzky [De Bra et al., 1992].

In this dissertation we are going to base our model on the Dexter model, because it was formalized [Spivey, 1989; van Ossenbruggen, 2001] and even implemented [Grønbæk and Trigg, 1994]. However, to give an impression of the similarity and/or difference between our chosen model and other models we first briefly introduce the HAM model, which was the first reference model and inspired the other models.

2.2.1 The HAM model

In 1987, Brad Campbell and Joseph M. Goodman published HAM [Campbell and Goodman, 1988], at the first Conference on Hypertext. It was a milestone in the development of hypertext as a research field, because it was the first attempt to define a reference model, an abstract model in which actual hypertext systems could be expressed. Instead of presenting HAM model as a reference model, Campbell and Goodman presented it as an abstract machine.

"The HAM is a transaction-based server for a hypertext storage system. The server is designed to handle multiple users in a networked environment. The storage system consists of a collection of contexts, nodes, links, and attributes that make up a hypertext graph. The versatility of the HAM can be illustrated by showing how Guide Buttons, Intermedia Webs, and NoteCard FileBoxes can be implemented using its storage model."

Campbell was one of the developers of a HAM system at Tektronix. The work was inspired by the Neptune system of Delisle and Schwartz [Delisle and Schwartz, 1986]. Although the HAM system was used by several groups in and outside Tektronix it never became a commercial product. Because the HAM was a very powerful system, describing its properties provides a way to compare features of different hypertext systems.

The definition of the HAM consists of a description of the HAM objects and of the operations that can be applied to them. Since 1987 hypertext systems have been developed that offer richer structures and features than those envisioned by the designers of the HAM. This has led to newer reference models, presented at a workshop of the National Institute of Standards and Technology in 1990, and to the definition of the Tower Model in 1992.

Like most reference models, the HAM does not describe the hypertext system completely. The HAM sits in between the file system and the user interface. Campbell and Goodman envisioned the graphical representation given in Figure 2.2.

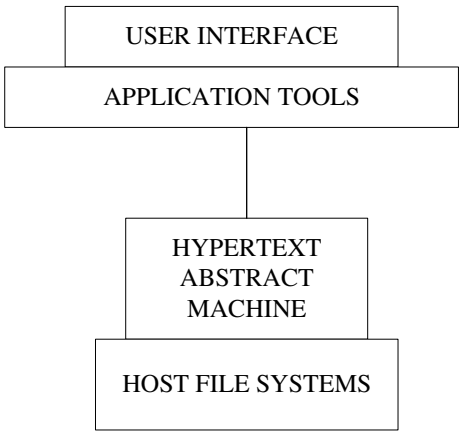

Figure 2.2: HAM reference model

Figure 2.2 clearly indicates that the HAM is a lower level machine, tied closely to the storage (file) system, while having a looser connection to the applications and user interfaces. It is also clear that the HAM is only part of this architecture and not the whole system.

2.2.2 The Dexter model for hypertext

Janet Walker and John Leggett organized two workshops on hypertext. The first one was held in October 1988, at the Dexter Inn in New Hampshire. The discussions at these workshops were mostly about the major existing hypertext systems like Augment, Concordia/Document Examiner, Hypercard, Hyperties, Intermedia, KMS, Neptune/HAM, NoteCards and others. Many leading hypertext developers attended these workshops, including Rob Akscyn, Doug Engelbart, Steve Feiner, Frank Halasz, John Leggett, Don McCracken, Norm Meyrowitz, Tim Oren, Amy Pearl, Catherine Plaisant, Mayer Schwartz, Randy Trigg, Janet Walker and Bill Wieland. The result was a paper, written by Halasz and Schwartz, describing the Dexter reference model for hypertext [Halasz and Schwartz, 1990].

The Dexter hypertext reference model is an attempt to capture, both formally and informally, the important abstractions found in a wide range of existing and future hypertext systems. The goal of the model is to provide a principled basis for comparing systems as well as for developing interchange and interoperability standards. The model has been formally specified in the language Z [Spivey, 1989]. Later Jacco van Ossenbruggen [van Ossenbruggen, 2001] gave an Object-Z description of Dexter model which is more precise, more straightforward, and more compact.

The Dexter model is divided into three layers (see Figure 2.3): Runtime Layer, Storage Layer, and Within-Component Layer. The Within-Component Layer is introduced to isolate the other layers from all data and media-specific details. The elaboration of the Within-Component Layer is considered beyond the scope of the model. The two remaining layers can be said to represent two alternative views on what the important characteristics of a hypertext system are.

The first view focuses on the underlying data structure used to model hyperlinks, and is reflected on the middle layer of the model, the storage layer. This layer describes a hypertext as a networked data structure of nodes and links, the essence of hypertext. It uses "components" to cover both nodes and links.

The second view stresses the importance of the unique, associative user interface and behavior of a hypertext system. This view is reflected in the third layer of the model, the runtime layer. This layer describes mechanisms for supporting the user's interaction with the hypertext.

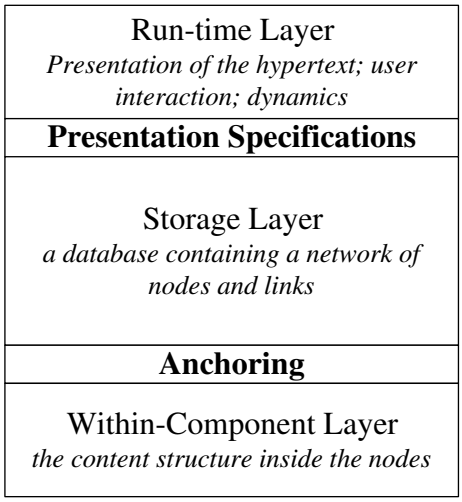

Figure 2.3: The Dexter reference model for hypermedia systems

The notion of *anchoring* is introduced to describe the main mechanism between the within-component layer and the storage layer. Anchoring is used to be able to address locations or items within components, without knowledge of their inner structure. This extra level of indirection allows a description of hyperlinks that is independent of the structure of the media items at the end points of the link.

The notion of *presentation specifications* is used to describe the interface mechanism between runtime layer and storage layer. It tells how a component is to be presented to the user. Since presentation specifications are typically application and/or media dependent, Dexter makes no attempt to model their internal structure. Only the aspects that are needed to bridge the gap between storage layer and runtime layer are described.

The emphasis of the Dexter reference model for hypertext systems is on the storage

layer, the anchoring, and the presentation specification. Below we give a definition of these parts of the model.

The Storage Layer focuses on the mechanism that enables the data-containing components to be interconnected by link components. Figure 2.4 depicts the three types of components of the Dexter model: atoms, links, and composites.

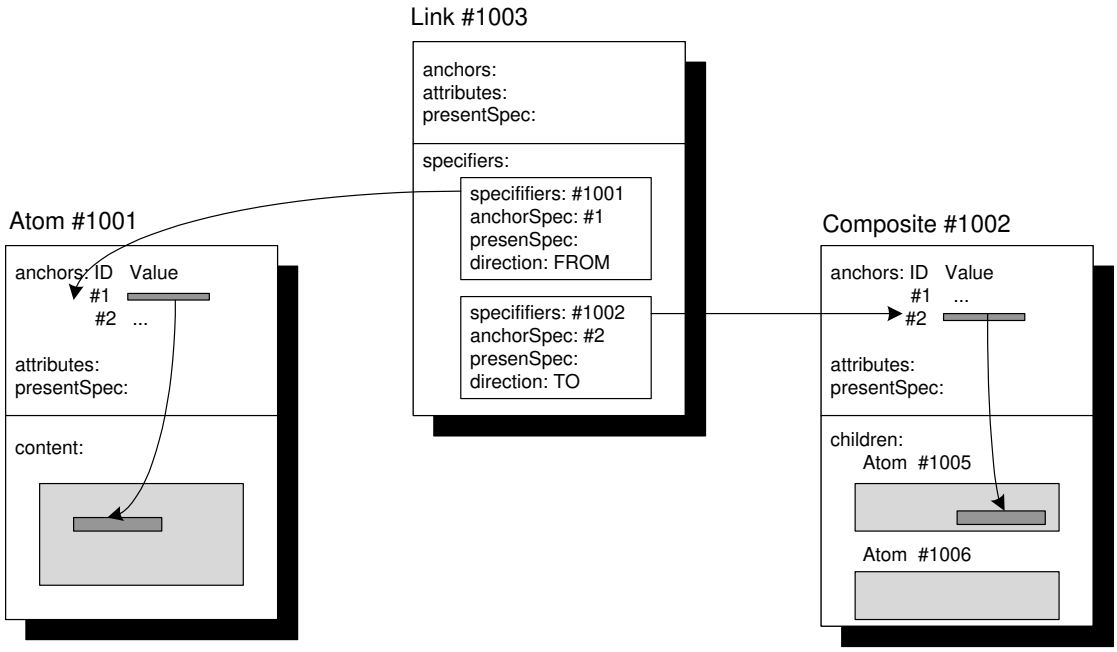

Figure 2.4: Dexter linking from an atomic to a composite component

Atomic components are primitive in the model. Their content is part of the within-component layer and thus not described in the model. Composite components are recursive structures made out of other components. The "containment" relationship between components is restricted to be a directed acyclic graph (DAG), which means a component may be a subcomponent of multiple components, but no component may contain itself either directly or indirectly. The *link* component models relations between components. Links contain a sequence of *endpoint specifiers*, each of which refers to (a part of) a component in the hypertext. A system can support unary, binary or arbitrary arity links.

The addressing of components involves a two-step process. A *resolver* function maps a given component specification to the unique identifier (Uid) of the component, and an *accessor* function maps a given Uid to the component itself. The explicit use of these two functions intends to address the issue of support for search and query facilities, and the issue of support for virtual structures, as advocated by Halasz [Halasz, 1988].

In the Dexter model a hypertext is modeled as a set of atomic, link and composite components with an associated accessor and resolver function. We define the concepts of the Dexter model more formally below.

Definition 2.1 *A* component *is a pair* $< uid, cinfo >$*, where uid is a globally unique identifier for the component, cinfo represents "component info" and consists of:*

- *a* set of attribute-value pairs
- *a* sequence of anchors
- *a* presentation specification

Attributes can be of any type and used for any purpose. They are not elaborated further. We also do not describe the details of presentation specifications. We distinguish two types of components: atomic and composite.

An atomic *component is a component for which the anchor values and possibly some attribute values belong to the within-component layer. (This means that we cannot describe their internal structure in the model.)*

A composite *component is a component where the anchor values are the unique identifiers of other components (that we refer to as subcomponents).*

Definition 2.2 *An* anchor *is a pair* $< aid, avalue >$*, where aid is a unique identifier for the anchor within the scope of its component and avalue is an arbitrary value that specifies some location, region, item or substructure of a component. Anchor values of atomic components belong to the within-component layer and are not elaborated in Dexter. Anchor values of composite components are uid-aid pairs, where the uid identifies a subcomponent of the composite and the aid identifies an anchor within that subcomponent.*

The notion of anchoring is regarded as one of the major contributions of the Dexter model. The anchor identifier *aid* is media-independent, all anchors need to be explicitly present at the level of the storage layer, links can only refer to anchors by means of the anchor identifier. The associated anchor value represents the role of the anchor in the within-component layer, and is typically media-specific. It is interpretable only by the end-user application, and not by the "hypertext engine". This indirection isolates the link from the lower level details needed to interpret the anchor value.

Definition 2.3 *A* specifier *is a 4-tuple* $< uid, aid, dir, pre >$ *where uid is the unique identifier of a component, aid is the identifier of an anchor, dir is a direction (which is FROM, TO, BIDIRECT or NONE), and pre is a presentation specification.*

Definition 2.4 *A link is a component* $< uid, ss, cinfo >$ *where uid is a unique identifier, ss is a sequence of specifiers, and cinfo is the component information. A component's information consists of:*

- *a set of attribute-value pairs*
- *a sequence of anchors*
- *a presentation specification*

Note that the endpoints of a link are contained in a sequence of specifiers. The sequence of anchors of a link allow links to become endpoints of other links themselves. In this dissertation we will not consider links that have other links as endpoints and thus we do not need the sequence of anchors as part of a link component.

More details about the Dexter reference model will be given (as needed) in the next chapter where we describe our reference model for adaptive hypermedia.

2.3 Adaptive Hypermedia Systems (AHS)

Adaptive hypermedia is a relatively new direction of research on the crossroad of hypermedia and user modeling. Adaptive hypermedia systems (AHS) maintain a user model to store a user's "features", and use these features to provide adaptive content and adaptive navigation support.

Adaptive hypermedia can be traced back to the early 1990s. Peter Brusilovsky gave an overview of adaptive hypermedia systems, methods and techniques in 1996 [Brusilovsky, 1996]. The year 1996 can be considered a turning point in adaptive hypermedia. Before this time, research in this area was performed by a few isolated teams. Most systems that were developed were not Web-based. After 1996 adaptive hypermedia has gone through a period of rapid growth and fast adoption of Web technology. Brusilovsky gave a new version of the overview of adaptive hypermedia systems in 2001 [Brusilovsky, 2001]. This section gives a review of existing adaptive hypermedia systems by summarizing the old review and the new review in [Brusilovsky, 1996, 2001]. It shows which different types of adaptive hypermedia systems were developed in the short history of this research field, which features of users are used in adaptation, and which methods and techniques are used to achieve the adaptation goals. We summarize the three main parts that most adaptive hypermedia applications have, and these parts are described in more detail in the description of the reference model in the next chapter. Finally we give two examples of different ways of authoring adaptive hypermedia applications for InterBook and AHA!. An important goal of our reference architecture for adaptive hypermedia applications is to represent adaptive applications in such a way that it becomes possible to separate the authoring from the system design. When new systems will be developed following our architecture they will support powerful adaptation strategies through a very simple authoring process.

2.3.1 AHS and adaptation

Hypermedia systems in general, and Web-based systems in particular, are becoming increasingly popular as tools for user-driven access to information. The goal of adaptive hypermedia research is to improve the usability of hypermedia applications by making them personalized. An AHS builds a model of the goals, preferences and knowledge of each individual user. It uses this model to adapt the application to the needs of the user. The user modeling necessary for the adaptation is based on observing the user's behavior. Most often that behavior will be browsing. An AHS thus works "in the background",

without asking the user for specific input on his or her goals, preferences or knowledge. In addition AHS also support the adaptation when users explicitly set their features in the user model (for instance through a form), but this is not the main concern here because it is relatively simple to realize.

Definition of adaptive hypermedia systems

To clarify the goal of this research, we recall the following definition from Brusilovsky [Brusilovsky, 1996]:

> 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. In other words, the system should satisfy three criteria: it should be a hypertext or hypermedia system; it should have a user model; it should be able to adapt the hypermedia using this model.

Identified dimensions for adaptation

To review adaptive hypermedia systems we follow the identified dimensions described in [Dieterich et al., 1993] which are typical for the analysis of adaptive systems in general. These dimensions are the basis for the classification of adaptive hypermedia methods and techniques.

- The first dimension considered is *where adaptive hypermedia systems can be useful*. The review identifies several application areas for AHS (Table 2.1) and for each area it points out the problems which can be partly solved by applying adaptive hypermedia techniques (Section 2.3.2).

- The second dimension is *what to adapt to*. It consists of two categories.
  User characteristics: what features of the user are used as a source of the adaptation, i. e. to what features of the user can the system adapt its behavior. Brusilovsky's review identifies several user features that are considered important by existing AHS and discusses the common ways to represent them.
  Environment: adaptation to the user's environment. This is a new kind of adaptation that was brought on by web-based systems. It considers adaptation to the equipment (computer, PDA, cellphone) and the network (from high-speed fixed connection down to low speed modem connection) (Section 2.3.3).

- The third dimension is *what can be adapted* by a particular technique. Which features of the system can be different for different users. Along this dimension the reviews [Brusilovsky, 1996, 2001] identify seven ways to adapt hypermedia. They can be divided into two essentially different groups - content adaptation and link adaptation. Brusilovsky calls these different ways to adapt hypermedia *technologies of adaptation* (Section 2.3.4).

- The fourth dimension is the *adaptation goals* achieved by different methods and techniques: why these methods and techniques are applied, and which problem of the users they can solve. The adaptation goals are dependent on the application goal in some range of application areas. The adaptation goals are considered in parallel with our review of relevant adaptation methods and the techniques that implement these methods (Section 2.3.4).

2.3.2 History and application areas of AHS

As we stated previously, AHS started around 1990. 1996 is a turning point in their development. Before 1996 all systems had been built to explore some new methods. Since 1996 the introduction of the Web has had a great impact on hypermedia in general. Many adaptive hypermedia systems have become mature, partly because the Web provided an easy development and application platform. Some systems provide complete frameworks including authoring tools. We will describe AHS in these two periods of time.

Before 1996: Brusilovsky gave an overview of adaptive hypermedia systems in 1996. He classified existing AHS into six groups according to their application areas: (Table 2.1). They are educational hypermedia systems, on-line information systems, on-line help systems, information retrieval hypermedia, institutional hypermedia, and systems for managing personalized views in an information space. The first two of these areas were most popular, the next two were reasonably populated being represented by 5-6 systems each and the last two were the least investigated being represented by 1-2 pioneer systems each. All hypermedia systems from before 1996 are similar in the sense that they offer a "browsing" environment. They keep user features in a user model that is constructed by following the user's browsing behavior, and use that model as the basis for adaptation.

The most popular area for adaptive hypermedia (AH) research is *educational hypermedia systems*. Its specific features are as follows. It has a relatively small information space representing a particular course or section of learning material on a particular subject. The goal of the student is usually to learn all the learning material or a reasonable part of it. The hypermedia form supports student-driven acquisition of the learning material. The most important user feature in this area is user knowledge of the subject being studied. The adaptive hypermedia techniques can be useful to provide different content to different users and to the same user at different knowledge stages. It can be useful to suggest how to navigate through the hyperspace.

Another popular application for AH is the area of various *on-line information systems* and on-line documentation and encyclopedias. The goal of these systems is to provide reference access to information (rather than a systematic introduction as in educational hypermedia) for the users with a different knowledge level of the subject. The information space in these systems can range from small to very large. The user's knowledge, background and goal play an important role to provide help in finding relevant pieces of information. User goals can be provided by the users [Höök et al., 1996] or inferred by the system [Micarelli and Sciarrone, 1996; Höök et al., 1996].

Educational Hypermedia Systems Anatom-Tutor [Beaumont, 1994], C-Book [Kay and Kummerfeld, 1994a,b], <Clibbon> [Clibbon, 1995], ELM-ART [Brusilovsky et al., 1996a,c], ISIS-Tutor [Brusilovsky and Pesin, 1994, 1995], ITEM/PG [Brusilovsky et al., 1993; Brusilovsky and Zyryanov, 1993], HyperTutor [Perez et al., 1995; Perez, 1995], Land Use Tutor [Kushniruk and Wang, 1994], Manuel Excel [de La Passardiere and Dufresne, 1992], SHIVA [Zeiliger, 1993], SYPROS [Gonschorek and Herzog, 1995], ELM-PE [Brusilovsky and Weber, 1996], Hypadapter [Böcker et al., 1990; Hohl et al., 1996], HYPERCASE [Micarelli and Sciarrone, 1996]
On-line Information Systems Hypadapter [Böcker et al., 1990; Hohl et al., 1996], HYPERCASE [Micarelli and Sciarrone, 1996], KN-AHS [Kobsa et al., 1994], MetaDoc [Boyle and Encarnacion, 1994], PUSH [Höök et al., 1996], HYPERFLEX [Kaplan et al., 1993], CID [Boy, 1991], Adaptive HyperMan [Mathe and Chen, 1994]
On-line Help Systems EPIAIM [de Rosis et al., 1993], HyPLAN [Fox et al., 1993], Lisp-Ctritic [Fischer et al., 1990], ORIMUHS [Encarnacao, 1995], WING-MIT [Kim, 1995], SYPROS [Gonschorek and Herzog, 1995]
Information Retrieval Hypermedia CID [Boy, 1991], DHS [Shibata and Katsumoto, 1993; Katsumoto et al., 1994, 1996], Adaptive HyperMan [Mathe and Chen, 1994], HYPERFLEX [Kaplan et al., 1993], WebWatcher [Armstrong et al., 1995]
Institutional Hypermedia Hynecosum [Vassileva, 1994, 1996]
Personalized Views Basar [Thomas, 1995; Thomas and Fischer, 1996], Information Islands [Waterworth, 1994]

Table 2.1: Classification of AHS according to their application areas

On-line help systems are very close to on-line information systems. These systems serve on-line information about computer applications, such as a spreadsheet, programming environment, or expert system, to help users to use the systems. Like on-line information systems, they serve different information to different users. They are different from on-line information systems in the sense that they are not independent from their application systems and their information space is reasonably small. At the same time, the problem of helping users to find relevant pieces of information is not too difficult because the information space is not large and the system knows the context from which the user called for on-line help (context-sensitive help). So the context of work is a reliable source to offer the most relevant help items [Encarnacao, 1995; Grunst, 1993; Kim, 1995].

Information retrieval (IR) hypermedia systems form a new class of IR systems which combine traditional information retrieval techniques with a hypertext-link-based access from the index terms to documents. The systems provide the possibility of browsing the information space using similarity links between documents [Agosti et al., 1995; Helmes et al., 1995]. They may use browsing instead of constructing a proper formal query to find the required documents. They are different from on-line information systems. Firstly, they have a very large information space and the hyperspace can not be constructed "by hand" as in on-line information systems. Secondly, their users are more often professionals in different areas who use the system in their everyday work with different IR goals. An existing adaptive IR system [Kok, 1991] shows some ways to help the user in pure IR settings. Adaptive IR hypermedia systems can offer some additional help by limiting the navigation choices [Boy, 1991; Mathe and Chen, 1996], and by suggesting the most relevant links to follow [Kaplan et al., 1993; Katsumoto et al., 1996; Mathe and Chen, 1996].

Another area of application for adaptive hypermedia is *institutional information systems* which serve on-line all the information required to support the work of some institution, for example a hospital [Vassileva, 1996]. It is work-oriented in the sense that most users only need to access a specific area of the information space relevant to their work. This is significantly different from search-oriented IR hypermedia and on-line information systems where the "working area" of a user is the entire information space. These kinds of systems need assistance in organizing a more convenient personalized access to their work area [Vassileva, 1996]. Similar to educational hypermedia, these systems try to help new users to avoid getting lost even in their (relatively small) professional subarea.

The last of the application areas consists of *systems for managing personalized views in information spaces* such as Information Islands [Waterworth, 1994] and Basar [Thomas, 1995]. By defining their personalized views on the entire hyperspace, users protect themselves from the complexity of the overall hyperspace. Each view can be devoted to one of the goals, background or interests related with the work of the user [Thomas, 1995; Brusilovsky and Eklund, 1998]. Similar to institutional hypermedia users of these systems need a convenient access to a subset of an information space for everyday work. Personalized views in world-wide information spaces require permanent management because of the dynamic character of the information space.

Brusilovsky pointed out that the six application areas are not mutually exclusive. Some of them are "pairwise" similar and share the same problems. These pairs are: IR hyper-

media and on-line information systems, on-line information/help systems and educational hypermedia, educational hypermedia and institutional hypermedia, institutional hypermedia and information space management systems. Also, the difference between neighboring areas is not always clear-cut and some systems belong to both areas, for example, Hypadapter [Hohl et al., 1996] and HYPERCASE [Micarelli and Sciarrone, 1996] share features of educational hypermedia and on-line information systems, and HYPERFLEX [Kaplan et al., 1993] shares features of on-line information systems and IR hypermedia.

New applications	ELM-ART [Weber and Specht, 1997], Medtec [Eliot et al., 1997], AST [Specht et al., 1997], ADI [Schoch et al., 1998], Hy-SOM [Kayama and Okamoto, 1999], AHM [Pilar da Silva, 1998], CHEOPS [Negro et al., 1998], PATH [Hockemeyer et al., 1998], TANGOW [Carro et al., 1999], Arthur [Gilbert and Han, 1999], PAKMAS [Süßet al., 1999], CAMELEON [Laroussi and Benahmed, 1998]
New frameworks	InterBook [Brusilovsky et al., 1998], KBS-Hyperbook [Henze et al., 1999], AHA! [De Bra and Calvi, 1998b], SKILL [Neumann and Zirvas, 1998], Multibook [Steinacker et al., 1999], ACE [Specht and Opermann, 1998], ART-Web [Weber, 1999], MetaLinks [Murray et al., 1998]

Table 2.2: AHS in education after 1996

After 1996: Since 1996 these areas have been expanding at different paces. Educational hypermedia and on-line information systems are now established leaders. They account for about two thirds of the research efforts in adaptive hypermedia. Information retrieval (IR) hypermedia is challenging the leaders. The traditional scope of IR hypermedia was extended and now includes also systems for managing personalized views. On-line help systems and institutional hypermedia have received almost no attention in the last few years. One possible reason is that these kinds of systems are still in the process of transition from stand-alone hypermedia to Web-based hypermedia: stand alone versions of these systems are not attracting adaptive hypermedia researchers anymore and Web-based versions are not yet mature. We summarize the new extensions in the first three areas below from Brusilovsky's new review of adaptive hypermedia [Brusilovsky, 2001].

Classic on-line information systems	SWAN [Garlatti et al., 1999], ECRAN Total [Geldof, 1998], ELFI [Schwab et al., 2000]
Electronic encyclopedias	PEBA-II [Milosavljevic, 1997; Hirashima et al., 1998; Signore et al., 1997]
Information kiosks	AVANTI [Fink et al., 1998]
Virtual museums	ILEX [Oberlander et al., 1998], Power [Milosavljevic and Oberlander, 1998], Marble Museum [Paterno and Mancini, 1999], SAGRES [Bertoletti and da Rocha Costa, 1999]
Handheld guides	HYPERAUDIO [Not et al., 1998], HIPS [Oppermann and Specht, 1999]
E-commerce systems	SETA [Ardissono and Goy, 1999], TELLIM [Milosavljevic and Oberlander, 1998; Joerding, 1999]
Performance support systems	ADAPTS [Brusilovsky and Cooper, 1999], MMA [Francisco-Revilla and Shipman, 2000; de Carolis et al., 1998]

Table 2.3: AHS serving on-line information after 1996

All the early *educational hypermedia systems* were essentially lab systems, built to explore some new methods that used adaptivity in the educational context. Since 1996 the introduction of the Web has impacted both the number and the type of systems being developed. The choice of the Web as a platform has become a standard. Several more

recent systems provide complete frameworks and even authoring tools. This indicates the maturity of adaptive hypermedia and a response to a Web-provoked demand for user-adaptive distance education courses. Among these systems, some are extended versions of old systems, some are newly developed systems. Table 2.2 shows new educational applications and new complete frameworks.

Since 1996 the area of *on-line information systems* grew into many subgroups (see Table 2.3). These are "classic" online information systems, electronic encyclopedias, information kiosks, virtual museums, handheld guides, e-commerce systems, and performance support systems. Along with "classic" on-line information systems, these systems provide specialization by taking into account a specific type of user activity in a particular application area to provide better adaptivity and special kinds of adaptive behavior. Electronic encyclopedias and information kiosks remain very close to classic on-line information systems. However, they enhance these by providing some specialized enhancements that are not possible in generic systems by tracing user browsing. For example, an encyclopedia can trace user knowledge about different objects (for example, animals) described in the encyclopedia and provide adaptive comparisons [Milosavljevic, 1997]. Or it can trace user browsing, deduce his or her interest and offer a list of most relevant articles [Hirashima et al., 1998].

Virtual museums and handheld guides retain some similarity with traditional information systems and have the same structured hyperspace of objects in their core. The unique feature of these systems is the ability to provide adaptive guided tours in this hyperspace, and to support the user's exploration of a virtual or real museum with context-adapted narration. Such handheld guides can trace and support user navigation both in the physical museum space and in a virtual hyperspace.

E-commerce systems and performance support systems have diverged quite far from classic on-line information systems. While a hyperspace of information items still constitutes a major part of these systems, browsing of the information space is not a major activity, but is a byproduct of the major activity (such as performing a particular job or shopping for goods). The better these systems work, the less browsing should be required. These systems have information about the context of the user's work and the structure of the user's goal. This results in a higher level of precision in user modeling, and in a superior level of adaptation.

From the large number of adaptive *IR hypermedia systems* developed to date, we can distinguish four groups; search-oriented systems, browsing-oriented systems, systems for managing personalized views, and information services (see Table 2.4).

Search-oriented systems provide a list of links to documents that satisfy the user's current information request by taking into account not only the set of words specifying the current request but also a long-term/and short-term model of users' interests and preferences.

Browsing-oriented systems support their users in the process of search-driven browsing. As in other types of adaptive hypermedia systems this is done through standard adaptive navigation support technologies. Adaptive guidance systems mark one or more links on the current page that are most relevant to the user's goal. Adaptive annotation systems

attach various visual cues to the links on the current page in order to help the user select the most relevant one. Adaptive recommendation systems attempt to deduce the user's goals and interests from his or her browsing activity, and build a list of suggested links to nodes that usually cannot be reached directly from the current page, but are most relevant to that user.

Search oriented adaptive IR hypermedia systems	
Classic IR in web context	SmartGuide [Gates et al., 1998]
Search Filters	Syskill and Webert [Pazzani et al., 1996; Marinilli et al., 1999]
Browsing oriented adaptive IR hypermedia systems	
Adaptive Guidance	WebWatcher [Joachims et al., 1997], Personal WebWatcher [Mladenic, 1996]
Adaptive Annotation	Syskill and Webert [Pazzani et al., 1996], IfWeb [Asnicar and Tasso, 1997]
Adaptive Recommendation/Closed Corpus	SiteIF [Stefani and Strapparava, 1999], PEBA-II [Hirashima et al., 1998, 1999]
Adaptive Recommendation/Open Corpus	SurfLen [Fu et al., 2000], Letizia [Lieberman, 1995], IfWeb [Asnicar and Tasso, 1997]
Systems for managing personalized views	
Adaptive Bookmark Systems	WebTagger [Keller et al., 1997], PowerBookmarks [Li et al., 1999], Siteseer [Rucker and Polanco, 1997]
Information services	
Search Services	FAB [Balabanovic and Shoham, 1997], PEA [Montebello et al., 1998], Edited AH [Höök et al., 1997; Newell, 1997]
Filtering Services	ELFI [Schwab et al., 2000], AIS [Billsus et al., 2000]

Table 2.4: AHS related to IR problems in 1996-2001

Systems for managing personalized views belong to the IR universe and can be considered complementary to classic IR hypermedia systems. While IR systems help users to locate relevant information nodes, personal view management systems aim to organize

this information in some way. In the context of the Web, MyYahoo or MyNetscape and bookmark organizers are personalized site views. The majority of personalized sites and bookmark organizers are adaptable but not adaptive. There are few adaptive bookmarking systems such as WebTagger [Keller et al., 1997], PowerBookmarks [Li et al., 1999] and Siteseer [Rucker and Polanco, 1997].

The new class of IR hypermedia systems, Web-based information services work by collecting a common pool of documents from an open corpus hyperspace over a long period of time. They work with a community of users, and have the opportunity to learn about both the pool of users and the pool of documents. In doing so, they can provide different kinds of user support with the same adaptation engine by applying both content-based and clique-based (collaborative) technologies. Information services are typically built using agent-based technology. Information services have the potential to provide all the known types of IR hypermedia services from search to managing personalized views.

2.3.3 Features used in AHS

In adaptive hypermedia systems before 1996, the adaptation was based on taking into account various characteristics of their users represented in the user model. Currently the situation is different. A number of adaptive Web-based systems are able to adapt to something other than user characteristics. Kobsa et al. proposed distinguishing adaptation to user data, usage data, and environment data. User data comprise the traditional adaptation target, encapsulated by various characteristics of the users. Usage data comprise data about user interaction with the systems that can not be resolved to user characteristics (but still can be used to make adaptation decisions). Environment data comprise aspects of the users' environment that are not related to the users themselves.

User's characteristics: What user features can be taken into account when providing adaptation? Brusilovsky identified five features used by many existing adaptive hypermedia systems: knowledge, goals, background, hyperspace experience, preferences, and two newer features: interests and individual traits.

- **Knowledge:** A user's knowledge of the subject represented in the information space appears to be the most important feature of the user for existing adaptive hypermedia systems. It is used by about one third of all adaptation systems. It is a variable for a particular user. This means that an adaptive hypermedia system that relies on the user's knowledge has to recognize the changes in the user's knowledge state and update the user model accordingly.

  The user's knowledge of the subject is most often represented by an overlay model (Hypadapter [Böcker et al., 1990; Hohl et al., 1996], EPIAIM [de Rosis et al., 1993], KN-AHS [Kobsa et al., 1994], ITEM/PG [Brusilovsky et al., 1993; Brusilovsky and Zyryanov, 1993], ISIS-Tutor [Brusilovsky and Pesin, 1994, 1995], ELM-ART [Brusilovsky et al., 1996a,c], SHIVA [Zeiliger, 1993], HyperTutor [Perez et al., 1995; Perez, 1995]) which is based on the structural model of the subject domain. Generally,

the structural model is represented as a network which represents the structure of the subject domain. An overlay model represents an individual user's knowledge as an "overlay" of the domain model. For each domain concept, an individual overlay model stores an estimate of the user's knowledge level for some concept. Overlay models are powerful and flexible, they can represent a user's knowledge of individual topics. They were originally developed in the area of intelligent tutoring systems and student modeling [Greer and McCalla, 1991]. But the overlay model has the problem of initialization. Another simpler user model, the *stereotype* user model is often used to initialize the user model. A stereotype user model distinguishes several typical or "stereotype" users which have preset values for the domain overlay.

- **Goals:** A user's goal or task is a feature related to the context of a user's work with the hypermedia system rather than with the user as an individual. It tells what the user wants to achieve by using the hypermedia system. In application systems, a user's goal is a work goal; in information retrieval systems, a user's goal is a search goal; in educational systems, a user's goal is a problem solving or learning goal.

- **Background and experience:** A user's background describes all the information related to the user's previous relevant experience *outside* the subject of the hypermedia system. For example, the user's profession, experience of work in related areas, and the user's point of view and perspective. The user's experience in the given hypermedia means how familiar the user is with the structure of the information space (not its contents) and how easy the user navigates in it. These two user features are usually modeled by a stereotype user model.

- **Preferences:** Preferences are user features describing that a user can prefer some types of nodes and links over others. They can indicate e. g. preferred colors, learning or navigation styles, etc.

- **Interests:** The user's interests have been adopted in web IR hypermedia systems that attempt to model the user's long-term interests, and use these in parallel with the user's short-term search goal in order to improve the information filtering and recommendations. This feature is also becoming popular in various online information systems.

- **Individual traits:** The user's individual traits is a group name for user features that together define a user as an individual. They are similar to user background in that they are stable features of a user, but different in the sense that they are extracted not by a simple interview, but by specially designed psychological tests.

Environment: Adaptation to the user's environment is a new kind of adaptation that was brought on by Web-based systems. Adaptation decisions may depend on the user's location and the user platform. Simple adaptation to the platform, such as hardware, software and network bandwidth, usually involves selecting the type of material and media (e. g. still image vs. movie) to present the content [Joerding, 1999]. This direction of adaptation will

provoke new interesting techniques. Adaptation to the user's location becomes useful for many on-line information systems. SWAN [Garlatti et al., 1999] demonstrates a successful use of user location for information filtering in a marine information system. The current research in this area shows a number of interesting adaptation techniques that take into account user location, direction of sight and movements [Not et al., 1998; Oppermann and Specht, 1999].

2.3.4 Methods and techniques used in AHS

Following Brusilovsky [Brusilovsky, 1996, 2001] we distinguish between high level methods for adaptive hypermedia support and lower level techniques that are used to realize or implement that support. By a method we mean a notion of adaptation that can be presented at the conceptual level. A technique is then a way to implement a specific method. Techniques operate on actual information content and on the presentation of hypertext links. It may be possible to implement the same method through different techniques and to use the same technique for different methods. In this section we present different methods and techniques for adaptive presentation, and for adaptive navigation support.

Adaptive presentation

It may be desirable to present information on a certain topic in different ways, depending on the user's (fore)knowledge, goals, preferences or other characteristic properties. For example, a user with experience in the domain can be shown more specific, more detailed information, while a novice receives additional (and maybe introductory) explanations. In a hypermedia system the content of a "page" may not only be text, but it may also contain various items of other media types. We must remark however that the current generation of AHS only offers adaptive presentation for text, and just media selection for multimedia items.

Adaptive presentation methods: In [Brusilovsky, 1996] three main adaptive presentation methods are considered:

- additional, prerequisite, and comparative explanations
- explanation variants
- sorting

The most popular method appears to be the *additional explanations*. Its goal is to provide additional information, explanations, illustrations, examples, etc., to those users who appear to need or want them. At the same time the system hides such explanations from users who do not want them, for instance because of a mismatch in level of difficulty, or because they prefer terse and basic information only. Prerequisite explanations are to some extent a special case: an explanation is added because the system decides that without

this explanation the user will (or may) not understand the remainder of the page. The method *prerequisite explanations* is thus used to compensate for the lack of prerequisite knowledge. The method *comparative explanations* is used when there is information about other concepts that are similar to the one described in the "current" page (or dissimilar in a specific way, or otherwise related in some "interesting" way). Showing this additional information about how the current topic relates to these other concepts only makes sense when these other concepts are known to the user.

The method *explanation variants* is used in cases where all users need roughly the same information (or explanation), but they need a different presentation of it. Some variants may be more or less verbose, and may use or avoid specific technical terms for instance, and some may even use a different media type.

The method *sorting* refers to situations where the information about the concept the user wishes to study consists of more or less independent fragments that can be presented in any order. The AHS can sort these fragments from most to least relevant, depending on the user's background and knowledge.

Adaptive presentation techniques: We concentrate on techniques in what is called "canned text" adaptation. (This in contrast with adaptation by "generating" natural language or by filtering text using natural language understanding.) There are five kinds of techniques to implement the above mentioned methods:

- inserting/removing fragments
- altering fragments
- stretchtext
- sorting fragments
- dimming fragments

Inserting/removing fragments is equal to the term "conditional text" in the 1996 review. With it all available information about a concept is divided into several fragments of text (or multimedia content). With each fragment a (Boolean) condition is associated on elements of the user model. When displaying a page about the concept, the system only presents the fragments for which the condition is **true**. This is the lowest-level technique, but also the most flexible one. It can be used to "simulate" the four other techniques. With conditional text it is easy to implement the methods of additional, prerequisite and comparative explanations. However, because it is such a low-level technique writing conditional text makes authoring look like programming. The method of sorting cannot easily be implemented using this technique. Inserting/removing fragments is the technique used in the current AHA! system [De Bra and Calvi, 1998a].

Altering fragments is similar to the old term "fragment variants". With it one can easily implement explanation variants. The AHS stores several variants of the same information fragment, and selects the variant to display based on the user model. On one information

page there may be many fragments, each with several variants. The author needs to keep an overview of all possible combinations of fragment variants the system may select and present on one page. Altering fragments covers the old terms "page variants" and "frame-based techniques", because the fragment is the whole page in page variants, and the fragment is defined more fine-grained in frame-based techniques.

The Guide hypertext system [Brown, 1987] offered replacement links. Clicking on a word or phrase would open up a content fragment (paragraph) on that term. Parts could be opened and closed as desired. The technique of *stretchtext* is very similar. But while in Guide all fragments would initially be closed, in stretchtext the AHS determines which fragments to open (or stretch) when displaying a page. Stretchtext is useful for implementing additional, prerequisite or comparative explanations, but less for explanation variants and not at all for sorting.

Sorting fragments is similar to the old term "sorting" as an adaptive presentation method. It is more suitable to call it a technique of adaptive presentation. With sorting fragments the goal is to present the same information items to all users, and to order them from most relevant to least relevant or according to some other criterion, which not only depends on an explicit goal (such as a search request) but also on a user's background and foreknowledge.

Dimming fragments is a way to dim, shade or deemphasize (in some way) a fragment to indicate that it is not (or at least less) relevant for the user [Hothi and Hall, 1998]. As an adaptation technique it is new but it is a common (non-adaptive) technique for "sidebars" in magazines.

Adaptive navigation support

The basic idea with adaptive navigation support is to adapt the rich link structure in such a way that the user is guided towards interesting, relevant information, and kept away from non-relevant information. Adaptive navigation support tries to simplify the rich link structure to reduce orientation problems, while maintaining a lot of navigation freedom, which is typical of hypermedia systems.

Adaptive navigation support methods: Brusilovsky [Brusilovsky, 1996] mentions five adaptive navigation support methods:

- global guidance
- local guidance
- global orientation support
- local orientation support
- managing personalized views

Guidance can be provided in hypermedia applications where users have some goal in terms of information they want (they need information which is contained in one or several pages somewhere in the hyperspace) and where browsing is the preferred or only way to find the required information. *Global guidance* means that the system suggests navigation paths on a global scale. This is especially useful in educational hypermedia. When a user wishes to learn about a certain topic, the system may suggest a set of pages to read, along with an indication of a desired or at least meaningful reading order. *Local guidance* means that the system suggests the next step to take, for instance through a "next" or "continue" button.

Orientation support means that the system presents an overview of the whole (link) structure of the hyperspace (*global orientation support*), or of a part thereof (*local orientation support*). The system also indicates the position of the user (of the "current" page) in this structure. When the orientation support is adaptive it means that the structure which is shown indicates relevant parts to go to, parts that were visited before, and parts that are (still) to be avoided. Global orientation support is sometimes offered through a kind of "table of contents" column, frame or (sitemap) page in Web-based presentations. Local orientation support shows a small part thereof, like one or two (link) levels up or down from the "current" page.

Another way to protect users from the complexity of the overall hyperspace is to let them organize *personalized (goal-oriented) views*. Each view may be a list of links to all pages or sub-parts of the whole hyperspace that are relevant for a particular working goal. Classic hypermedia systems and modern WWW browsers suggest bookmarks and hotlists as a way to make personalized views. More advanced systems suggest some more high-level adaptability mechanisms based on metaphors [Waterworth, 1994] and user models [Vassileva, 1996]. Basar [Thomas, 1995] is an example of a system that provides adaptive management of personalized views. Basar uses intelligent agents to collect and maintain an actual set of links relevant to one of the user's goals. We have not found any generalization of existing techniques for managing personalized views.

Adaptive navigation support techniques: We use the new review of hypermedia of Brusilovsky [Brusilovsky, 2001] to describe adaptive navigation support techniques. The the new and the old review differ in two aspects. The first aspect is that the new overview has generalized "link hiding", "link disabling", and "link removal" to a more general technology called "link hiding". The second aspect is that it adds a new technology "link generation". Below we give a detailed description of these techniques for adaptive navigation support:

- direct guidance
- link sorting
- link hiding
  - link hiding

  - link removal
  - link disabling

- link annotation
- link generation
- map adaptation

Direct guidance means that the "best next" node for the user to visit is shown. It is an obvious choice to implement local guidance. The destination of this link is determined by the AHS, based on the user's goal and other elements from the user model. It is not like a static "next page" button.

The basic idea of adaptive *link sorting* is to sort all the links on a particular page according to the user model and to some goal-oriented criteria: the more towards the top of the page, the more relevant the link is. Adaptive sorting can be used for global guidance: all links from the node are sorted according to their relevance according to the global goal. Sorting is typical for information retrieval systems. However, the "teach me" feature of InterBook [Brusilovsky et al., 1996b] for instance illustrates that sorting can be used in an educational AHS as well.

With the technique of *link hiding* the navigation space is simplified by hiding links to "non-relevant" pages. With contextual links (links occurring as "hot words" or "anchors" within the text) hiding can be realized by changing the color of the anchors to that of normal text. For non-contextual links this technique is not very helpful. (In a list, for instance, items are clearly visible, whether in a typical link color or not.) In an index or map hiding can only be achieved by making the anchors transparent (or giving them the "background color"). However, this looks more like link removal than like hiding (see below).

The basic idea of *link removal* [De Bra and Calvi, 1998a] is that link anchors for undesired links (non-relevant or not yet ready to be read) are removed. This technique works well for non-contextual links, as in lists or maps, but cannot be used in running text, because the words that make up the link anchor can in general not be omitted from the sentence in which they appear.

The *link disabling* [De Bra and Calvi, 1998a] technique is based on the idea that the "link functionality" of a link is removed. This technique is not usable by itself. The disabled links must be either hidden or annotated so that the user does not expect the links to work.

The idea of the adaptive *link annotation* is to augment the link with some form of comment that tells the user more about the current state of the pages to which the annotated links refer. It is more powerful than hiding (as hiding is in essence also a form of annotation), but it does not reduce the cognitive overhead as much as hiding does. (The navigation space is not simplified by showing fewer links.) Link annotation informs the user about the current "state" of the pages the links point to. We have found three methods for deciding how to annotate links:

- The annotation may indicate the relevance of a link. Colors may be used to distinguish for instance "highly relevant", "somewhat relevant", "non-relevant".

- The annotation may indicate whether the user already knows the concepts described in the page a link points to. Several degrees of knowledge may be distinguished, for instance "not known", "known", "well known".

- The annotation may also indicate whether a user is able to understand the information contained in the destination page ("ready to be read" vs. "not ready to be read").

All three types of annotation can be used simultaneously (provided that the system is able to distinguish many link types in a visual way). For instance, links to non-relevant pages may be hidden, while links to relevant pages are colored differently depending on whether the user is "not ready to read" them, "ready to read them for the first time", or "has already read" them.

Recommender systems introduced a new kind of adaptive navigation support, to generate new, not-authored links for a page. *Link generation* includes three cases: discovering new useful links between documents and adding them permanently to the set of existing links; generating links for similarity-based navigation between items; and dynamic recommendation of relevant links.

Map adaptation is mentioned by Brusilovsky [Brusilovsky, 1996] as a separate technique, in which the content and presentation of a map of the link structure of the hyperspace is adapted. But in fact, the adaptation to the map can be seen as a combination of link removal operations, annotations and (maybe two-dimensional) sorting.

2.3.5 The main parts of adaptive hypermedia applications

Most adaptive hypermedia applications can be described as consisting of the three parts below. (This is just a conceptual view and does not imply that the systems actually have a separate representation for each of these parts.)

- An *information domain* contains the information content of the application. With "content" we mean not only the information items (fragments, pages) but also the relationships between these items. Every piece of content belongs in this part. So all "explanation variants", all versions of pages, etc. This part is an extension to the traditional view of hypertext consisting of nodes and links.

- A *user profile* for each user represents user features that are relevant to the adaptation, e. g. the user's preferences, knowledge, goals, navigation history and possibly other relevant aspects that are used to provide personalized adaptations. Part of this profile relates to the information domain and part relates only to the user and his or her environment. This part was described in Section 2.3.3.

- An *adaptation description* consists of a description of how to maintain the user profile by observing the user's (browsing) behavior, and how to generate adaptation. It uses the information from the information domain and the user profile. This part was described in Section 2.3.4

In the next chapter we will use this three part description of AHS as the basis for a reference model for adaptive hypermedia applications.

2.3.6 Examples of authoring adaptive hypermedia applications

Authoring an adaptive hypermedia application involves creating the three main parts mentioned in the previous subsection. Different systems take a different approach towards authoring. For example InterBook [Brusilovsky et al., 1996b] requires authors to create annotated MS Word files. Each word file defines a page for a section of the textbook. Each page contains a description of the information content and relationships between pages. The author connects pages to concepts, meaning that the user will learn about a concept when reading a page. The author also defines which concepts are prerequisites for which other concepts. InterBook creates and maintains a user profile as an overlay model. InterBook also performs all adaptation automatically in a fixed way, as we shall describe in Chapter 5. For InterBook an author thus creates only the *information domain*.

AHA! [De Bra et al., 2000] requires authors to create annotated HTML or XML files for the information pages. The conceptual structure must be defined in a separate XML file. This file also defines how the system will update the user profile, based on the user's browsing behavior. Another XML file defines the (prerequisite) requirements that determine when pages are "recommended". Inside the information pages the author must add conditions for fragments. AHA! has some predefined behavior, but the author has to create not only the *information domain* but also most of the *user profile* and the *adaptation description* (and both the user profile update part and the adaptation generation part). However, from the above description one can already see that the three parts are not clearly separated in AHA!. The structure of the user profile is defined together with the definition of how the profile is updated. Also, the information pages are mixed with adaptation rules for the conditional inclusion of fragments.

The authoring processes for InterBook and for AHA! illustrate shortcomings in current AHS and show how authoring should be improved in the future. Authoring should be made easy in the sense that the author should be able to describe adaptation in a simple way. On the other hand an author should also have the opportunity to define how the user profile is updated and how the adaptation is performed. In InterBook the authoring is easy, but the system behavior is all "hardcoded", so an author cannot change how the adaptation is performed. In AHA! an author can define how the user profile is updated and how adaptation decisions should be made by the system. But authoring is complicated, mostly because the descriptions of the information domain, the user profile and the adaptation are mixed instead of clearly separated.

2.4 Summary of the research background

Hypertext systems provide navigational freedom: users can read pages in many different orders. When using a hypertext application users may experience personalization because they can follow any link they want. However, the personalization is only "perceived" and not real. Traditional hypertext systems use a "one size fits all" approach: they do not take individual users' features into account. So in answer to research question 1 we can state that hypertext systems do not offer any personalization, even though it may appear to users like they do. Hypertext systems suffer from inherent navigation and orientation problems. Many hypertext systems try to solve the problem by providing some navigation support based on the conceptual structure of the information.

In answer to research question 2 we have seen that adaptive hypermedia systems approach navigation and orientation problems by providing adaptive content and adaptive navigation support. These systems maintain a user model to trace the users' browsing behavior. AHS can make hypertext more useful and usable in the sense that they personalize the information according to users' needs. AHS are being used in many application areas, but the majority is currently targeting educational applications. We realize that we have not fully answered research question 2. We have shown how adaptive hypermedia systems *try* to solve the problems of (non-adaptive) hypermedia systems, but we have not shown that these attempts actually work.

Adaptive hypermedia applications can be described as consisting of three main parts: information domain, user profile, and adaptation description. We will use this approach in the reference model that we describe in the next chapter. Different adaptive hypermedia systems may require authors to create these three parts in different ways, sometimes complicating the authoring task by mixing these three parts. The reference model we developed and describe in this dissertation advocates the clear separation of these parts in order to provide rich adaptation flexibility to authors in an architecture that is easy to understand and to use.

Chapter 3

AHAM: Adaptive Hypermedia Application Model

This chapter presents the Adaptive Hypermedia Application Model (in short AHAM) which describes the adaptation functionality of AHS at an abstract level. AHAM is an extension of the Dexter reference model [Halasz and Schwartz, 1990] for hypermedia applications. AHAM advocates a clear separation of three parts of an adaptive application: (a) the information domain (b) a representation of the user and how the user "relates to" the information content, and (c) a description of how the AHS adapts the information (presentation and navigation) to the user. We call these three parts the *domain model* (DM), *user model* (UM), and *adaptation model* (AM) in AHAM. The AHAM model was first introduced in [De Bra et al., 1999].

Section 3.1 introduces the overall structure of AHAM. Section 3.2 describes the domain model which consists of a set of concepts and concept relationships. The DM supports different "levels" of concepts, from very general abstract concepts to more specific and concrete concepts that correspond to a single Web page, and even to smaller information fragments such as a paragraph or sentence. Section 3.3 describes the user model as a fine-grained overlay of the DM. The UM also captures other user features that are used in adaptation. Section 3.4 describes the adaptation model which consists of a set of rules to describe how to update the UM and how to generate adaptive presentation specifications. Section 3.5 describes the communication between different adaptive hypermedia systems by including information exchange functions in AHAM. When different AHS can import each other's user models (of the same users) they have more information to base their adaptation on. Section 3.6 summarizes features of AHAM.

3.1 Introduction

After studying the current generation of AHS in Chapter 2, we are now going to design a general reference model for adaptive hypermedia systems, called AHAM for Adaptive Hypermedia Application Model. AHAM is an extension of the Dexter reference model for

hypermedia systems. AHAM should cover the three main parts (Section 2.3.5) of adaptive hypermedia applications. We define three sub-models, the domain model (DM), the user model (UM), and the adaptation model (AM) to describe the three parts, the information domain, the user profiles, and the adaptation description. We pay particular attention to the issue of how to make authoring easier when we discuss the requirements to design these sub-models of AHAM.

The first aim of AHAM is to be a *reference model* for AHS. This means that it should be possible to describe the structure and functionality of existing (and hopefully also future) AHS in the model. A secondary aspect of a reference model is that the descriptions of different AHS may enable us to compare different AHS and possibly even to define a translation of an adaptive hypermedia application from one system to another. Because we already have studied and described a reference model for hypermedia systems we first look at the Dexter model to see whether we can use it to describe AHS. Our choice for the Dexter model is motivated by the facts that Dexter is a well-known reference model for hypermedia systems, that it has been used as a standard for many years, and that it has been formalized and even implemented. Should the Dexter model (as we demonstrate) not be able to express the full functionality of AHS we can then try to extend it. Extending the Dexter model not only gives us a reference model for AHS but also shows that AHS can be described as extensions to hypermedia systems in general (thereby answering research question 3).

In the Dexter reference model, as described in Chapter 2, the Storage Layer describes the information domain at an abstract level. (The concrete content is implementation dependent and hidden in the Within-Component Layer.) The Storage and Within-Component Layers treat hypertext as an essentially passive data structure [Halasz and Schwartz, 1990]. They do not story any user-specific information. Everything relating to accessing and presenting the information to the user belongs in the Run-time Layer. This layer offers a *session* entity to keep track of the interaction between the user and the system. Session information however is not stored permanently. It therefore cannot be used to represent the persistent (and constantly updated) user profile and the adaptation based on that. We can thus conclude that the Dexter model cannot be used directly to represent AHS. It cannot describe any personalization and adaptation to a persistent user profile.

As argued in Chapter 2 an AHS needs to store persistent information about a user in order to be able to perform adaptation. This information cannot be represented in the Dexter model. Therefore we extend the Dexter model by including a user profile and adaptation in the Storage Layer, without throwing away the structures and functions Dexter already offers. We can use Dexter for the basic hypertext functionality and focus our description on the extensions for adaptation.

As shown in Figure 3.1, AHAM focuses on the *Storage Layer* and uses the *anchoring* and the *presentation specifications* as an interface to the Within-Component Layer and the Run-time Layer. In AHAM the Storage Layer consists of three parts: a domain model (DM), which describes the information domain, a user model (UM), which describes user features used in adaptation, and an adaptation model (AM), which describes the adaptation

strategies. With the help of the user model and the domain model, the adaptation model specifies how to update the user model and how to generate adaptation. We will describe the details of each sub-model of AHAM in the following sections.

The "T" structure of the Storage Layer, shown in Figure 3.1 stems from the boundaries and interaction between the different sub-models. The AM generates the presentation specifications so it sits right below that layer. Adaptation rules use the structure of the information domain and the user profile to decide how to perform the adaptation. It thus sits between the DM and the UM. All connections between the actual information content (in the Within-Component Layer) and the submodels of the Storage Layer use connection points specified in the Anchoring layer.

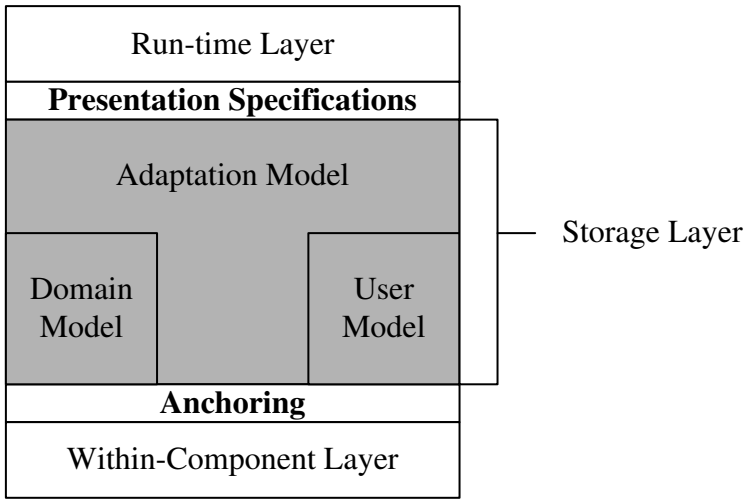

Figure 3.1: The AHAM model

3.2 The Domain Model (DM)

In the Dexter model, the central notion of the Storage Layer is the *component*. This notion covers both *nodes* and *links*. In adaptive hypermedia we extend the term component to *concept component*, or just *concept*. Instead of link components, AHAM uses *concept relationships*.

3.2.1 Concepts

In the Dexter model, the component may represent a very small amount of information or a very large amount of information for a user to read. Users simply receive what is defined in the nodes, which are considered "atomic" from the point of view of the Storage Layer. Dexter also considers composite components, but few systems at that time used them. For adaptive applications two requirements change this situation:

- Several adaptive presentation techniques deal with (small) parts of a node or page: inserting/removing fragments, altering fragments, stretchtext and fragment sorting (see Section 2.3.4). In order to be able to describe these techniques at an abstract level in the DM, we consider *pages* as composites consisting of *fragments*.

- AHS often use the notion of *prerequisites*, e. g. for *prerequisite explanations* (see Section 2.3.4). In order to decide whether a user has sufficient prerequisite knowledge the system may have to consider whether the user has already read a number of pages about the same topic. To make it easy to define such prerequisites we need the ability to combine pages into larger conceptual units, which we call *concepts*. (Without concepts an author would have to include all the pages of a concept in a description of a prerequisite.)

Because of the above reasons we introduce three levels in the DM. We introduce a *page* component to capture the amount of information that is presented to a user at any one time. For a larger amount of information, an *abstract composite* component is used (or simply *abstract concept* or just *concept* if there is no confusion). A composite component consists of a number of pages and/or other (smaller) composite components. A *page* can consist of a number of smaller items that can be included or left out, but that cannot be changed by the AHS. They are "opaque" to the AHS. We call them *atomic* concepts or *fragments*. (We use these terms interchangeably in the sequel.)

The structure formed by *fragments*, *pages* and *abstract concepts* must form a hierarchy. Figure 3.2 illustrates a part of an example concept hierarchy. In this figure we see the three levels of concepts: at the lowest level, we have the fragments (that cannot be changed by the AHS). At the middle level, we have pages which are the units of information presented to the user for each interaction. Pages are bundled into abstract concepts and in the same way we can build higher level abstract concepts. We use the constraint that the composition structure must be a directed acyclic graph (DAG). Thus, no component may contain itself as a subcomponent (directly or indirectly). Also, every atomic concept must be included in some page (possibly more than one).

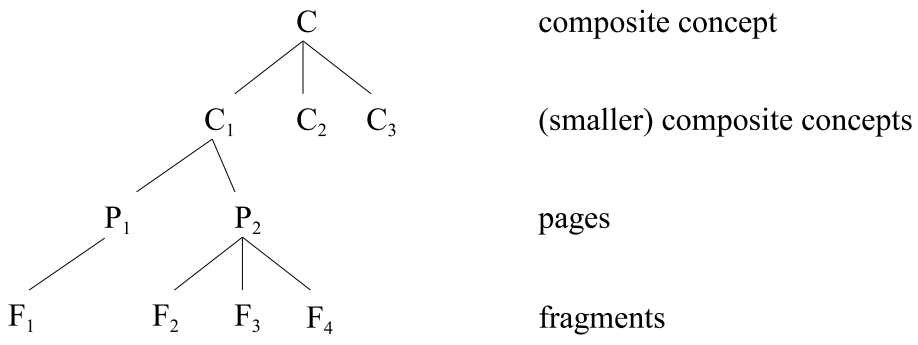

Figure 3.2: An example of a concept hierarchy

After this informal introduction we now define the AHAM model in a semi-formal way.

Definition 3.1 *A* concept component *(or* concept *for short) is an abstract representation of an information item from the application domain. A concept is a pair* $<$ uid, cinfo $>$*, where uid is a globally unique (object) identifier for the concept, and cinfo is the component information. A component's information consists of:*

- *a* set of attribute-value pairs
- *a* sequence of anchors
- *a* presentation specification

The structure of attribute values and anchors are described below. The presentation specifications are not described in detail in AHAM. (They are also not described in detail in the Dexter model.)

The Dexter model distinguishes atomic from composite components. AHAM does the same for concepts.

Definition 3.2 *An* atomic concept component *corresponds to a* fragment *of information. It is primitive in the model. Its anchor values and possibly some attribute values belong to the* Within-Component Layer. *They are not described in AHAM (or Dexter).*
A composite concept component *has a* children *attribute, which (has a value that) is a sequence of concepts. It has a* constructor *attribute, which indicates the "structure" of the composite, and which indicates "how much" of the composite each subcomponent represents. We distinguish two kinds of composite concepts:*

- *An* abstract *composite concept has only children that are composites themselves.*
- *A* page *composite concept has only children that are atomic concepts (i. e. fragments).*

The composite concept component hierarchy must be a directed acyclic graph, i. e. no component can be a subcomponent of itself, either directly or indirectly. Also, in AHAM every atomic (fragment) concept is required to be a subcomponent of at least one page concept.

Note that composites that have composite as well as atomic children can be simulated by introducing extra intermediary composites. The restriction in our definition simplifies the implementation of *resolver* and *accessor* functions that translate uid's to components in order to allow their presentation. The resolver function can be specified in the adaptation model. The accessor function is part of the adaptation engine of the AHS. (The adaptation engine is an implementation of the adaptation functionality of AHS and is described in Chapter 4.)

It may seem that there is no information content. However, the content is an attribute of fragments, belonging to the Within-Component Layer. Also, information is accessed as

pages, constructed out of fragments. We require each fragment to be contained in at least one page because otherwise there is no way to access that fragment's information.

Dexter's notion of "composition" does not have any associated semantics. Dexter does not model the effect of a composite on link traversal, nor does it explicitly model the presentation specification of composites. To address the adaptation in the presence of composites we extend the Dexter model by introducing two functions in the adaptation model, the *page selector* and *page adaptation generator*. They are part of the Dexter model's accessor function.

Definition 3.3 *When a user accesses an abstract concept the concept's* page selector *selects one or more subcomponents to present. A selected subcomponent may itself still be an abstract component. The "selection" process is therefore repeated until all subcomponents are page concepts. A* page constructor *describes how the presentation specification of a page is generated. In AHAM we only describe the* page adaptation generation *that generates* adaptation attributes *for the presentation specification of the component.*

In AHAM we treat *adaptation attributes* just like normal attributes of a component. In the rule language described later we will for instance write $F.pres :=$ "*show*" to indicate that a fragment F should be shown. Page selectors, page constructors and adaptation generators can all be described in the adaptation model.

3.2.2 Concept relationships

Some AHS use relationships between concepts to provide effective adaptive content and navigation support. (An example are *prerequisite relationships* in educational applications.) Concept relationships exist in almost every application information domain. AHAM tries to provide a framework to allow the description of any concept relationship in order to let the systems use this information to provide effective or more sensible adaptation automatically.

In the Dexter model, a component is either an atom, a composite or a link. Links represent relationships between components. While the term *link* suggests that these relationships are used for navigation, the model does not require that. In Dexter, the Storage and Within-Component Layers treat hypertext as an essentially passive data structure, so it does not require links to have the dynamic property that they can be used for navigation.

In order to represent semantic (non-navigational) relationships between concepts within the Dexter framework we define *concept relationships* as *link components*. They can be used for any type of *relationship between components (concepts)*. As shown in definition 2.4 the Dexter model also allows links of which (some) endpoints are links. We do not consider such relationships, mostly because we know of no existing AHS that offers this feature.

In AHAM, as well as in the Dexter model, the *sequence of anchors* of a concept component provides a way for links to be attached to a specific part of a component (in a media or implementation independent way):

Definition 3.4 *An* anchor *is a pair* <aid, avalue>*, where* aid *is a unique (object) identifier for the anchor within the scope of its component and* avalue *is an arbitrary value that specifies some location, region, item or substructure within a concept component. Anchor values of atomic components belong to the* within-component layer *and are not elaborated in AHAM (or Dexter). Anchor values of composite components are* uid-aid *pairs, where the* uid *identifies a subcomponent of the composite and the* aid *identifies an anchor within that subcomponent.*

Concept relationships in AHAM, just as *links* in the Dexter model, consist of a list of components. These components however are not given as a unique identifier, but rather as a *specifier* that needs to be *resolved* to an identifier (or a set of identifiers). In AHAM, unique identifiers are used as specifiers, but these are not (always) the identifiers of the "real" endpoint of a link or the element in a concept relationship. When the endpoint or element is a composite concept component, it needs to be "resolved" by traversing the composite hierarchy down to the page level. This is what the *page selector* does that we described earlier.

Definition 3.5 *A* specifier *is a tuple* <uid, aid, dir, pres> *where* uid *is the uid of a concept component,* aid *is the id of an anchor,* dir *is a direction (which is* FROM*,* TO*,* BIDIRECT *or* NONE*), and* pres *is a* presentation specification*.*

At first sight, it appears that there are no "computable" specifiers. However, the computable part is hidden in the *page selector* (corresponding to the Dexter model's *resolver* function) that must translate the uid of an abstract concept to the uid of one or more pages to be presented.

Definition 3.6 *A* concept relationship component *is a component* <uid, ss, cinfo> *where* uid *is the identifier of the relationship component,* ss *is a* sequence of (two or more) specifiers *and* cinfo *is the component information which consists of:*

- *a* set of attribute-value pairs*; this set must include an attribute-value pair to indicate the* type *of the relationship.*
- *a* sequence of anchors *(not used in AHAM)*
- *a* presentation specification

A common type of relationship is the *link*, which corresponds to *link components* in the Dexter model and which are used for navigation. However, in AHAM we consider other types of relationships as well that play a role in the adaptation. Figure 3.3 shows a small example of a set of concepts and concept relationships. In this example we assume the type link to represent navigational links between concepts, e. g. a concept C_1 has a link to concept C_2. We also assume two other types of concept relationships. The type "prerequisite" describes a certain desirable reading order between concepts. When a concept C_1 is a prerequisite for C_2 it means that the user should read C_1 before reading C_2.

It does not imply that there must be a (direct navigational) link from C_1 to C_2. It only means that the system must somehow take into account that "visiting" C_2 is not desired before C_1 has been "visited".

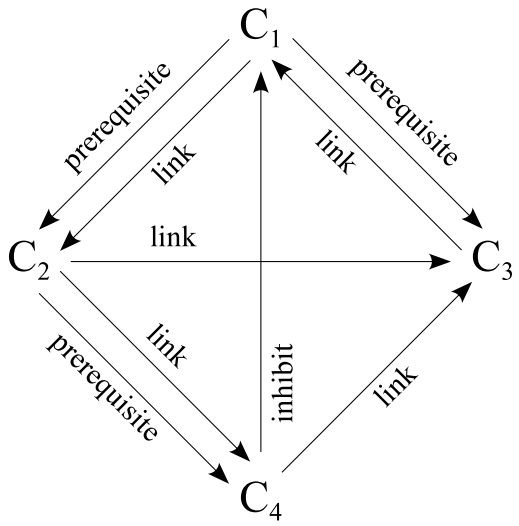

Figure 3.3: An example of concept relationships among concepts

Note that a graph presentation like Figure 3.3 only works for binary concept relationships (such as simple links, some prerequisites and inhibitors). AHAM allows relationships between a (longer) sequence of concepts, in order to be able to describe features like guard fields in Storyspace [Bernstein et al., 1991] and "requirement" relationships in AHA! [De Bra and Calvi, 1998c].

In AHAM authors or system designers can (in theory) define arbitrarily many types of relationships. In fact, one can even imagine an AHS without links (as in spatial hypertext [Marshall and Shipman, 1995]). A relationship typed "inhibit" can describe an unusual type of desirable reading order: C_4 "inhibits" C_1 means that after visiting C_4 it is no longer desirable to visit concept C_1. In Section 3.4 we show that one way to make the "desirability" of a link destination clear to the user is to use different presentations of link anchors, e. g. by giving them a different color.

In actual AHS there is usually a fixed set of concept relationship types. This may limit the use of the AHS to a certain kind of applications. For authors who only need the existing types of relationships the creation of applications is easy. But when a different type of relationship is needed authors may have to play "tricks" with the available types to create the adaptation they want. We will see examples of this in Chapter 6 where we describe how to perform certain adaptation in the AHA! system [De Bra et al., 2000]. New developments in AHA! include the ability to define concept relationship types. Unfortunately usability experiments have not yet been performed at the time of writing this dissertation.

Definition 3.7 *The atomic concept components, page concept components, composite concept components and concept relationship components together form the* domain model (DM) *of an adaptive hypermedia application.*

The domain model shows the combination of concepts and concept relationships. The concept hierarchy describes the structure of concepts at many different levels, allowing authors to describe adaptation using concepts at any level. By using concept relationships between concepts, authors can describe adaptation based on semantic relationships, and not just on navigational links.

3.3 The User Model (UM)

Adaptive hypermedia systems distinguish themselves from other hypermedia systems by maintaining a (permanent and continuously updated) *user model* (UM). As explained in Section 3.1 that the UM is part of the *Storage Layer* because it needs to be persistent accross session boundaries (so it cannot be in the Run-time Layer) and it needs to be described in detail (so it cannot be in the Within-Component Layer). In Section 3.3.1 we describe the structure of the UM. In Section 3.3.2 we explain how user model updates are triggered. User modeling in AHAM was first published about in [Wu et al., 1999b].

3.3.1 The UM - an overlay of the DM

Most AHS store a user's knowledge about or interest in concepts. Some AHS also want to store a user's preferences, such as a preferred media type (text, audio, video), a desired verbosity level (terse, medium, verbose), link annotation type (hiding, annotation, and if so, using which color scheme); and a user's background information, such as experience in the use of adaptive hypermedia systems, possible disabilities, age, education, etc. Other systems may want store information about a user's environment. In summary, AHS should be able to adapt to three types of information:

- stable personal characteristics of the user, such as interests, background, learning style, etc.
- aspects of the user's environment, including place, time, used equipment (computer, PDA, phone) and network
- the relation between the user and the concepts in the domain model, such as knowledge about or interest in each concept

In this dissertation we are mostly interested in the part of the user model that deals with the third aspect: how the user relates to the domain model when browsing the hyperspace. We therefore build the UM as an overlay model of the DM. For each concept in the DM, the UM stores attributes values about the concept. The UM is fine-grained, based on the DM being well-structured at many different concept levels (from high level concepts down

to fragments). It is the author's responsibility to define concepts at the appropriate level of abstraction and concreteness for their applications.

Of course one could simulate personal characteristics and environmental factors through "pseudo concepts". Doing so would enable AHAM to also describe AHS that take these other aspects into account, albeit in an artificial way.

Definition 3.8 *An AHS associates a number of user-model attributes with each concept component of the domain model. This user model (UM) is a set of* $< uid, uinfo >$ *pairs, where uid is the unique identifier of a DM concept and uinfo is a set of attribute-value pairs. An AHS maintains a UM (instance) for each user.*

We often use an alternative view of the UM, in which we consider the UM to be a (sparse) table. The columns of the table are first the uid of a concept, and then all the attributes that appear in uinfo for some concept. The rows of the table contain first the uid of a concept and then the UM attribute values for that concept. (There may be holes in that table because the uinfo for concepts may contain different attributes for different concepts.

AHAM does not require the presence of specific attributes in the user model. However, many AHS store at least the following two attributes, and maybe also the third:

- The *knowledge* (or *knowledge_value*) indicates how much the user knows about the concept. The concept-value pairs together form an *overlay model*, which represents the "knowledge" of the user. Some AHS use a "Boolean user model" [De Bra and Calvi, 1997, 1998c], meaning that for each concept the user either knows or does not know the concept (according to the system). Other AHS use either a small set of knowledge values [Brusilovsky et al., 1996a,b], such as "not known", "learned", "well learned" and "well known", or even a large set, such as a percentage or a (real) value between 0 and 1 [Pilar da Silva, 1998].

- The *read* attribute indicates whether the user has read something (a fragment, a page or a set of pages) about the concept. In Web-based systems the *read* attribute is used to generate a different presentation for anchors of links to pages that have been read than for links to unread pages. (By default, in most browsers the difference is a purple vs. blue color for the anchor text or image border.) The *read* attribute may have Boolean values in some AHS while it may be a list of access times in other AHS.

- A less common attribute is *ready* (or *ready_to_read*), sometimes also referred to as *desirable* or *recommended*. It indicates whether the user is ready to read about this concept. (In a learning application this would mean that sufficient prerequisite knowledge has been acquired.) If this attribute exists in the user model it can be used to ensure that recommended pages (or fragments) always remain accessible. If the AHS calculates the desirability on the fly (as many do) pages may lose their "ready-to-read" status and sometimes even become inaccessible. (In a learning application this is probably not desired system behavior, but in an adventure game for instance this could be common.)

Table 3.1 shows an example of a user model (instance). In the example, HTML and HTTP are subconcepts of the composite WWW. By learning everything about HTML (but still nothing about HTTP) the composite WWW already becomes "learned". It is thus possible to already have learned something about a concept while "read" is still **false**. It is even possible that there is no page for the concept WWW, and that it thus can only be learned by reading about subconcepts. In that case "read" is "undefined".

uid (name)	**knowledge**	**read**	**ready**
intro	well learned	true	true
Xanadu	learned	true	true
KMS	not known	false	true
HTML	well known	true	true
HTTP	not known	false	true
WWW	learned	–	–

Table 3.1: Example user model instance

The "table" in Table 3.1 is only a conceptual representation. Actual implementations of AHS may implement this structure in a different way. The AHA! system [De Bra and Calvi, 1997, 1998c] for instance uses a logfile (separate for every user) in which the time is logged at which a user requests a page, and also the time when a user leaves the page. Also, an AHS may implement just one user model table for all users together, by adding a *uid* attribute.

While most AHS to date provide a fixed set of attributes, future AHS allow authors to "invent" new attributes. For the AHAM model this makes no difference. From now on we shall use the notation *C.attr* as a convenient way to denote the value for the attribute *attr* of the concept with uid *C* for the "current" user. When *Xanadu.read* is **true**, it means the user has read the page about concept Xanadu. (We assume the AHS distinguishes users in a way that does not require us to include the user id in our notation.)

3.3.2 Events influencing the UM

The main source of input for an AHS is the browsing behavior of the user. However, as we saw previously, the system may wish to store aspects of the user or the user's environment. While some of this information may also be deduced from the browsing behavior, other parts of information must be gathered through a forms-based interface. Likewise, reliable information about the user's knowledge cannot be obtained from just observing the browsing. In AHAM we assume that the inputs for updating the UM are "events". The model does not constrain the number of types of events to take into account. Four such event types that we frequently find in AHS are following:

1. The user follows a link to a (different) page.
2. The user performs a test (typically in learning applications).
3. Information (about the user) is imported from an external system, possibly also an AHS.
4. A user preference is (explicitly) set or declared by the user (through a form).

Besides observing the browsing behavior, the application can change the user features based on information that is explicitly imported from the environment. (A server may for instance receive information about the browser's window size in a request, or may measure data transfer rates.)

These four different kinds of changes, or events, lead to four kinds of "rules" for updating the user model. Every system includes rules for events of type 1. The system can be made more author centered by including rules for events of types 2 and 3, while the application can become more user centered by including rules for events of type 4. It is also possible to choose a combination that suits the application. In this dissertation we focus on rules for event type 1, meaning that we focus on the access to pages.

3.4 The Adaptation Model (AM)

In AHAM the basis for adaptive functionality can be found in the *adaptation model* (AM). The AM defines how the user model is updated and adaptation is done by using information from the *domain model* and the *user model* and the interaction of the user. These tasks are performed by an adaptation engine that we describe in the next chapter. There are different ways to express the functionality of the adaptation model. We choose the mechanism of a language of rules to express the adaptation strategies at an abstract level. In Chapter 4 we describe the *interpreter* for these rules.

3.4.1 The definition of adaptation rules

Adaptation can be specified in a structured way or in an ad-hoc way. With structured, or *generic*, we mean that we associate a certain adaptation behavior with concept relationships (or other structures) that exist in the domain model. In InterBook [Brusilovsky et al., 1996b] for instance the annotation of link anchors (with green, white or red balls) depends on the user's knowledge and the defined prerequisite relationships. (See Chapter 5 for details.) In AHAM we can also allow authors to write adaptation rules that are *specific* for some given concepts (irrespective of the existence of relationships between these concepts). In the AHA! system [De Bra and Calvi, 1997, 1998c], for instance, all adaptation rules are specific (i. e. tied to concrete concepts and/or pages). If we allow generic and specific adaptation rules it becomes possible that a generic rule and a specific rule contradict each other. In this case, it is normal to let the specific rule take precedence over the generic one.

Rules in an AHS can perform one of the following three functions: initialization of the user model (IU), updating the user model (UU), and generation of the adaptation (GA). In the definition below we split the user model updates in two parts: one (UU-pre) performed before generating the adaptation and one (UU-post) performed after generating the adaptation. The reason for having different execution phases to update the user model in the definition is that one may wish to first do some adaptation based on the "current" state of the user model (after executing the rules in UU-pre) and later update the user model to a new state after generating the presentation of the page(s) that result from following a link. (This second set of updates are defined by the rules in UU-post.)

Definition 3.9 *A* generic adaptation rule *is a tuple* <R, PH, PR> *where* R *is a rule (triggered by some event),* PH *is the* execution phase *of the rule and* PR *is a Boolean* propagate field*, which indicates whether this rule may trigger other rules. The rule R changes user-model attributes or adaptation attributes of concept variables. The* phase PH *of a rule can have the value* IU*,* UU-pre*,* GA *or* UU-post*. The phase* IU *is executed during initialization of the user model; the phase* UU-pre *is executed before the generation of the page(s); the phase* GA *performs adaptation and generates the page(s);* UU-post *is executed after the page generation.*

Definition 3.10 *A* specific adaptation rule *is a tuple* <R, SC, PH, PR> *where* R *is a rule (triggered by some event),* SC *is a set of concept components used in the rule,* PH *is the* phase *and* PR *is the Boolean "propagate" field. The rule changes user-model attributes or adaptation attributes of the specific concepts of* SC*.*

An AHS may have predefined or built-in generic adaptation rules. If these rules suffice there is no need for a language in which authors can write new rules. InterBook [Brusilovsky et al., 1996b] for instance has only built-in (generic) rules. In case a system offers built-in rules and author-defined rules it is clear that author-defined rules must take precedence over predefined rules.

Because we create the UM as an overlay model of the DM each UM concept should correspond to a concept in the DM. User model attributes are different from domain model attributes and different from adaptation attributes. When we choose the attribute names in such a way that there is no overlap between UM attributes, DM attributes and adaptation attributes we can simplify the syntax of the rule language by not including a mechanism to associate an attribute with the UM, DM or presentation specifications. (In example rules we shall often use *pres* as the name for an adaptation attribute.) Although the definition does not place any constraints on the use of attributes in rules in any of the phases, we will normally update UM attributes in the UU-pre and UU-post phases and adaptation attributes in the GA phase.

Definition 3.11 *The* adaptation model (AM) *of an AHS is the set of generic and specific adaptation rules.*

The separation of the AM from the UM and DM makes it is easy to focus on each design and authoring issue separately when designing an AHS. The creation of generic adaptation rules can be done by a system designer, while the creation of the concept relationships used by these rules can be done by an author who creates the content (information) of the application. This separation is very clear in InterBook for instance. In the AHA! system no such separation exists, meaning that the author of the information space must also be capable of writing adaptation rules.

3.4.2 Adaptation rule examples

To give readers of this dissertation a clearer idea what adaptation rules are, and how they work, we give a few examples of typical generic and specific rules, using an intuitive syntax. (In Chapter 4 we formally define a different syntax.) For the application of these rules we assume that the AHS is displaying one or more pages and that the user "clicks" on a link anchor. This activates the *FollowLink* operation from the Dexter model's Run-time Layer, which in turn results in a call to a *resolver function* for the specifier corresponding to the link anchor (on the current page). In AHAM the resolver *must* translate the given specifier to the uid of a composite concept component that *corresponds to a page*, or to a set of such uid's. It may require several recursive calls to the resolver to go from a high-level abstract concept down to the page level. Which pages are selected depends on the domain model (that defines the hierarchy and structure of composites as well as concept relationships that may indicate a preferred reading order) and on the user model (that states what the user's knowledge about different concepts is). For each selected page an *accessor function* is called, according to the Dexter model, which returns the (page) concept component that corresponds to the resolved uid. The adaptation rules in AHAM are "triggered" through this *accessor function*.

Example 3.1 *The following generic rule expresses that when a page is accessed the* read *user-model attribute for the corresponding concept is set to* ***true*** *in the UU-post phase:*

$$< access(C) \Rightarrow C.read := \textbf{\textit{true}},\ UU\text{-}post,\ \textbf{\textit{true}} >$$

The rule also says that it will trigger other rules that have read *on their left-hand side.*

An interesting point here is the choice of the execution phase for this rule. It is likely that the presentation of links to a page depends on the "read" status of that page. If the "read" attribute is updated in the UU-pre phase and the page contains a link to itself then that link will be presented as a link to an already visited page. If the rule is executed in the UU-post phase then the link will be presented as a link to a page that was not yet read. It is thus possible to express both system behaviors in AHAM, but of course any given system will exhibit only one behavior, which in this case depends on the execution phase of the rule.

Example 3.2 *The following rule expresses that when a page is "ready-to-read" and is accessed, the knowledge value of the corresponding concept becomes "well learned" in the UU-pre phase. This is similar to the behavior of InterBook [Brusilovsky et al., 1996b].*

$$< \textit{access(C)} \ \textbf{\textit{and}} \ \textit{C.ready} = \textbf{\textit{true}}$$
$$\Rightarrow \textit{C.knowledge} := \text{"well learned"}, \ \textit{UU-pre}, \ \textbf{\textit{true}} >$$

In this example the phase was chosen to be UU-pre because this is the behavior of InterBook and many other AHS. This choice is counterintuitive at first glance but illustrates a shortcoming of Web-based AHS, or any AHS that does not change the presentation of a page while the user is reading it: the presentation of a page is adapted to the knowledge the user will have *after* reading the page. This behavior is motivated by the need to present as "relevant" the anchors for links to pages that only become relevant after reading the page (i. e. for which this page was the last missing piece of prerequisite knowledge). By having two phases in the AHAM model it becomes possible to describe the behavior of future AHS that will register the new knowledge *after* the page has been generated and presented, and that will be able to then change the presentation accordingly.

Example 3.3 *The following rule illustrates how a prerequisite relationship works: this generic rule states that a prerequisite relationship between two concepts is satisfied when the prerequisite concept is at least* "well learned"*. For simplicity we assume that when CR is a concept relationship, the uid of the i-th specifier is CR.ss*[*i*]*.uid, the knowledge value of the concept in the user model is CR.ss*[*i*]*.uid.knowledge, the relationship type is CR.cinfo.type, the direction of the i-th specifier is represented CR.cinfo.dir*[*i*]*, etc. We also assume that knowledge values can be compared using* $>$ *and* $\geq$*, where higher values mean more knowledge.*

$$< \textit{CR.cinfo.type} = \text{"Prerequisite"} \ \textbf{\textit{and}} \ \textit{CR.cinfo.dir}[1] = \text{"FROM"}$$
$$\textbf{\textit{and}} \ \textit{CR.cinfo.dir}[2] = \text{"TO"} \ \textbf{\textit{and}} \ \textit{CR.ss.length} = 2 \ \textbf{\textit{and}}$$
$$\textit{CR.ss}[1]\textit{.uid.knowledge} \geq \text{"well learned"}$$
$$\Rightarrow \textit{CR.ss}[2]\textit{.uid.ready} := \textbf{\textit{true}}, \ \textit{UU-pre}, \ \textbf{\textit{true}} >$$

Note that this rule only "works" if it is triggered. Example 3.2 shows that from an "access" event a change to the knowledge is generated, which propagates as a new event. So if the knowledge value of CR.ss[1] *has changed because of an "access" event, that change triggers the rule given in this example.*

Note that a completely "correct" syntax would be more complicated because of the complex nature of relationships and specifiers in the Dexter model (and thus also in AHAM) and because we would need to discriminate between attributes of concepts in the domain model and in the user model.

We now turn to examples that deal with the presentation aspect of an AHS. In the Dexter model, and also in AHAM, the link between the Storage Layer and the Run-time Layer is formed by *presentation specifications*, which are not described in detail. We give a few examples of how adaptation rules are used to generate an *adaptation attribute* of the presentation specifications.

Example 3.4 *For atomic concepts (i. e. fragments) we assume there is an adaptation attribute* pres *(of the presentation specification) that is a two-valued (almost Boolean) field, which is either* show *or* hide*. When a page is being accessed, the following rules set the visibility for fragments that belong to a "page" concept, depending on their "ready-to-read" state.*

$$< \mathit{access(C)}\ \textbf{\textit{and}}\ F \in \mathit{C.children}\ \textbf{\textit{and}}\ \mathit{F.ready} = \textbf{\textit{true}} \Rightarrow \mathit{F.pres} := \text{“show”},\ \mathit{GA},\ \textbf{\textit{false}} >$$

$$< \mathit{access(C)}\ \textbf{\textit{and}}\ F \in \mathit{C.children}\ \textbf{\textit{and}}\ \mathit{F.ready} = \textbf{\textit{false}} \Rightarrow \mathit{F.pres} := \text{“hide”},\ \mathit{GA},\ \textbf{\textit{false}} >$$

Here we again simplified things, by assuming that we can treat C.children as if they were a set, whereas it really is a sequence.
This adaptation attribute is used by the adaptation engine of the AHS to include only those fragments in a page that are "ready-to-read".

Note that the update to the adaptation attribute is not propagated: the presentation specification is passed on to the implementation-specific part of the AHS that is not part of the AHAM model. (Thus, an attribute of the presentation specification is *set* by an adaptation rule, but not *read* by other rules.) However, there may be other rules that are triggered by the "access" event, for instance a rule executed in the UU-post phase that will set the *read* attribute of the fragments to *true*.

Example 3.5 *The following rules set the adaptation attribute* pres *of a specifier that denotes a link (source) anchor depending on whether the destination of the link is considered relevant ("ready-to-read") and whether the destination has been read before. For simplicity we consider a link with just one source and one destination.*

$$< \mathit{CR.type} = \text{“Link”}\ \textbf{\textit{and}}\ \mathit{CR.cinfo.dir}[1] = \text{“FROM”}\ \textbf{\textit{and}}\ \mathit{CR.cinfo.dir}[2] = \text{“TO”}\ \textbf{\textit{and}}\ \mathit{CR.ss}[2].\mathit{uid.ready} = \textbf{\textit{true}}\ \textbf{\textit{and}}\ \mathit{CR.ss}[2].\mathit{uid.read} = \textbf{\textit{false}} \Rightarrow \mathit{CR.ss}[1].\mathit{pres} := \text{“good”},\ \mathit{GA},\ \textbf{\textit{false}} >$$

$$< \mathit{CR.type} = \text{“Link”}\ \textbf{\textit{and}}\ \mathit{CR.cinfo.dir}[1] = \text{“FROM”}\ \textbf{\textit{and}}\ \mathit{CR.cinfo.dir}[2] = \text{“TO”}\ \textbf{\textit{and}}\ \mathit{CR.ss}[2].\mathit{uid.ready} = \textbf{\textit{true}}\ \textbf{\textit{and}}\ \mathit{CR.ss}[2].\mathit{uid.read} = \textbf{\textit{true}} \Rightarrow \mathit{CR.ss}[1].\mathit{pres} := \text{“neutral”},\ \mathit{GA},\ \textbf{\textit{false}} >$$

$$< \mathit{CR.type} = \text{“Link”}\ \textbf{\textit{and}}\ \mathit{CR.cinfo.dir}[1] = \text{“FROM”}\ \textbf{\textit{and}}\ \mathit{CR.cinfo.dir}[2] = \text{“TO”}\ \textbf{\textit{and}}\ \mathit{CR.ss}[2].\mathit{uid.ready} = \textbf{\textit{false}} \Rightarrow \mathit{CR.ss}[1].\mathit{pres} := \text{“bad”},\ \mathit{GA},\ \textbf{\textit{false}} >$$

These rules say that links to previously unread but "ready-to-read" pages are "good", links to previously read and "ready-to-read" pages are "neutral" and links to pages that are not "ready-to-read" are "bad". In the AHA! system [De Bra and Calvi, 1997, 1998c] this results in the link anchors being colored blue, purple or black respectively. In ELM-ART [Brusilovsky et al., 1996a] and InterBook [Brusilovsky et al., 1996b] the links would be annotated with a green, white or red ball.

3.5 Communication Between AHS

The Dexter model assumes that all "history information" is limited to a single browsing or authoring session. It even states that when closing a session *by default, pending changes to instantiations are not saved* [Halasz and Schwartz, 1994]. In AHAM we explicitly model a permanent user model, thus taking into account that a user's interaction with a hypermedia information source may span several sessions. It is possible to extend the user model to include a representation of the evolution of the user's state of mind throughout his or her interaction with several different adaptive hypermedia applications. The exchange of user models is one of the areas for which the IEEE Learning Technology Standards Committee (LTSC) (P1484) is currently developing a standard. (See http://grouper.ieee.org/groups/ltsc/.)

Modeling the exchange of user model information in AHAM is as simple as adding two events, in addition to the "access" event:

- An AHS may offer an externally accessible function:

  GetValue(user, auth, cuid, attr)

  where user identifies a user, auth is a system-dependent authorization, cuid is a unique concept identifier, and attr is an attribute of the user model. The function returns a value (of the appropriate type) for that attribute. The caller of the function needs to know which data type to expect.

  Here it is convenient that the Dexter model, and thus also AHAM, requires that the unique identifiers for all components are globally unique, not just within a hypermedia application but unique within the entire universe of discourse.

- An AHS may offer an externally accessible procedure:

  SetValue(user, auth, cuid, attr, value)

  where user identifies a user, auth is a system-dependent authorization, cuid is a unique concept identifier, attr is an attribute of the user model and value is the new value for this attribute. This procedure (or void function) updates the user model.

Authorization is needed for obvious reasons: not every external application can be allowed to read and/or update arbitrary user models.

It may seem that it is very inconvenient to have these functions take or return values of specific data types, which may be different in every AHS. It is possible to translate many (but not nearly all) data types to a "standard" one, such as real (floating point) numbers between 0 and 1. However, the biggest problem in the communication between systems is not the technical data conversion but the semantic conversion. In order for an application to use the "knowledge value" for a concept C, which is imported from a different AHS,

(the author of) that application must know what the concept C means in terms of its own concepts, and it must know what the "knowledge value" means. If a system that uses values between 0 and 1 wishes to retrieve a knowledge value from another system that returns "well learned", the system needs to be able to interpret which of its own values has the same meaning as "well learned" in the other system. Even when two systems use knowledge values between 0 and 1 the identity may not be the most appropriate conversion of knowledge values.

An immediate application of the functions GetValue and SetValue is in the communication with semi-external applications such as an evaluation tool that uses multiple-choice tests, or an initial questionnaire that is used to initialize a user model and to set preferences.

3.6 Summary of AHAM

AHAM provides a general framework to describe the adaptation functionality of AHS at an abstract level. We built AHAM as an extension of the Dexter model for hypermedia applications because we want to focus on solving adaptation problems and use the other functions from the Dexter model as much as possible. AHAM provides the answer to research question 3: we can describe the functionality of AHS at an abstract level, and at the same time describe it as an extension to hypermedia systems in general.

AHAM strongly advocates a clear separation of the DM, UM, and AM in future AHS design. AHAM has following properties:

- It decomposes the design problem in functionally different parts so that the role of each part and the relationships between the parts are made explicit. This also enables people with different skills to develop an adaptive hypermedia application together.
- It makes adaptation in AHS easy to understand, independent of the information content and structure of a concrete application.
- It guides the development of future flexible AHS in the sense that it becomes possible to write different adaptation rules for the same information domain. It also enables the exchange of information about users and users' knowledge captured in the user model by different adaptive hypermedia applications.
- It supports advanced adaptation by using fine-grained concepts (fragments), high-level abstract concepts, and concept relationships in the DM.
- It allows the contol over the adaptation to be more user-centered or more system-centered by choosing different types of user attributes in the UM.
- It provides a framework to compare the adaptation functionality of different AHS.

Chapter 4

AE: Adaptation Engine

AHAM, as described in the previous chapter, describes the *structure* of an adaptive hypermedia application. It describes the *behavior* of the AHS at an abstract level through adaptation rules. In order to know completely how the AHS behaves we also need to study the *interpreter* for these rules. This part of the AHS is called the *adaptation engine* (AE). In this chapter we study how an AE works and what its properties are.

This chapter consists of two parts. The first part describes the design considerations for a general-purpose AE. Section 4.1 describes the tasks performed by a (general purpose) AE, at a high level as well as at the level of individual rule executions. It also defines some desirable properties of an AE. Section 4.2 describes requirements for a general purpose AE. It stresses the goal to make authoring easier in future AHS. We only consider an AE that uses a rule-based adaptation model (AM) as described in the previous chapter. This does not imply that we can only describe the behavior of AHS that use such a rule-based adaptation engine. The rule-language is only a way to describe the behavior of the system, not a requirement for its implementation. In Chapters 5 and 6 we describe InterBook and AHA!. InterBook does not use a rule-based adaptive engine as described in this chapter. AHA! uses a limited kind of rule engine. Nonetheless we can describe the behavior of InterBook and AHA! using the AHAM model.

It is always difficult to predict the "final results" in rule-based systems, because the rule execution may run into an infinite loop (called *termination problem*), or generate unpredictable results (called *confluence problem*). Problems with termination have been encountered in the AHA! 1.0 system. Recovering from an infinite loop meant "killing" and restarting the server each time. An ad-hoc solution was implemented but this could not guarantee confluence. In the newer AHA! 2.0 implementation, termination remains a problem and may be "solved" by introducing an arbitrary limit on the number of rule instances that can be executed, until an accurate prediction of infinite loops at authoring time becomes possible.

Termination and confluence are thus two important properties of (trigger-based) rule systems. In the second part of this chapter we first review how the issues of termination and confluence are dealt with in the research area of active databases. Section 4.3 thereby focuses on *static analysis* of trigger-based rules in active databases. This means that the

analysis is done at authoring time, when the adaptive application is being defined, and not at run-time, when a user is browsing through the adaptive information space. The most common type of rule is called *Event Condition Action rules* or ECA rules. We observe that the sufficient conditions for termination and confluence we find in literature in general are too strict to be useful. The active database literature also discusses rules without triggering events. They are called *Condition Action rules* or CA rules. The static analysis of CA rules leads to the identification of sufficient conditions for termination and confluence that are less conservative (i. e. that are closer to being also necessary conditions). We will use CA rules for AHS for two reasons: they are easier for authors (because only two parts have to be specified instead of three) and an authoring tool can more accurately warn authors when there are potential problems with the defined set of rules. Section 4.4 defines a syntax of rules for AHS, called AHAM-CA rules. Section 4.5 describes how rules work together to update the UM and to generate the adaptation. Section 4.6 proposes a way to use the active database research results to perform static analysis for AHS. By using the specific knowledge about AHS (which are a "special kind of" active database system) we obtain an analysis method that is less conservative than the static analysis for active database [Baralis and Widom, 2000]. This research was reported on in [Wu et al., 2001]. Section 4.7 defines a few possible ways (or "tricks") to enforce the termination and confluence during the rule execution for the case that termination and confluence are not guaranteed by the static analysis. To lower the complexity of performing the static analysis, or to eliminate the analysis altogether, Section 4.8 proposes some constraints on AHAM-CA rules and proves that a certain set of constraints together guarantees termination and confluence. This result is useful because algorithms for checking these constraints are more efficient (have lower time complexity) than those for checking the sufficient conditions in general.

4.1 Introduction

An adaptive hypermedia application consists of not only the structures defined by AHAM but also "contains" an adaptation engine (AE). AHAM describes the adaptation functionality at an abstract conceptual level, and the AE describes the adaptation functionality at a (still abstract) implementation level.

Definition 4.1 *The AM specifies the relationships between the UM and the DM, in particular how user access of the DM (via its presentation in hypermedia) changes the values within the UM. The* adaptation engine *AE executes the rules specified within the AM. It is thus an* interpreter *for the adaptation rules in the AM.*

The examples presented in Section 3.4 illustrate how an *adaptation engine* of an AHS can use adaptation rules to generate presentation specifications. In the examples this generation is very simple (setting the visibility of a fragment or the class of a link anchor). In general, however, the adaptation engine may have more difficult tasks, e. g. when the presentation of a page requires fragments to not only be selected but also sorted. Some tasks of the engine are of an administrative kind, for instance registering new users or

handling forms input when users change preferences, knowledge or interest settings or when students complete a multiple-choice test in an adaptive course. We look at only one type of task of the adaptation engine: dealing with a user's (access) request for a specifier S. (Typically such a request is the result of following a hyperlink.) The AE then, in principle, performs the following steps:

1. The AE initializes the user model (UM) by assigning default values to all attributes of all concepts that are used in the adaptation rules but that are not stored permanently in the UM. Having default values reduces the number of rules that must be defined. For instance, if *ready* is used in some rules, we can set the *ready* attribute for all concepts to **true** by default, and then only have rules to set it to **false** when needed. Defaults also reduce the number of rule executions because it does not require a rule to set a default value.

2. The AE retrieves the user's stored user model (UM). The run-time layer ensures, by using "sessions", that the identity of the user is known. All stored attributes of all concepts are retrieved. This, together with the previous step, ensures that the UM is ready for use (with everything initialized). An AE may use a cache to avoid this expensive retrieval operation, but this does not matter for our conceptual view of the AE's actions.

3. The AE resolves the specifier S to a page concept C by applying adaptation rules that are aimed at determining a "desired" page(s) for specifier S, depending on the UM. These rules may use prerequisite, inhibitor and other similar concept relationships.

4. The AE starts executing the rules associated with *access*(C) as the trigger. (We explain *access*(C) later.) All triggered rules from the UU-pre phase are executed first, to perform UM updates before generating the presentation specifications.

5. The AE then does whatever is necessary to actually present the page(s). This is the "GA" phase. A page (or set of pages) is built, using the presentation specifications. Fragments are selected and possibly sorted, for each page to be presented. This action crosses the boundary between the "storage layer / presentation specifications" and the run-time layer. In a Web-based system this would also be the boundary between server and browser. The adaptation engine generates an HTML page and sends it to the browser.

6. Next all triggered rules from the UU-post phase are executed. These rules cannot change the presentation specification anymore in a meaningful way, because the presentation is already generated and handed over to the run-time layer. However, it may be useful to change certain values in the UM to influence the next transaction, e. g. to reset some default values for the next transaction or to ensure that next time the same concept is accessed its presentation is different.

7. The updated values for persistent attributes of UM concepts are committed (or saved in that user model).

Some adaptation engines only have a UU-pre phase. The UU-post phase is an additional possibility to update the user model after the page generation. In the systems we describe later (InterBook [Brusilovsky et al., 1996b] and AHA! [De Bra and Calvi, 1997, 1998c]) we will show how the UU-post phase is used to "prepare" the adaptation engine for the next event. Many current AHS are Web-based. When the user "clicks" on a link a CGI-script or a servlet generates and adapts the requested page. Any processing that can be done before the user's request arrives helps speed up the processing of events generated by the user. The UU-post phase lets the engine work while the user is reading a page.

Many AHS show not only the content of "accessed" concepts, but also add some information related to the concept, e. g. navigation support. This is convenient for modeling AHS that divide the display (or browser window) into different parts, each showing different aspects of the adaptive hypermedia application. A "kiosk" system and a course text realized in AHA! [De Bra and Calvi, 1998c] for instance use an adaptive *table of contents* frame and an *information* frame. (See http://wwwis.win.tue.nl/IShype/ and http://wwwis.win.tue.nl/2R350/.) Applications of InterBook [Brusilovsky et al., 1996b] also include frames that show outcome concepts or that are prerequisites for the "current" page. Each time a link is followed all frames need to be updated.

Now that we have introduced all the components that make up an adaptive hypermedia application we can give the following definition:

Definition 4.2 *An adaptive hypermedia application is a 4-tuple* $< DM, UM, AM, AE >$ *where DM is a domain model, UM is a user model, AM is an adaptation model, and AE is an adaptation engine.*

4.1.1 System transactions

In an adaptive hypermedia application the DM and AM are static, but the UM is updated by the AE according to the rules defined in the AM. This section describes how an AHS works using "triggered rules". However, this does not imply that only rule and trigger based AHS's can be described in AHAM. Triggers and rules are just our means for describing the system behavior. When an external event occurs it triggers some rule(s) in the system. The execution of these rules changes the "state of the UM". Such state changes can be triggers for other rules. We first described a transaction-based view on the adaptation engine in [Wu et al., 2000].

An *event* in the system means that something outside the system triggers the system to change its state. The user or external programs can only observe the following:

$$UM_s \xrightarrow{event} UM_f$$

We call this a *system transaction*. UM_s and UM_f are two states of the system. UM_s is the *start-state* and UM_f is the *final-state*. Usually these states will differ in at least one attribute value so that one can tell from looking at the UM that the transaction took place. A transaction is the smallest observable action performed by the system. The term corresponds to the term transaction in databases. "Inside" the AHS, but invisible to

any external observer (user or program) the transaction may be realized by sequentially executing a number of rules:

$$UM_s \xrightarrow{R_1} UM_2 \xrightarrow{R_2} \cdots\ UM_i \xrightarrow{R_i} \cdots\ \xrightarrow{R_m} UM_f$$

Here R_i is a specific rule or an instance of a generic rule (meaning that the concept variables in the generic rule have been instantiated by concrete concepts from the DM), i is in $[1 \ldots m]$, m must be finite. Thus, internally, the system applies a finite sequence of rule instances to arrive at UM_f. Each step in this transaction is called a *transition*. When an event occurs, it triggers the rules that deal with that event. These rules will change some values in the UM, and these changes will trigger (or "propagate to") other rules. The order in which rules are applied is called the *execution order*.

Each rule has an (optional) condition (which is a Boolean expression). When a rule's condition changes from **false** to **true**, the rule becomes an *active rule*. The system only executes active rules in the transaction. After execution of an active rule, that rule becomes *inactive* (unless it performs an update that explicitly makes it active again). Different execution models for rules have been proposed and studied [Widom and Ceri, 1996]. Two extremes are when either all the active rules are executed simultaneously (in parallel), or when all the active rules are collected in a queue (or stack or set) and executed sequentially. In our study of properties of the AE we only consider the sequential execution model (and we also talk about having a set or queue of active rules). Parallel execution would require a conflict resolution mechanism to avoid executing rules in parallel that try to perform conflicting updates. The execution of a rule may activate other rules (i. e. make inactive rules become active). A rule may also activate itself. Because in our model rules are taken from the queue of active rules and executed one by one it is possible that active rules become inactive before they are "selected" for execution. Rules may thus miss their "window of opportunity". When a rule is selected for execution the AE first checks whether the rule is still active. If the rule is found to be inactive it is discarded. If there are no more active rules, then the transaction *terminates*.

As well as the termination of a transaction, the predictability of its final state is also an issue. For the same user the system should provide the same result or have the same final state in the same situation. We call this property *confluence* and define it in the next section.

If the execution order of the rules does not affect the final state, we call the transaction an *order independent transaction* (OIT). If the system can generate and perform only order independent transactions it is always confluent if it terminates, because it can provide the same final result for the user in the same situation. This is the ideal situation. In many more complicated trigger-based systems confluence cannot be guaranteed.

If the execution order does affect the final state, we call this transaction an *order dependent transaction* (ODT) . Order-dependence may arise when there is a choice as to which rule to execute first. For instance, an event may activate more than one rule. In the sequential procedure presented above, we select one rule and execute it. This will produce a new (intermediate) state of the UM in which some rules may have become active and

some active rules may have become inactive. When, for instance, two rules become active that try to deactivate each other, the order in which the rules are put in the queue (or in the set-interpretation the order in which the rules are taken from the set) determines which of the two rules will be executed first. The second rule will not be executed because the first rule's execution makes it inactive. The final result may also depend on the order in which subsequent updates of the same concept attribute take place. If two rules try to set the same attribute of the same concept to a different value the rule that is executed last determines the final value. To guarantee that an AE exhibits confluent behavior, we have to resolve the order dependence.

4.1.2 Properties: termination and confluence

Termination and confluence are two important properties to judge the behavior of all kinds of trigger-based rule systems, not just AHS. These two properties have been studied quite thoroughly in the area of active databases. As the state change of the AHS can be seen as the state change of the UM by applying a set of rules, and as the UM can be seen as a (simple) database, we can draw a parallel with some terms and algorithms from active databases [Baralis and Widom, 1994].

Definition 4.3 *A* rule execution state *S is a pair (d, R_A), where d is a database state and $R_A \subseteq R$ is a set of active rules. (R is the complete set of rules.)*

Definition 4.4 *A* rule execution sequence *is a sequence σ consisting of rule execution states linked by (executed) rules. A rule execution sequence is* complete *if the last state is $(d, \emptyset)$, i. e., the last state has no active rules. A rule execution sequence is* valid *if it represents a correct execution sequence: only active rules are executed, and pairs of adjacent states properly represent the effect of executing the corresponding rule; for details see [Aiken et al., 1992; Baralis, 1994].*

Definition 4.5 *A rule set (or set of rules) R* terminates *if the rules cannot activate each other indefinitely, or in other words if every valid rule sequence is finite.*

Definition 4.6 *A rule set R is* confluent *if, for every initial rule execution state S, every valid and complete rule execution sequence beginning with S has the same final state.*

There are two ways to achieve termination and confluence properties. One is called *detection*, another is called *enforcement* [van de Voort, 1994].

Literature on detection of termination can be divided into two categories, one concerned with decidability and the other with finding sufficient conditions. Whenever a property is decidable then there exists a decision algorithm for its detection. Obviously, decidability depends on the trigger definition language for which it is studied. Unfortunately termination and confluence are undecidable in general for any "interesting" language [Aiken et al., 1992; Bailey, 1997]. Therefore, instead of trying to determine exactly whether a set of rules defined by an author (or system designer) *will* or *will not* cause termination or confluence

problems we try to find sufficient (but not necessary) conditions to guarantee that a set of rules *cannot* cause these problems. Sufficient conditions for termination and confluence in the context of active database systems are given in [Aiken et al., 1995; Baralis and Widom, 2000]. Rule execution in AHS is similar to the behavior of active databases, so we can use static analysis results from the active database field and extend them for the specific situation of AHS.

Termination and confluence are also studied in the context of term rewriting systems. In [Karadimce and Urban, 1994] triggers are mapped to term rewriting systems in order to be able to use termination and confluence results developed for term rewriting systems in the analysis of triggers. The results for termination are based on the identification of a decreasing function on the rewrite rules, and the results for confluence are based on the analysis of critical overlaps between triggers. A term-rewriting system is an automaton. The use of automata (where nodes are represented by states and links by transitions) is not straightforward in the case of conditional links, because these lead to an explosion of the required number of states. Therefore we do not use automata and term-rewriting system to analyze triggers in AHS .

Once it is detected that the termination and confluence properties do not hold, enforcement of these properties during the execution becomes a relevant issue. Enforcement can be static or dynamic. Static enforcement means that properties are enforced through restrictions in the way rules can be specified or in the way rules can trigger each other. Dynamic enforcement means that properties are enforced by observing the trigger execution process and intervening when the process goes wrong.

Static enforcement of termination can be achieved by defining a restrictive trigger definition format that guarantees termination. At run time, termination can be enforced by limiting the number of allowed trigger executions, as in [Paredaens et al., 1993]. Static enforcement of confluence can be based on assigning priorities to triggers. If a total order is defined, trigger execution is guaranteed to be confluent.

Static enforcement is not an elegant way to enforce termination and confluence in general, because its result may be unpredictable or hard to understand. If the assigned total order of priorities is not known (and understood) by the author the system may be confluent but may not produce the results the author expects.

Dynamic enforcement used in [Zhou and Hsu, 1990] aborts the trigger execution when the system detects triggers that cause conflicting updates. The result is equal to the no-operation by aborting the trigger execution. As in databases, no externally observable action takes place when the execution is aborted.

In this dissertation we concentrate on static analysis of rules. We give only a simple strategy for dynamic enforcement.

4.2 Requirements for the AE

In AHAM we have decided to describe the adaptation using adaptation rules that are executed by the adaptation engine (AE). In the previous section we already indicated that

powerful rule systems may exhibit termination and confluence problems. Before turning to the study of these problems we first describe the kind of adaptation rules and adaptation engine we need in AHS.

Our design goals for future AHS are to create systems that allow the definition of interesting adaptation strategies, are easy to use by authors, guarantee termination and confluence properties (or detect termination and confluence problems) and are efficient (so end-users do not have to wait for adapted pages to be generated). At the same time, AHAM should be able to express the adaptation as it exists in current AHS. (This should not be difficult because our goals for future AHS are much more ambitious than the functionality offered by the current generation of AHS.) We briefly elaborate on these goals as follows:

- *Adaptation strategies*: An AHS should be able to perform adaptation based on the user's knowledge, interest, goals, learning style, background, environment, etc. In AHAM we can represent each aspect for each concept of the DM by adding attributes to the concepts in the UM. Adaptation rules can then use and update these UM attributes. Adaptation rules can combine attribute values in any desired way to generate presentation specifications. In order to achieve all this functionality it is sufficient to have an adaptation rule language that allows events to trigger UM updates to any attribute of any concept, and that allows such updates to trigger other update rules. When we look back at the structure of the DM (Section 3.2) we see that there is a concept hierarchy. In an educational application (like in [Brusilovsky et al., 1996b] or [Pilar da Silva, 1998]) we need to be able to model that knowledge about a page contributes to knowledge about a section in the course, and that knowledge about a section contributes to knowledge about a chapter, and to the course as a whole. While we could associate rules with every page access to define the knowledge contribution to the page, section, chapter and course it is easier to do this in steps and to let the knowledge updates propagate from page to section to chapter to course. The "propagate" field in an adaptation rule (see Section 3.4) was introduced with this propagation in mind.

- *Ease of authoring*: The adaptation that an AHS can perform may be partly built-in, defined by a system designer, and partly defined by an author. In order to make authoring as easy as possible all application independent adaptation rules should be defined by a system designer, leaving only some application dependent rules to the author. As an example, the AHS should provide built-in adaptation rules to deal with knowledge propagation though the concept hierarchy, and to generate link adaptation based on prerequisite relationships. The author can then concentrate on defining the concept hierarchy and the prerequisite relationships without worrying about the adaptation that results from these structures. System-defined rules will almost always be *generic* adaptation rules. Authors may wish to write some *specific* adaptation rules to create some uncommon adaptation linked to a few specific concepts, thereby possibly overriding the adaptation defined by some generic rules. This division of authoring tasks illustrates the need for generic and specific adaptation

rules. Propagation of updates is not only needed for knowledge transfer through the concept hierarchy, but also to create transitive behavior of prerequisite relationships.

- *Performance*: Performance is a frequently neglected aspect of adaptive hypermedia applications. When the execution of the adaptation rules takes a long time, there will be a very noticeable delay between the "click" on a link anchor and the appearance of the link destination page. Indeed, all the rules in phases IU, UU-pre and GA (see Definition 3.9) are executed before the page is sent to the user's browser. The easiest way to guarantee fast response times is to disable (or disallow) rule propagation. However, propagation is needed according to the previous two design goals. As a result we need to deal with the problems of termination and confluence. We also need to ensure that, whichever measures we take to guarantee termination and confluence, rule execution never takes a long time while the user is browsing. We consider the following possibilities:

  - *Static Analysis*: When the author creates the adaptive application an authoring tool can analyze the defined DM, UM and AM to decide whether situations (combinations of events and UM instances) are possible that cause the AE to run into an infinite loop or that generate a result that depends on the rule execution order. Section 4.3 describes how to detect that (conservatively chosen) sufficient conditions for guaranteeing termination and confluence of active database rules are satisfied. Section 4.6 does the same for rules in AHS.

  - *Enforcement*: An AHS may allow propagation in a restrictive way such that termination and confluence are guaranteed. The AHA! system for instance guarantees termination by only allowing monotonic updates to be propagated. It also ensures that the rule execution ends after a (relatively) small number of steps, thus guaranteeing fast response times. Another design requirement is that the enforcement should always be performed in such a way that no error messages or no undesired system behavior is ever shown to the end-user of the adaptive application. It should not be possible for end-users to "do anything wrong" for which they are to blame. Section 4.7 deals with the issue of enforcing termination and confluence.

  - *Constraints*: Anomalous rule execution behavior is caused by a combination of structures in the DM and propagation in AM. Infinite loops in rules with propagation can often be avoided by placing constraints on DM structures. In Section 4.8 we investigate how to guarantee termination and confluence through constraints on DM and AM. Since these constraints must be verified during the authoring process it is also important that this verification does not take a very long time. (It does not have to be as efficient as checks during the end-user's browsing activity, though.)

4.3 Review of Static Analysis in Active Databases

This section briefly reviews the static analysis of trigger-based rules in active databases. In the literature we find two types of rule formalisms that seem useful to describe the behavior of an adaptation engine: *Event Condition Action* (ECA) rules and *Condition Action* (CA) rules. ECA and CA rules have been studied extensively in the context of active databases in [Aiken et al., 1995; Baralis and Widom, 2000].

An ECA rule consists of three "independent" parts: an *event*, a *condition* and an *action*. The semantics of an ECA rule is that when the event is triggered, the condition is checked. If the condition holds, the action will be executed. ECA rules are general rules to describe triggers. The conditions may be **true** (or **false**) independent of the cause of the events. In [Aiken et al., 1995] a static analysis method for ECA is described. Static analysis is performed on the definition of the rules and the database scheme. It does not take into account the database instance. The static analysis for ECA rules, described in [Aiken et al., 1995] is "conservative" in the sense that it takes into account the event(s) that trigger a rule, but not the condition when analyzing the possible rule activation. This seems logical because to determine whether the condition of a rule holds one needs to look at the database instance which is not available at design time (when the static analysis is performed). When there are cycles in the EA part of the rules (when only looking at evens and actions) the static analysis will detect a potential termination problem. The fact that cycles may be prevented by the conditions is ignored in this analysis.

A CA rule is much simpler than an ECA rule: it only consists of a condition and a action. Whenever a rule's condition becomes **true**, the rule's action will be executed. Hence, a CA rule is like an ECA rule where the fact that the condition becomes **true** *is* the *event* that triggers the rule. In [Baralis and Widom, 2000] it is stated that many practical applications of ECA rules have the property that a rule's condition becomes **true** exactly when that rule's event occurs. Such rules are called *quasi-CA* and behave exactly like CA rules. The static analysis for CA rules that is known to date [Baralis and Widom, 2000] is less conservative than for ECA rules. However, static analysis of CA rules is still conservative in the sense that it is based only on the definition of the rules and the database scheme. It does not use information on the database instance because that is unknown at design time.

4.3.1 Event Condition Action (ECA) rules

The syntax of Event Condition Action (ECA) rules is partially described by the following grammar:

<rule> ::= on <event>
if <condition>
then <action>

In order to determine the actual behavior of the system, we follow the execution model used in [Aiken et al., 1995].

Compute the set of triggered rules;
Repeat until no more rule is triggered:
Select a triggered rule r;
If r's condition is **true**
then execute r's action (this may trigger more rules)

If more than one rule becomes active at the same time, the system first executes the rule with highest priority. If there is no pre-assigned execution priority the system will choose an active rule in some arbitrary way. This potentially causes confluence problems.

We summarize the results of static analysis of termination and confluence of ECA rules [Ceri and Widom, 1990; Aiken et al., 1995].

Definition 4.7 *The* triggering graph *(TG_R) for a set R of EAC rules is constructed as follows: the nodes in TG_R represent all rules; the edges in TG_R represent the $Triggers$ relationship. Rule r_i triggers rule r_j if the <action> of rule r_i modifies an attribute that appears in the <event> of r_j. (r_j is then an element of $Triggers(r_i)$.)*

Theorem 4.1 *If there are no cycles in TG_R then the rule set R is guaranteed to terminate.*

Note that this theorem is very conservative: it guarantees termination based only on the events and the actions of rules, without considering the actual conditions of the rules. We will see an improvement in the static analysis of termination of CA rules in Section 4.3.2.

Definition 4.8 *Let S_i (for arbitrary i) denote rule execution states (see Definition 4.3). The* execution graph *(EG) for a given starting execution state S is constructed by adding a node S_j and edge (S_i, S_j) whenever from the execution state S_i (already in the graph) the state S_j can be reached by executing one of the active rules in S_i. The execution graph thus represents all possible rule executions (given a set of rules R and an initial execution state).*

Assume that rule execution terminates, i. e. all paths in the execution graph are finite. We wish to determine if every possible execution order results in the same (final) database state. Let S_i, S_j denote execution states, let $S_i \rightarrow S_j$ denote that there is an edge from S_i to S_j in the execution graph and let $S_i \rightarrow\!\!\rightarrow S_j$ mean that there is a path of length 0 or more from S_i to S_j in that graph. From [Aiken et al., 1995] we learn that:

Lemma 4.1 *(Path Confluence) Suppose that for any three states S, S_i and S_j in an arbitrary execution graph EG such that $S \rightarrow\!\!\rightarrow S_i$ and $S \rightarrow\!\!\rightarrow S_j$, there exists a fourth state S' such that $S_i \rightarrow\!\!\rightarrow S'$ and $S_j \rightarrow\!\!\rightarrow S'$. Then EG has at most one final state.*

It is quite difficult in general to determine when the supposition of Lemma 4.1 holds, since it is based on arbitrarily long paths. Aiken [Aiken et al., 1995] gives a somewhat weaker condition that is easier to verify and implies the supposition of Lemma 4.1; it does, however, add the requirement that rule processing is guaranteed to terminate.

Lemma 4.2 *(Edge Confluence) Suppose that for any three states S, S_i and S_j in an arbitrary execution graph EG such that $S \rightarrow S_i$ and $S \rightarrow S_j$, there exists a fourth state S' such that $S_i \rightarrow\!\!\rightarrow S'$ and $S_j \rightarrow\!\!\rightarrow S'$. Then for any three states S, S_i and S_j in EG such that $S \rightarrow\!\!\rightarrow S_i$ and $S \rightarrow\!\!\rightarrow S_j$, there exists a fourth state S' such that $S_i \rightarrow\!\!\rightarrow S'$ and $S_j \rightarrow\!\!\rightarrow S'$. Hence in this case EG has at most one final state.*

It is still difficult to determine when the supposition of Lemma 4.2 holds. So Aiken [Aiken et al., 1995] defines the following *confluence requirement*.

Definition 4.9 *(Confluence Requirement) Consider a set of rules R and any pair of rules $r_i, r_j \in R$. Let $r \in Triggers(r_m)$ represent r_m can trigger r, P be the "priority relation" for R. Let $R_1 \subseteq R$ and $R_2 \subseteq R$ be constructed as follows:*

Initially $R_1 := \{r_i\}$; $R_2 := \{r_j\}$;

Repeat until unchanged:

$$R_1 := R_1 \cup \{r \in R \mid r \in Triggers(r_1) \text{ for some } r_1 \in R_1 \text{ and } r > r_2 \in P \text{ for some } r_2 \in R_2 \text{ and } r \neq r_j\}$$

$$R_2 := R_2 \cup \{r \in R \mid r \in Triggers(r_2) \text{ for some } r_2 \in R_2 \text{ and } r > r_1 \in P \text{ for some } r_1 \in R_1 \text{ and } r \neq r_i\}$$

The confluence requirement says that for every pair of rules $r_i, r_j \in R$ and for any $r_1 \in R_1$ and $r_2 \in R_2$, r_1 and r_2 must commute.

Theorem 4.2 *Suppose the Confluence Requirement holds for R and there are no infinite paths in any execution graph for R. Then every execution graph for R has exactly one final state, i. e. the rules in R are confluent.*

We can use ECA rules to describe the adaptation strategies for AHS, but this leads to a very conservative estimate of what condition must be satisfied to guarantee termination and confluence. Therefore we devote more attention to the study of Condition Action rules.

4.3.2 Condition Action (CA) rules

When the events that trigger ECA rules *are* in fact database updates, the ECA rules become equal to CA rules. Baralis and Widom [Baralis and Widom, 2000] proposed CA rules instead of ECA rules to describe triggers. The important contribution in that paper is that it provides a better static rule analysis for active database systems.

The syntax of a CA rule is defined as $C \rightarrow A$, where:

- C states the rule's condition as an expression in extended relational algebra.
- A states the rule's action as a data modification operation on the same database.

The semantics associated with a CA rule is as follows:

Definition 4.10 *The condition C of a rule $C \rightarrow A$ is* ***true*** *if and only if the result of the expression C applied to the current database instance is not empty.*
At any time we denote by C^{new} the current state of C and by C^{old} the state of C the last time the rule was evaluated during rule processing. Initially $C^{old} = \emptyset$.
A rule is active *if $C^{new} - C^{old} \neq \emptyset$.*

This definition means that a rule r becomes active when (another) rule has produced new tuples that satisfy r's condition. The algorithm for the processing of a set of CA rules is:

Repeat until no rule is active:
Select an active rule $C \rightarrow A$ to execute; (for this rule $C^{new} - C^{old} \neq \emptyset$)
Execute the active rule;
Replace C^{old} by C^{new} (as it was before the execution)

The rule processing is an iterative loop in which, at each iteration, an active rule is executed on the current database state. Rule processing continues until there are no more active rules. When a rule is selected for execution C^{old} is replaced by C^{new}. If the rule does not influence its own condition then whenever the rule is evaluated again $C^{new} - C^{old}$ will be empty, so the rule is not executed again and again in an infinite loop. Of course, if the rule's action creates new tuples that are in C the rule can be selected for execution again.

CA rules may have termination problems just like ECA rules. Also, because there may be many active rules at one time the system may have to choose which rule to execute first and that choice may determine the final result. This indicates a confluence problem. In the next subsections we summarize the study of termination and confluence for CA rules as described in [Baralis and Widom, 2000].

Static analysis of termination

In the above rule execution process CA rules may interact with each other, in the sense that the execution of an active rule may activate or deactivate some other rules.

Whether a rule activates or deactivates another rule depends on the database instance. Static analysis is done at design time when only the database scheme is known. It must take into account all possible database instances. We say that a rule *may* activate (or deactivate) another rule if there is a database instance in which the first rule is active and in which the execution of that rule *does* activate (or deactivate) the other rule.

Definition 4.11 *Consider two rules $r_i : C_i \rightarrow A_i$ and $r_j : C_j \rightarrow A_j$. r_i* may activate *r_j if r_i is active in some database instance and the execution of action A_i can change the database instance from a state in which $C_j^{new} - C_j^{old} = \emptyset$ to a state in which $C_j^{new} - C_j^{old} \neq \emptyset$.*
r_i may deactivate *r_j if r_i is active in some database instance and the execution of action A_i can change the database instance from a state in which $C_j^{new} - C_j^{old} \neq \emptyset$ to a state in which $C_j^{new} - C_j^{old} = \emptyset$.*

The static analysis of CA rules in active databases is based on a general algorithm, called the *Propagation Algorithm (PA)*, which uses syntactic analysis to determine how a database query (C) can be affected by the execution of a data modification operation (A). The modification can consist of insertions, deletions and updates.

Definition 4.12 *The Propagation Algorithm PA takes as input a query Q (which can be the condition C of a CA rule) and a modification M (which can be the action of a CA rule), both expressed in extended relational algebra as defined in [Baralis and Widom, 2000]. The output of the algorithm is zero or more of each of the operations insert, delete, and update, characterizing how the result of the query may change due to the execution of the modification. If the algorithm produces an insert or delete operation, then the query may contain more or less data after the modification; if the algorithm produces an update operation, then the query may also contain updated data after the modification.*

We will not describe the details of this algorithm. It is not trivial to deduce from a query Q and a modification M whether the modification will add tuples to the result of Q, delete tuples from it or modify tuples in that result. We will discuss this issue when discussing CA rules for adaptive hypermedia in Section 4.6.

The static analysis of termination is to check the influence of the action of one rule on the condition of another rule by using the Propagation Algorithm for every pair of rules. This analysis process is defined as the calculation of an *Activation Graph*.

Definition 4.13 *The Activation Graph (AG) consists of a set of nodes and directed edges. Each node represents a CA rule. AG contains an edge $r_i \rightarrow r_j$ if and only if the Propagation Algorithm applied to A_i and C_j produces an insert and/or update operation. In other words $r_i \rightarrow r_j$ belongs to AG if and only if A_i may add elements to C_j^{new}. Similarly we can construct a Deactivation Graph (DG) consisting of edges $r_i \rightarrow r_j$ when A_i produces delete and/or update operations for C_j, or in other words when A_i may remove elements from C_j^{new}.*

The Activation Graph is the key to solving the termination problem:

Theorem 4.3 *If there are no cycles in the Activation Graph then the rule execution is guaranteed to terminate.*

The static analysis of CA rules [Baralis and Widom, 2000] is more precise than the one for ECA rules because it takes the condition into account when analyzing possible rule activation. However, the analysis is still conservative: termination is guaranteed when there are no cycles in AG, but it is possible that in a real application termination may be guaranteed even in a situation where AG contains a cycle. Only analysis of the possible database instances would reveal that.

Static analysis of confluence

Suppose that rule execution is guaranteed to terminate. We want the rule execution, which starts with the same initial state, to always end in the same final database state.

For confluence we need to know how rules activate and deactivate each other. Furthermore we also need to know whether the actions of rules interfere with each other in any other way.

Definition 4.14 *Consider two rules $r_i : C_i \rightarrow A_i$ and $r_j : C_j \rightarrow A_j$. The actions A_i and A_j commute if, for all database states, the execution of A_i followed by A_j and the execution of A_j followed by A_i produce the same database state.*

For checking whether A_i and A_j commute we apply the following "trick": We create two new rules: r' has as condition C', a query that corresponds to A_i, and r'' has as condition C'', a query that corresponds to A_j. We can then use PA to determine whether A_i may influence C'' or whether A_j may influence C'. (If PA produces $\emptyset$ as the set of insert, delete and update operations in both cases then A_i cannot change the effect of A_j and vice versa.)

From the paper [Baralis and Widom, 2000] we cite the following results:

Lemma 4.3 *Two distinct rules r_i and r_j commute if: (1) r_i cannot activate r_j; (2) r_i cannot deactivate r_j; (3) condition (1) and (2) with i and j reversed; (4) r_i's action and r_j's action commute.*

Note that a rule always commutes with itself, even though it may not satisfy the conditions (1) to (4).

Theorem 4.4 *A rule set R is confluent if all pairs of rules in R commute.*

To guarantee the commutativity of two distinct rules r_i and r_j, we need to verify the conditions (1)-(4) in the above Lemma. Condition (1) can be seen from the *Activation Graph*: an edge $r_i \rightarrow r_j$ indicates that r_i may activate r_j. Condition (2) can be seen from the *Deactivation Graph*. For condition (3) we only reverse the role of i and j. Condition (4) is checked as described above.

The time complexity of the static analysis of CA rules is determined by the time complexity of the Propagation Algorithm (PA). According to [Baralis and Widom, 2000], while in the case of queries containing aggregation, the execution time of the PA can be exponential in the depth of the query tree (or in rough approximation in the number of operations in the query). When the query does not contain aggregations, the execution time of the PA grows linearly with the depth of the tree. In any case, this depth is normally small and the analysis is performed at design time (not run-time) so time complexity should not be problematic.

The condition to guarantee confluence is very strict. Unfortunately it cannot be improved without taking properties of database instances into account. The strict condition becomes much less of a problem if priorities are associated with rules. Only rules with the same priority need to commute to guarantee confluence. In AHAM we use rule execution phases for this purpose.

4.4 Defining AHAM-CA, a Rule Syntax for the AE

To show how an adaptation engine works, we design a rule language for a general-purpose *adaptation engine* (AE). Based on this rule language we will focus on issues of the termination and confluence in Section 4.6, Section 4.7, and Section 4.8. At first sight it seems that ECA rules are best for AHS, because the system always reacts to an event generated by the user (such as clicking on a link anchor). However, after this initial event all rules are just triggered through changes in the user model. By translating the initial event to a simple user model update (updating some attribute value to represent the "click") we can also describe the behavior of an AHS through CA rules.

Since both DM and UM have a structure that consists of objects (like concepts and concept relationships) with an object identity and a set of (named) attributes it appears natural to base our rule language on trigger and query languages for databases. Doing so also makes it relatively easy to draw a parallel between the properties of our language with CA rules and those of CA rules in active databases, as studied in [Baralis and Widom, 2000]. In the paper [Baralis and Widom, 2000] the relational algebra syntax is used, but in order to obtain easily readable rules we base our language on the well-known SQL syntax. Our language and the static analysis issues were first introduced in [Wu et al., 2001].

4.4.1 Definition of AHAM-CA rule language

In Section 3.4 we defined the syntax of generic rules and specific rules for AHAM. We combined them as a tuple $<R,\ \{SC\},\ PH,\ PR>$ where R is a "triggered" rule; SC is optional, it is used for specific rules to represent a set of concept components used in the rule; PH is the "phase" and PR is the Boolean "propagate" field. To illustrate how an AE works, now we describe the syntax of a "triggered" rule R by the grammar described in Table 4.1. Note that we simplified details like where to use quotes and we also allow some meaningless constructs such as names with multiple dots, etc. Such details are not important for understanding how the language is used in AHAM-CA rules.

All concepts of the DM are present in the UM using the same name. We use *relationship* predicates to represent that a certain condition holds between two concepts in the DM. An example is $Prerequisite(C_1, C_2)$. Most other AHAM-CA rule constructs apply to the UM. Therefore we opt to sometimes omit the **from** clause. Whenever we use a concept (variable) C in a rule we assume that a clause "**from** UM **as** C" was present in the query. For concept relationships the same holds, but there we even omit a symbolic name to refer to the relationship (instance).

In our examples (below) we also use the convention that uppercase names are used to indicate concept variables (used in *generic* rules) whereas lowercase names are used for specific concepts (used in *specific* rules and in instantiated rules defined below) and for attributes.

In our later discussion of AHAM-CA rules, we analyse the rules according to different phases, but for now we ignore the *execution phase*, as well as the *propagate field* we introduced in Chapter 3. AHAM-CA rules form a subset of CA rules in active databases. They

have the same semantics as database CA rules, but the actions only contain updates, no insertions or deletions.

<R>	::=	$< C > \rightarrow < A >$
<C>	::=	<query>
<A>	::=	**update** <list_assign> {**where** <condition>}
<condition>	::=	<Boolean> | '(' <condition> ')' | **exists** <query> | <relationship> | <expr><eq_op><expr> | **not** <condition> | <condition> **and** <condition> | <condition> **or** <condition>
<eq_op>	::=	'=' | '≠' | '<' | '≤' | '>' | '≥' | **in**
<expr>	::=	<constant> | <name> | <query> | <expr><bmath_op><expr> | <umath_op><expr>
<relationship>	::=	<name> '(' <name> { ',' <name> } $^+$ ')'
<bmath_op>	::=	'+' | '−' | '×' | '/' | **max** | **min**
<umath_op>	::=	'−'
<constant>	::=	<Integer> | <Boolean> | <String> | <Set> | **null**
<Integer>	::=	<digit> {<digit>}*
<String>	::=	'"' <alpha> { <alpha> | <digit> | <stringspecial> }* '"'
<Boolean>	::=	**true** | **false**
<Set>	::=	∅ | '{' <constant> { ',' <constant> }* '}'
<query>	::=	**select** <name_list> { **from** UM **as** <name> } { **where** <condition> }
<list_assign>	::=	<name> := <expr> { ';' <list_assign> }*
<name_list>	::=	<name> { ',' <name> }*
<name>	::=	<alpha> { <alpha> | <digit> | <special> }* | <relationship> '.' <name>
<alpha>	::=	'a' ... 'z' | 'A' ... 'Z'
<digit>	::=	'0' ... '9'
<special>	::=	'_' | '.'
<stringspecial>	::=	<special> | ' '

Table 4.1: Definition of AHAM-CA rule syntax

4.4.2 Examples of AHAM-CA rules

The following examples are intended to illustrate our rule language. They are not meant to present generally accepted behaviour of AHS (as all systems behave differently).

Example 4.1 *The following (generic) rule specifies that all relevant fragments of page P will be shown to the user when this page is presented.*

C: ***select*** *P.access*
A: ***update*** *F.pres* := "show"
where *Fragment(P, F)* ***and*** *F.relevant* = ***true***

Fragment(P, F) is a predicate. The ***where*** *clause in A means:*

CR.cinfo.type = "Fragment" ***and*** *CR.ss*[1].*uid* = *P and CR.ss*[2].*uid* = *F*
and *CR.cinfo.dir*[1] = "FROM" ***and*** *CR.cinfo.dir*[2] = "TO"
and *CR.ss.length* = 2 ***and*** *CR.ss*[2].*uid.relevant* = ***true***

Here we have made the association between objects and attributes explicit and we "merge" DM and UM concepts. (Indeed, the *relevant* attribute is a UM attribute and is used here in a DM construct *CR.ss*[2].*uid*.) We will also do this in the following examples. Two parts in the rule of this example need some explanation:

1. The rule monitors a change in *P.access* because *P.access* appears in the **select** clause. So whenever a page is accessed this rule can be executed.

2. For all fragments F that are part of page P and for which the *relevant* attribute is **true**, the *pres* attribute is set to the value "show". Note that in a "real" SQL syntax we would need an **exists** and subquery here to check for the existence of the part-of relationship between F and P.

An interesting aspect of this example is that it shows how information is carried over from the *Storage Layer* to the *presentation specification* of the AHAM model.

Example 4.2 *Suppose that the knowledge value of a concept can have the values "not known", "known" and "well known". (In later examples we will also assume the property that "not known" < "known" < "well known".) A concept can be "ready-to-read" or not. Accessing a concept may have the effect described by the following rule:*

C: ***select*** *P.access*
where *P.ready* = ***true***
A: ***update*** *P.knowledge* := "well known"

Note that while the condition of this rule checks the value of two attributes, the rule is only triggered when the *access* attribute changes. The **select** clause tells the AE to only look for changes to *access*. Also, because this rule only changes attribute values for the object (page) that appears in the condition, the action has no **where** clause.

Example 4.3 *Suppose that a concept becomes "ready-to-read" when all its prerequisites are at least "known". We can write a rule to take this into account when a knowledge value changes:*

C: ***select*** $C_1.knowledge$
  where $C_1.knowledge \geq$ "known"
A: ***update*** $C_2.ready :=$ ***true***
  where $Prerequisite(C_1, C_2)$ ***and***
    not exists (***select*** C_3
      where $Prerequisite(C_3, C_2)$ ***and***
        $C_3.knowledge <$ "known")

Predicate $Prerequisite(C_1, C_2)$ *means that the following condition must hold in the DM:*

$CR_1.cinfo.type =$ "Prerequisite" ***and*** $CR_1.ss[1].uid = C_1$ ***and*** $CR_1.ss[2].uid = C_2$
and $CR_1.cinfo.dir[1] =$ "FROM" ***and*** $CR_1.cinfo.dir[2] =$ "TO"
and $CR_1.ss.length = 2$

The other predicate $Prerequisite(C_3, C_2)$ ***and*** $C_3.knowledge <$ *"known" represents that the following condition must hold in the DM:*

$CR_2.cinfo.type =$ "Prerequisite" ***and*** $CR_2.ss[1].uid = C_3$ ***and*** $CR_2.ss[2].uid = C_2$
and $CR_2.cinfo.dir[1] =$ "FROM" ***and*** $CR_2.cinfo.dir[2] =$ "TO"
and $CR_2.ss.length = 2$ ***and*** $CR_2.ss[1].uid.knowledge <$ "known"

This example means that when $C_1.knowledge$ has been changed to "known" or "well known", all the concepts for which C_1 is a prerequisite are verified, and for each of these concepts we check whether they have unsatisfied prerequisites; if they do not, then the *ready* attribute is set to **true**.

Example 4.4 *As another example let us consider the effect of an "inhibit" relationship that gets activated when a concept becomes "well known".*

C: ***select*** $C_1.knowledge$
  where $C_1.knowledge =$ "well known"
A: ***update*** $C_2.ready :=$ ***false***
  where $Inhibit(C_1, C_2)$

Predicate $Inhibit(C_1, C_2)$ *means that the following condition must hold in the DM:*

$CR.cinfo.type =$ "Inhibit" ***and*** $CR.ss[1].uid = C_1$ ***and*** $CR.ss[2].uid = C_2$
and $CR.cinfo.dir[1] =$ "FROM" ***and*** $CR.cinfo.dir[2] =$ "TO"
and $CR.ss.length = 2$

Example 4.3 and Example 4.4 together already show some of the potential problems with adaptation rules in general: if we allow the arbitrary creation of "prerequisite" and "inhibit" relationships and apply these adaptation rules it is possible that two rules are triggered at the same time, one of which tries to set *ready* attribute to **true** while the other one tries to set *ready* attribute to **false**, for the same concept. In Section 4.6 we describe how to detect such conflicts.

4.4.3 Other issues about the AHAM-CA language

The adaptation rules describe most of the adaptation behavior of AHS. (A part still depends on how the AE works.) What can be specified in the rule language has a direct impact on the expressive power and on the complexity of the system control. Because AHAM-CA rules are defined as a subset of database CA rules (but with a different syntax), we can discuss which aspects of active database rule languages are relevant for the study of AHAM-CA rules. We use the active database trigger mechanisms to simulate adaptation rule execution in AHS, but we have a much simpler situation than in active databases.

Events

In an active database rule, the *event* specifies what causes the rule to be triggered. Useful triggering evens are:

- Data modification: A data modification can be specified as one of three SQL data modification operations: *insert*, *delete*, or *update* on a particular table. In AHAM-CA rules we do not use events explicitly; events are implicitly combined in the condition. We use only *update* as an event to trigger rules. Update can be used to represent the (internal) event generated during the rule execution. For external events we use an attribute, e. g. the *access* attribute we used in Example 4.1 to represent the user requesting a page. When an external event occurs, the system will update the corresponding attribute value, and this update will trigger rules as an (internal) event.

- Data retrieval: A data retrieval event might be specified as a *select* operation on a particular table. We do not use it in AHAM-CA rules.

- Time: A temporal event might specify that a rule should be triggered at an absolute time (e. g. 1 Jan 95 at 12:00), at a repeated time (e. g. every day at 12:00), or at periodic interval (e. g. every 10 minutes). We do not consider time events in AHAM-CA rules. (They can easily be added.)

- Application-defined: An application-defined event might be specified by allowing an application to declare a name E as denoting an event (e. g. high-temperature, user-login, or data-too-large) and allowing active database rules to specify E as their triggering event. Then, each time an application notifies the database system of the occurrence of the event E, any rule specifying E as its event is triggered. In this way the application may perform any monitoring or computation it desires with or without accessing the database to detect when event E should occur. We do not use it in AHAM-CA rules, but we can use this to simulate external events in AHS if we use ECA rules for AHS.

For complicated situations, a single event is not enough; a composite event is used to combine single events or other composite events in active databases. Useful operators for combining events are:

- Logical operators. Events might be combined using the Boolean operators **and**, **or**, **not**, etc.
- Sequence. A rule might be triggered when two or more events occur in a particular order.
- Temporal composition. A rule might be triggered by a combination of temporal and non-temporal events, such as "5 seconds after event E_1," or "every hour after the first occurrence of event E_2."

We do not consider composite events in AHAM-CA rules.

Condition

In an active database rule, the *condition* specifies a condition to be checked once the rule is triggered and before the action is executed. There are four kinds of conditions:

- Database predicates: A condition can specify that a certain predicate must hold on the database, where the predicate is defined using a language corresponding to a condition clause in the database query language. We use *relationship* predicates like $Prerequisite(C_1, C_2)$ in AHAM-CA rules to represent that a relationship holds between two concepts in the DM.
- Restricted predicates: Condition evaluation can be the most expensive aspect of rule processing, so restrictions on predicates, e. g. not allowing aggregate operations and joins, are used for performance reasons. We do not consider using aggregation in AHAM.
- Database queries: The condition may specify a query using the database system's query language. The condition is **true** if and only if the query produces a non-empty answer. Note that because most database query languages include conditions as a component, there may be no difference in expressiveness between predicates and queries as rule conditions. A rule with any predicate P as its condition is equivalent to a rule with query Q(P) as its condition, where Q(P) retrieves all data satisfying predicate P. We only use database queries to specify the adaptation condition.
- Application procedures: A rule condition might be specified as a call to a procedure written in an application programming language, where the procedure may or may not access the database. We do not consider this in AHAM-CA rules as it makes analysis impossible.

Action

In an active database rule, the action is executed when the rule is triggered and its condition is **true**. Useful actions are:

- Data modification operations: A relational database system might allow rule actions to specify SQL *insert*, *delete*, or *update* operations. We only use *update* operations. (We only assign values to user model concept-attributes.) This is the singlemost factor that causes AHAM-CA rules to be much easier to analyse than active database rules in general.

- Data retrieval operations: A relational active database system might allow rule actions to specify SQL select operations. For the analysis of the behavior of AHAM-CA rules retrieval operations are not interesting because they cannot trigger other rules.

- Other database commands. A rule action might allow any database operation to be specified. In addition to data modification and retrieval operations, most database systems support operations for data definition, operations for transaction control (e. g. rollback, commit), operations for granting and revoking privileges, etc. We do not explicitly consider these in AHAM, but we assume that the rule execution that results from an event is a transaction that must be committed.

- Application procedures: A rule action might be specified as a call to a procedure written in an application programming language, where the procedure may or may not access the database. Again, we do not consider this in AHAM-CA rules as it makes analysis impossible.

4.5 The Semantics of AHAM-CA Rule Execution

In the previous subsections we presented (part of) the syntax used in the specification of rules, and gave some examples. To determine the actual behavior of the AHS we must describe how the *adaptation engine* actually selects and executes the *adaptation rules*. We call this the *execution model*.

4.5.1 Rule execution phases

Many rules must be "considered" during rule execution. (After each action the AE must determine which rules have become active.) In order to make the AE more efficient and also to reduce the time complexity of the static analysis of the rules we distinguish a number of *phases* in which the rules are executed that serve a similar purpose.

For the events "login" or "register", if the user logs in for the first time, the system builds a user model for the user and **i**nitializes that **u**ser model (phase IU), and selects the first page to adapt and show to the new user. (This is done through the *resolver* and *accessor* functions which we do not describe.) If the user logs in as an existing user, the system can use the already existing user model of that user, and only needs to select the first page to adapt and show to the user.

If the user accesses a page, the system first performs initialization of volatile attributes of concepts in the phase IU. The system then **u**pdates the **u**ser model (phase UU-pre), it

generates the **a**dapted page (phase GA), and finally it may **u**pdate the **u**ser model (phase UU-post) some more to reflect some user's "feature changes" after the adaptation.

The partitioning of rules into these phases and the sequential execution of these sets of rules can be realized through the association of rules with a priority. The AE will start by executing rules in IU, which must have the highest priority. Rules in UU-pre come later by giving them a lower priority, and by making sure that these rules do not cause rules from IU to be executed again. Rules from GA have a still lower priority and may not activate any rules from IU or UU-pre. Rules from UU-post have the lowest priority. It is easy to write conditions to ensure that rules never start a rule from a previously completed phase. In the sequel we will limit ourselves to rules within a single phase and not worry about accidentally invoking rules from another phase.

4.5.2 General constraints

It is all too easy to define sets of rules that trigger each other indefinitely and thus cause infinite loops in the AE. It is clear that some restrictions are needed to prevent undesired effects such as infinite loops. We first describe some "common sense" constraints that each rule system for AHS should observe.

- If the C part and the A part of a CA rule contain an attribute of the same concept then the C part must include an (attribute that represents an) event. This constraint basically says that an event that causes several attributes of a concept to be updated must do so through CA rules that are activated by this event. The event should not just cause an update of one attribute and then have that update trigger an update of another attribute of the same concept. We would consider this a sign of bad design because it hides the direct relationship between the event and the updates it causes.

- If the C part and A part of a *generic* adaptation rule contain different concept variables, then the *condition* (the **where** clause) of the A part must also include a concept relationship linking these concepts. Thus, propagation of user model updates between different concepts is only allowed then these concepts are linked through concept relationships.

- Rules with events have higher execution priority than rules without events that belong to the same execution phase. This means that rules with an event in their C part will all be executed first and then the AE will look at the rules that depend on attribute value changes in their C part.

We can illustrate these constraints by the examples in Section 4.4.2. Example 4.2 satisfies the first constraint. The update to the *knowledge* (attribute) of P depends on the *ready* attribute of the same P, and therefore must be triggered by an event such as $access(P)$.

Example 4.3 satisfies the second constraint. It says the change of the knowledge of concept C_1 propagates to the *ready* attribute of concept C_2 when there is a relationship $Prerequisite(C_1, C_2)$.

Example 4.2 will execute before Example 4.3; this illustrates the third constraint. However, Example 4.1 will execute later although it also contains an event. This is because it is a rule from the GA phase (it sets a presentation specification) whereas the other rules belong to the UU-pre phase.

It is clear that while the above constraints eliminate certain meaningless rule sets, infinite loops in the rule execution are still possible when there are cycles in the relationships that are used in the adaptation rules. For some common types of relationships, such as *prerequisites* we can require that they do not have cycles, but this restriction may be too severe for other types of relationships, e. g. *link* relationships. And even if there are no cycles when considering relationships of one type, the interaction between concept relationships of different types may still result in infinite loops. In Section 4.8 we describe constraints that eliminate termination and confluence problems. In Section 4.6 we will first discuss how to predict possible infinite loops and non-deterministic results in general.

4.5.3 Instantiating rules

The AE needs to *instantiate* the generic rules and specific rules before it can apply them. When a generic rule says that "when a page P is accessed $P.read$ is set to **true**", the AE will actually use a rule for each page p, saying that when page p is accessed $p.read$ becomes **true**. (Note that P means a variable and p means a concrete page.) Also, when a condition contains a concept relationship, the actual instance of the relationship is used to make the rule specific. For instance, let c_1 be a concept and let c_1 be an inhibitor for c_2 and c_3 (i. e. the relationships $Inhibit(c_1, c_2)$ and $Inhibit(c_1, c_3)$ exist). We can then instantiate Example 4.4:

Example 4.5 *The rule in Example 4.4 is instantiated to the following rule. It updates c_2 and c_3 separately.*

C: ***select*** $c_1.knowledge$
  where $c_1.knowledge =$ "well known"
A: ***update*** $c_2.ready :=$ ***false****;*
  $c_3.ready :=$ ***false***

The instantiation of rules makes it easy to detect the presence of specific rules that are in conflict with some generic rule. The instantiation of the generic rule for the objects that appear in the specific rule is then discarded.

The following pre-process translates a set of generic and specific rules to a set of instantiated rules (Ins_rules).

1. Let Ins_rules = $\emptyset$.
2. Instantiate all generic rules and add to Ins_rules.
3. Consider each specific rule and remove an instance of a generic rule from Ins_rules if both following conditions hold:

- their conditions are logically equivalent
- the updates of the A parts update the same attributes

4. After instantiating all generic rules to specific rules, all syntactically equivalent rules will be merged into one rule.

From now on we will always mean "instantiated rule" when we use the term "rule". (Sometimes we may wish to issue a reminder that rules are instantiated.) Using instantiated rules is the key to obtaining better conditions to guarantee termination and confluence than for active databases. Indeed, instantiated rules make use of the *instance* of the DM and not just the scheme. The analysis described in Section 4.6 is still a static analysis in the sense that it only uses information known at design time: the scheme and instance of the DM but only the scheme of the UM.

4.5.4 Other issues about the semantics

We give a brief description of other issues about AHAM-CA semantics according to relational active database rule languages [Widom and Ceri, 1996].

Rule processing granularity

The granularity of rule processing specifies how often the points occur at which rules may be processed, i. e., how often the rule execution algorithm is invoked. The finest granularity is that rules may be processed at any point during the database system's execution, as soon as any rule's triggering event occurs. In a relational database system database updates are typically defined as SQL operations. A database may start considering triggered rules for execution as soon as a single tuple has been added, deleted or modified, or may wait until the entire SQL statement finishes. A database may also delay rule processing until it reaches the end of a transaction (consisting of several SQL statements). It is possible but unusual for rule processing granularity to be coarser than the transaction boundaries.

In AHAM the condition C of a rule selects only one tuple, but the rule's action A may contain several updates to different attributes of (possibly different) concepts. We may consider the rule execution to be tied to the execution of a complete instantiated rule or to each statement in the rule's action. While it may seem most logical to only consider the granularity of the complete instantiated rule we cannot rule out the other option in the reference model as there may be AHS that trigger rules after each statement in a rule's action. In fact, the AHA! system [De Bra and Calvi, 1998c] exhibits this behavior (at least in version 1.0).

Instance-oriented versus set-oriented execution

Active database rule execution is instance-oriented if a rule is executed once for each tuple in the evaluation of the rule's condition (i. e. in the query result). Active database rule execution is set-oriented if a rule is executed once for the whole database instance

triggering the rule or satisfying the rule's condition. In AHAM-CA based AE, we use instantiated rules that have a condition containing only one specific tuple of a database instance (UM). For example, the rule instance in Example 4.5 has a condition that only contains c_1. AHAM-CA rule execution is thus instance-oriented execution.

Iterative versus recursive execution algorithm

An iterative execution algorithm selects and processes one rule, then selects and processes another rule, and so on. The AHAM-CA rule execution algorithm is iterative, e. g. when the rule instance in Example 4.5 is active and selected, it updates c_2 and c_3 before looking for the next active rule. A recursive algorithm would invoke rule execution recursively during the execution of another rule's action. In the example the update of c_2 may then be followed by other rule executions before returning to the update of c_3. In our study of termination and confluence we will concentrate on iterative rule execution. (But note that AHA! 1.0 [De Bra and Calvi, 1998c] uses recursive execution.)

Conflict resolution (Scheduling)

Several rules may be triggered simultaneously. There are three possible reasons for that:

- Several rules specify the same triggering event.
- The rule processing granularity is coarse enough so that many triggering events may occur before rules are processed.
- A rule is triggered but not (yet) selected, and the rule is still triggered the next time when the AE selects rules.

All these reasons exist in an AHAM-CA based AE. Conflict resolution is a way to select one rule from a set of triggered rules (adopted from the same concept in AI rule languages). There are numerous possibilities for conflict resolution; for example:

- A rule may be chosen arbitrarily.
- A rule may be chosen based on priorities specified in the definition of rules.
- A rule may be chosen based on other static properties of rules, such as the time of rule creation or the data on which rules are defined.
- A rule may be chosen based on dynamic properties of rules, such as the time of rule triggering. (Triggered rules may be placed on a stack or a queue of rules waiting to be selected for execution.)

We have four execution phases which have a linear order. In each phase we choose a triggered rule arbitrarily. Our design goal is to make authoring easier. Therefore we do not rely on priorities or stack or queue mechanisms to solve confluence problems. Such

measures would guarantee that the result is predictable in theory, but it may be too hard for authors to understand how to predict which rules are going to be executed in which order.

Sequential versus concurrent execution

The AHAM-CA rule execution algorithm performs rule processing in a sequential manner: one rule is executed at a time. A conflict resolution mechanism described in the above subsection may be used to choose which rule to execute when multiple rules are triggered. An alternative to executing one rule at a time is concurrent rule processing: if multiple rules are triggered, the rules' conditions are evaluated and their actions are executed concurrently. Concurrent rule processing avoids the issues of conflict resolution and, in the appropriate setting, can speed up rule processing. However, this requires that concurrently executing rules do not interfere with each other (which is hard to guarantee).

Coupling modes

Coupling modes discuss the relationship between rule processing and database transactions. For Event Condition Action rules, one needs two coupling modes. One coupling mode can specify the transactional relationship between a rule's event and the evaluation of its condition, while another coupling mode can specify the transactional relationship between a rule's condition evaluation and the execution of its action. For Condition Action rules, one needs one coupling mode. It can specify the transactional relationship between a rule's condition evaluation and the execution of its action. Possible coupling modes are:

- Immediate: the execution of the action of rules is performed immediately after the the evaluation of the conditions of rules when the condition is **true**. We can realize the priority of rule execution order by the set its evaluation order. The way we described the AE's behavior in Section 4.1.1 yields an immediate mode: rules become active immediately after their condition becomes **true**. AHAM-CA rule execution takes immediate mode.

- Deferred: the execution of the action of rules takes place at the commit point of the current transaction. We do not consider deferred coupling mode in AHAM-CA.

- Decoupled: the execution of the action of rules takes places in a separate transaction. We do not consider this mode either.

For the static analysis we perform in Section 4.6 it makes no difference whether we use the immediate or deferred mode.

4.6 Static Analysis of AHAM-CA Rules

In [Aiken et al., 1995; Baralis and Widom, 2000] the static analysis only looks at the database scheme, not the actual instance. So they must include every possible activation

or deactivation in the AG and DG. We can make our static analysis much more precise (or less conservative) than that of [Aiken et al., 1995; Baralis and Widom, 2000] by trying to *only* include edges when this activation is really possible (for some instance of the UM), given the DM instance that is available at design time. The use of the DM instance is effectuated by using instantiated rules in the static analysis.

4.6.1 The Propagation Algorithm for AHS

The static analysis of AHAM-CA rules is based on the activation graph AG and the deactivation graph DG, both of which are constructed using a *propagation algorithm* PA. The main differences between AHS as described by AHAM-CA rules and active databases as described in [Aiken et al., 1995; Baralis and Widom, 2000] are:

- In databases we have to consider *insert*, *delete* and *update* operations, whereas in AHS we only have *updates* to the user model. (The AE only changes attribute values for concepts.)
- In AHS the updates are defined using *instantiated* rules.
- In active databases many different possible updates can be the triggers for rules whereas in an AHS the rule execution starts after a very specific type of update that represents a user-generated event like clicking on a link anchor.

Below we first describe the general propagation algorithm. Then we show how the algorithm can be improved for the use with AHS.

The basic PA described in [Baralis and Widom, 2000] uses the action of r_i (say A_i) and the condition of r_j (say C_j) to decide whether r_i may activate r_j. The PA of [Baralis and Widom, 2000] does this by combining the relational algebra expressions of A_i and C_j to calculate an expression that defines which tuples will be added to, removed from or changed in the query result of C_j. For (instantiated) AHAM-CA rules the condition C names an attribute of a concept in its **select** clause. The rule is triggered only when this attribute of this specific concept changes. The action A of a rule contains assignments to attributes of (specific) concepts. It is clear that A_i can only influence C_j if A_i contains an assignment to the attribute of the concept in the **select** clause of C_j. Our basic PA for AHAM-CA rules is used to build AG and DG in the following way: an edge $r_i \rightarrow r_j$ is added to AG and DG if A_i assigns a value to the attribute of the concept used in the **select** clause of C_j. This may seem peculiar because the resulting AG and DG will be the same. It makes sense though because we only know to which attribute of which concept a value is assigned. We do not know which value, and we thus do not know if the **where** condition of C_j will be satisfied or not. (We would need the instance of the UM to decide this, and that is not available at design time.) We also do not know if the assignment actually takes place because without the UM instance we also cannot evaluate the condition (**where** clause) of A_i.

As an example it is easy to see that the action of the rule in Example 4.2 can result in the condition of the rule in Example 4.3 becoming **true** for the same concept. Accessing one

page may "generate" the last bit of required prerequisite knowledge for another concept. (The PA works on instantiated rules to decide exactly which edges to add to AG and DG based on which prerequisite relationships actually exist in the DM instance.)

There are a number of ways in which the basic PA can be improved, while remaining conservative (i. e. while guaranteeing that it detects all possible activations and deactivations):

1. Properties from the value domains of attributes can be used to determine that an update changes the value to something that the condition can accept. If A_i sets a knowledge attribute to "not known" and C_j checks for knowledge = "known", we know that r_i cannot activate but may deactivate r_j. If the assignment in A_i uses a constant and the condition in C_j compares the attribute value to a constant then this refinement can be done. In general the assignment may also use attributes of concepts (instead of constants) and the condition may compare with attributes of concepts (instead of constants) and then this optimization is not possible.

2. In [Baralis and Widom, 2000] it is already remarked that we can also include C_i in the process: when action A_i is executed it must be executed in a situation in which C_i was satisfied. This can possibly provide some information needed to see whether C_j will be satisfied after executing A_i.

3. Since we know the "event" that starts the AE we can start the construction of the activation graph at the concept with the initial "event", e. g. the concept for which the *access* attribute becomes **true**. We can thus improve on the above step by taking into account all the conditions of the rules on a path from the rule that is triggered first to the rules r_i and r_j for which we are investigating the possible activation. (For every attribute of every concept we need to maintain a range of possible values and can use that in evaluating whether A_i may change C_j.

4. A final improvement can be obtained by constructing a separate activation graph for each possible "event" that starts the AE. Indeed, for every link anchor we can construct an activation graph that represents the possible rule executions that are a consequence of a user clicking on that link anchor.

Following and extending [Baralis and Widom, 2000] we can analyze the behavior of the AHS by using the AG and the DG. In such graphs nodes represent instantiated rules and edges indicate that one instantiated rule may activate or deactivate the other instantiated rule.

The following theorem is quite straightforward:

Theorem 4.5 *A set of AHAM-CA rules R terminates if there are no cycles in the Activation Graph.*

For confluence we need to find out if rules commute. We recall that Lemma 4.3 states that two distinct rules r_i and r_j commute if: (1) r_i cannot activate r_j; (2) r_i cannot

deactivate r_j; (3) condition (1) and (2) hold with i and j reversed; (4) r_i's action and r_j's action commute. Conditions (1), (2) and (3) only require us to look at AG and DG. To decide whether the actions A_i and A_j commute (i. e. that their execution order never matters) we can check that the assignment(s) of A_i do not change an attribute of a concept used in the condition or assignment of A_j and that the assignment(s) of A_j do not change an attribute of a concept used in the condition or assignment of A_i.

Theorem 4.4 translates literally to AHAM-CA rules:

Theorem 4.6 *A set of AHAM-CA rules R is confluent if all pairs of rules in R commute.*

The sufficient condition for confluence is very strict because it forbids activation and deactivation. However, when using priorities confluence can be guaranteed under much less strict conditions (as we shall see in Section 4.8). The introduction of execution phases is also a step towards reducing the confluence problems.

4.7 Enforcement for AHAM-CA Rules

Termination and confluence can be enforced (when not guaranteed) by either placing restrictions on the circumstances in which rules are allowed to activate each other, or by monitoring the rule execution process and interrupting it when a problem is detected. The former is called *static enforcement*, the latter *dynamic enforcement*. We are not going to discuss dynamic enforcement because it confronts end-users with termination and confluence problems. We want to ensure termination and confluence before the rule execution starts, so that end-users never receive error messages about these problems.

4.7.1 Enforcement of termination

We propose some static enforcement methods for AHS that ensure that the rule execution will stop after some time, even when the rule definitions (together with cycles in the DM) cause loops. This approach is easier for the author, because the author need not know or be informed about (potential) loops. However, the enforced termination may not leave the system in the state the author desired. We propose three easy (and also simplistic) ways to make an AE ensure termination of each transaction (or rule execution process):

1. The system changes the attribute value of a concept at most once per transaction. Because UM has only a finite number of concepts and each concept has a finite number of attributes, the number of possible update steps that do not change a previously updated attribute value is finite. Unfortunately this method inhibits some possibly interesting ways to update the UM. For example, if concept A contributes knowledge towards B and C, and B and C both contribute (a possibly different amount of) knowledge towards D, D's knowledge value can only be updated in a predictable way if knowledge propagation from B and from C are both allowed to happen during one event-process. One can easily come up with similar examples

using concept relationships instead of the concept hierarchy, and also leading to "premature" termination of parts of the transaction.

2. Each rule instance (either an instance of a generic rule or a specific rule) is executed at most once in one event-process. A generic rule may be used several times, but with different bindings of its (concept) variables to actual concepts. This method again guarantees termination. In the above example the knowledge value of D can be updated twice because both updates are different rule instances. Unfortunately, if in this example D also contributes knowledge to E, the resulting two knowledge contributions cannot both be propagated to E because that would be done through the same rule instance.

3. The AE can make use of properties of the value domain for each attribute (of concepts) to determine whether repeated updates to a concept or repeated execution of the same rule instance are potential sources of infinite loops. The AE of the AHA! system for instance allows repeated monotonic updates to a concept's knowledge value. This poses no danger when the value domain consists of integers between 0 and 100. (All monotonic update loops terminate when the value reaches 100 or 0, depending on whether the value monotonically increases or decreases.)

The first two methods can be modified so that they allow not one but a larger (fixed or variable) number of updates or instances of rules to be executed. This may eliminate some of the negative side effects, but it also slows down the AE in case of an infinite loop that must be interrupted. The third method is preferable, but for some value domains it may be difficult to come up with a property that provides a good basis for terminating potential infinite loops.

4.7.2 Enforcement of confluence

A single event may activate several rules, and each rule execution may activate some more rules. In the sequential model we presented in Section 4.3.2, active rules are executed one by one. The order in which rule instances are executed may influence the final result UM_f. Also, when a potential infinite loop is cut short by one of the methods described in Section 4.7.1, the order in which rules are executed may influence which rules are executed and which rules are discarded. The easiest way to guarantee deterministic behavior of AE is to define the rule language in such a way that the author has to indicate *when* a rule must be executed, and not only *under which conditions* a rule may be executed. However, that would be like replacing the declarative nature of our language by an imperative one. This would make authoring more difficult and we therefore do not consider this acceptable. We have chosen an intermediate approach: the author must assign rules to the four categories IU, UU-pre, GA and UU-post, but within each category the author does not indicate any execution order.

While we do not have a general solution, an AE can find out potential sources of non-confluence by searching for conflicting user model updates that result from active rules.

For instance, considering Example 4.3 and Example 4.4, what these rules actually imply is that there cannot simultaneously be a prerequisite and an inhibit relationship between C_1 and C_2. If these relationships would exist however then whether $C_2.ready$ becomes **true** or **false** depends on the order in which the rules are executed. A smart authoring tool can detect that the rules have a conflicting outcome, and thus warn the author of the error in either the given relationships or the supplied rules. In fact, the check for commuting actions that is part of the static confluence analysis detects potential conflicting updates (but may give false alarms). When an author mistakenly thinks that a potential conflict will not occur in practice the AE can still be instructed to detect conflicts. What is needed then is a conflict resolution strategy that must be specified by the author or the system and that can be used to eliminate the conflict. Once all conflicts between the active rules have been resolved, the order of execution is not important any more and the event-process terminates in a well-determined state. In the case of the prerequisite and inhibit relationships one might for instance state that prerequisites take precedence and that in case of conflict the rule for the inhibitor relationship is not executed. In the case of a conflicting generic and specific rule, the specific rule always takes precedence.

A viable approach is to assign adaptation rules to groups and specify some general precedence relationships, which will constitute the AE default behavior. This will leave a number of situations in which the author has to provide some mechanism of choice for the order-dependent transactions. An attractive option appears to be the collective application of all rules that are active in a particular state and provide a conflict resolution strategy. This way, we can build general AEs that provide a clear separation of responsibilities between the system and the author. The input of the author will, of course, always be required and is best put in the form of overruling general AE behavior with specific, domain dependent choices.

4.8 Constraints for AHAM-CA Rules

Theorems 4.5 and 4.6 give sufficient conditions to guarantee termination and confluence. However, these conditions may be too restrictive. Especially the condition for confluence is strict as it does not allow any rule to trigger any other rule. In this section we discuss how to loosen the sufficient conditions for termination and confluence without losing these properties. This is an extension of the theory we published in [Wu and De Bra, 2001].

In common AHS the adaptation requirements are quite simple, e. g. each rule propagates the change through one relationship, and the relationship graph is a DAG (directed acyclic graph). For such simple cases, we can design a simple way to test for termination and confluence by enforcing some constraints on the rules.

In this section we only consider concept relationships that are binary. In adaptation rules, when an expression like $relationship(C_1, C_2)$ is used it can only be used to find the concepts C_2 that correspond to C_1. If we were to allow the relationships to be used in both directions we cannot guarantee termination by only looking at the structure of the rules. We can create an "inverse" relationship type, which is a *different* relationship type, and

apply constraints for different relationship types to a type and its inverse.

4.8.1 Definition of terms

When considering constraints for rules, we mainly look at possible ways in which a pair of rules may influence each other. This involves two aspects:

- Rules may activate or deactivate each other when the updated attribute (of a concept) is used in the **select** clause of the C part of another rule.
- Rules that are (or can be) active at the same time may influence each other's update result thus causing the result of a transaction to depend on the execution order.

We first introduce terminology to be able to formalize our constraints. In this section we will deal with constraints that use rules as given in the AM, not with instantiated rules.

Definition 4.15 *The terms used in Section 4.8 are defined as following:*

1. *The function att.exp for expressions (e_i) to attribute sets a is defined inductively as follows:*
   (a) $att.exp(a) = a$ *if a is an attribute*
   (b) $att.exp(constant) = \emptyset$
   (c) $att.exp(\{e_1, \cdots, e_n\}) = \bigcup_{i=1}^{n} attr.exp(e_i)$ *(for query results)*
   (d) $att.exp(e_1\ bmath\_op\ e_2) = att.exp(e_1) \cup att.exp(e_2)$
   (e) $att.exp(umath\_op\ e_1) = att.exp(e_1)$
2. *Let R: $C \rightarrow A$*
   (a) *S(R)=the set of attributes which are selected in C*
   (b) *U(R)=the set of attributes to which values are assigned in A*
   (c) *E(R)=the set of attributes used in the right-hand side of assignments in A*
3. *Let R: $C \rightarrow A$, the function num.rel for the number of relationships used in the **where** clause of A (A.**where**) is defined inductively by*
   (a) $num(relationship) = 1$
   (b) $num(e_1\ eq\_op\ e_2) = 0$
   (c) $num(not\ condition) = num(condition)$
   (d) $num(constant) = 0$
   (e) $num(exist\ query) = 0$
   (f) $num(e_1\ and\ e_2) = num(e_1) + num(e_2)$

(g) $num(e_1 \text{ or } e_2) = num(e_1) + num(e_2)$

4. *We divide the attributes in three sets:*

   (a) *Ext.att represents the external events.*

   (b) *Int.att represents the internal events to start different execution phases, generated by the system.*

   (c) *Usr.att represents the user features.*

5. *Let R: $C \rightarrow A$, Sel(C) is the set of concepts selected by C.**where**, Upd(A) is a set of concepts selected by A.**where**. We say that R is*

   (a) *a start rule* st_rule *if* $S(R) \subseteq Ext.att \cup Int.att$, $num(R) = 0$, *and* $Sel(C) = Upd(A)$

   (b) *a propagation rule* pr_rule *if* $S(R) \subseteq Usr.att$, *and* $(\forall c_2 \in Upd(A) : (\exists c_1 \in Sel(C) : (\exists rel \in$ *UM-rel:* $relationship(c_1, c_2))))$

6. *Others*

   (a) *St_rule is a finite set of start rules.*

   (b) *Pr_rule is a finite set of propagation rules.*

   (c) *Rule(rel) is a finite set of rules that propagate their change through the relationship type rel.*

   (d) *Pri(R) is a number to represent the priority of the execution order of rule R.*

   (e) *DM_rel is the set of relationship types in the DM.*

   (f) *Rule is a finite set of all rules in the AM.*

Function *attr.exp* calculates the number of different attributes that appear in an expression. In the simplest case we may want only one attribute in an expression. $S(R)$ represent the set of attributes *s*elected in the C part of a rule R. $U(R)$ is the set of attributes being updated (appearing in the left-hand side of the assignments) in the A part of a rule R. $E(R)$ is the set of attributes used in the *e*xpression of the assignment in the A part of a rule R. Function num.rel calculates the relationships used in the A part. In the simplest case we limit this to one relationship. $Ext.att$, $Int.att$, and $Usr.att$ are used to distinguish different kind of attributes in the UM. Rules having these different types of attributes as trigger events have different execution orders because we want the external event to trigger the rules and we divide rules into four different phases to simplify the static analysis.

A start rule *st_rule* which is the first rule to execute in each phase will use $Ext.att$ and $Int.att$ as its events. We can use $Int.att$ to connect the execution of the four phases. Rules triggered by events may update attributes of concept(s) appearing in their condition. The "external" source is required in *st_rules*, otherwise it is a sign of bad design, because we want the external event to trigger the rules and to update all relevant attributes of one

concept. This avoids recursive triggering among different attributes of one concept. (See also our first general constraint in Section 4.5.2.) A propagation rule pr_rule that executes after the start rules is triggered by some change in the UM and it propagates the change of values to different concepts through relationships between these concepts. (See also our second general constraint in Section 4.5.2.) $Rule(rel)$ is set of propagation rules that use relationship rel. Pr_rule is a set of propagation rules in general, and St_rule is a set of st_rules.

4.8.2 Constraints

Now we discuss the sufficient conditions for termination and confluence. We start with a simple but very strict Constraint 4.1.

Constraint 4.1 $\forall R_i, R_j \in Rule : (S(R_i) \cap U(R_j)) = \emptyset$

Constraint 4.1 specifies that for every two rules R_i, R_j, the set of select attributes of one rule is disjoint from the set of attributes updated in other rules. So they will not activate or deactivate each other. With Constraint 4.1, a set of finite rules applied to a finite database will always terminate; in worst case the rule execution terminates after N steps, where N is the number of rules.

Theorem 4.7 *A rule set Rule satisfying Constraint 4.1 terminates.*

Proof: $\forall R_i, R_j \in Rule$, the execution of R_i cannot active R_j, because $(S(R_i) \cap U(R_j)) = \emptyset$, that means the execution of action A_i cannot change the database from a state in which $C_j^{new} - C_j^{old} = \emptyset$ to a state in which $C_j^{new} - C_j^{old} \neq \emptyset$.

Constraint 4.1 still cannot guarantee confluence, because the updated attribute may appear in the expression of the assignment of another rule, or $(S(R_i) \cap E(R_j)) \neq \emptyset$.

Definition 4.16 $\forall R_i, R_j \in Rule$:

1. R_i *is independent from* R_j *if* $(S(R_i) \cup U(R_i) \cup E(R_i)) \cap U(R_j) = \emptyset$
2. R_i *is self-independent if* $(S(R_i) \cup E(R_i)) \cap U(R_i) = \emptyset$

Constraint 4.2 $\forall R_i, R_j \in Rule$: R_i *is independent from* R_j, *and* R_i *is self-independent*

Constraint 4.2 implies Constraint 4.1, which means it guarantees termination. It also requires that each rule is independent from the other rules, and it is also self-independent. That means the execution of any rule cannot affect the outcome of any other rule execution.

Theorem 4.8 *A rule set Rule satisfying Constraint 4.2 terminates and is confluent.*

Proof: Constraint 4.2 implies Constraint 4.1, hence the rule set *Rule* terminates. By Constraint 4.2, $\forall R_i, R_j \in Rule$, R_i cannot activate and deactivate R_j because A_i cannot influence R_j (it cannot change C_j^{new}) and vice versa. A_i and A_j commute according to Definition 4.14 because the execution order of them will not influence the final instance of UM. According to Lemma 4.3 R_i and R_j commute. And by Theorem 4.6 the rule set *Rule* is confluent.

Constraint 4.2 is a sufficient condition for termination and confluence. For each rule, the computational complexity of verifying that the rule is self-independent is $O(M^2)$, where M is the number of attributes. For each pair of rules the complexity of verifying independence is also $O(M^2)$. When there are N rules, the computational complexity of the algorithm to verify these constraints is $O(N^2 \times M^2)$.

Constraint 4.2 is very strict in the sense that it is impossible to describe any propagation (i. e. that a rule triggers other rules). It can only be used to allow the description of very simple adaptation. For more complicated applications authors need a more powerful rule language, in which they are allowed to express propagation. To provide more expressive freedom to authors we define a set of "reasonable" constraints that still enable the description of adaptation found in common AHS while maintaining the termination and confluence properties.

We first describe a general constraint for rules to be semantically correct in AHS as described in Section 4.5.2. We discuss termination and confluence under these general constraints.

Constraint 4.3 $Rule = St\_rule \cup Pr\_rule$, *and* $\forall R_i \in St\_rule,\ \forall R_j \in Pr\_rule : Pri(R_i) > Pri(R_j)$.

Constraint 4.3 means that the set of rules consists of start rules and propagation rules. This constraint also describes that (in each phase of the transaction if we divide rules into phases), the start rules execute before the propagation rules. Constraint 4.3 describes a general constraint for rules to be semantically correct in AHS. Of course Constraint 4.3 is not enough to guarantee termination and confluence.

Constraint 4.4 *Every relationship graph except that of the hyperlinks is a DAG.*

Constraint 4.4 is a constraint on DM. Hyperlinks are used as a way for connecting pages. The link structure is very often cyclic, but is not used for propagating user model changes. Other relationships are used to propagate the UM changes between different concepts. Cyclic relationship graphs are not very common. They require extra conditions in their adaptation rules to prevent infinite loops. Termination is easier to guarantee if we simply forbid cyclic relationships.

Constraint 4.5 $\forall rel_1, rel_2 \in DM\_rel$, *if* $rel_1 \neq rel_2$ *then* $\forall R_i \in Rule(rel_1), \forall R_j \in Rule(rel_2) : U(R_i) \cap S(R_j) = \emptyset$.

Constraint 4.5 means that rules using different types of relationships cannot trigger each other. This constraint forces propagation in a simple way. It forbids complicate propagation cases that use several different relationship graphs at the same time. Combining different relationship types could lead to infinite loops because the union of two DAGs may have cycles.

The following theorem describes sufficient conditions for termination. A rule set *Rule* consists of a finite number of *st_rules* and *pr_rules*. The *st_rules* will not trigger each other; they are triggered by external and internal events. The *st_rules* may trigger the *pr_rules*, and the *pr_rules* may also trigger *pr_rules*. The propagation for rules that use a relationship type always terminates because the relationship graph is a DAG. And rules that use different types of relationships cannot trigger each other, because they use different attributes, so different DAGs cannot be combined to form a cycle.

Theorem 4.9 *A rule set Rule terminates if it satisfies Constraints 4.3, 4.4 and 4.5.*

Proof: Assume there exists a cycle in the Activation Graph, then there exist two different relationship types that are used in this cycle, because every relationship graph is a DAG according to Constraint 4.4. Suppose these two different relationship types are rel_1 and rel_2, then there exist $R_i \xrightarrow{rel_1} R_j$ and $R_j \xrightarrow{rel_2} R_k$, that means $U(R_i) \bigcap S(R_j) \neq \emptyset$. This contradicts with Constraint 4.5. So we conclude that there does not exist a cycle in Activation Graph. By Theorem 4.5 that means the rule set *Rule* terminates.

Constraints 4.3, 4.4, and 4.5 together cannot guarantee confluence. For example, in one relationship graph, there may exist one node which has two incoming edges, that means there exist two ways to propagate different changes to this node. At a certain time, this node may have been changed to different values by choosing a different propagation order.

Constraint 4.6 *Every node in every type of relationship graph has at most one incoming edge, and* $\forall rel \in DM\_rel : (\forall R_i, R_j \in Rule(rel)$, *if* $R_i \neq R_j$ *then* $U(R_i) \cap U(R_j) = \emptyset)$.

Constraint 4.6 (combined with constraint 4.4) says that every relationship graph is not just a DAG but a tree. It also says that different updates made through different rules using the same relationship type must use different attributes. It thus implies that updates to an attribute of a concept cannot come from two sources (through the same relationship type).

When two propagation rules that use different types of relationships become active at the same time, the different execution order of these two rules may cause a different state in the processing of a set of rules, and this may cause different final results.

Definition 4.17 $\forall rel_i, rel_j \in DM\_rel$, $Independent(rel_i, rel_j)$ *holds if*
$\forall R_i \in Rule(rel_i)$, $\forall R_j \in Rule(rel_j)$:
$(S(R_i) \cup U(R_i) \cup E(R_i)) \cap U(R_j) = \emptyset$ *and* $(S(R_j) \cup U(R_j) \cup E(R_j)) \cap U(R_i) = \emptyset$.

This definition says that rules associated with rel_j do not update any attribute that is used (selected or updated) by rules associated with rel_i, or vice versa. The execution order of rules of such pair of relationship types does not matter to the final result.

Constraint 4.7 $\forall rel_1, rel_2 \in DM\_rel$, *if* $rel_1 \neq rel_2$ *then* $Independent(rel_1, rel_2)$ *holds.*

Constraint 4.7 needs to calculate many attribute sets, and in worst case we need to apply different relationships separately. In fact it is more natural to just define some execution order for them to replace Constraint 4.7.

Constraint 4.8 $(\forall R \in Rule, R : C \rightarrow A : num(A.\textbf{\textit{where}}) \leq 1)$ ***and***
$(\forall rel_1, rel_2 \in DM\_rel :$
$(\forall R_i \in Rule(rel_1), \forall R_j \in Rule(rel_2),$ *if* $rel_1 \neq rel_2$ *then* $Pri(R_i) > Pri(R_j))$ ***or***
$(\forall R_i \in Rule(rel_1), \forall R_j \in Rule(rel_2),$ *if* $rel_1 \neq rel_2$ *then* $Pri(R_i) < Pri(R_j))).$

Constraint 4.8 says that there is an order between relationship types. All rules associated with one type are executed before all the rules of another type, or after all the rules of that other type.

Theorem 4.10 *A rule set Rule is confluent if it satisfies Constraint 4.3, 4.4, 4.5, 4.6, and Constraint 4.7 or 4.8.*

Proof: With Constraint 4.3, 4.4 and 4.5 we know that Rule is guaranteed to terminate. With Constraint 4.6 the propagation through the same relationship graph has an exclusive route; each attribute on the path of this route can only be assigned to once. All actions on this route have fixed execution order. So the final result at the end of the route is always the same. But the value of each attribute may still be uncertain if we allow the propagation through other types of relationship at the same time. With Constraint 4.7, execution of rules through one relationship graph has no influence on the rules using other relationship graphs. Here no influence means rules from different groups cannot activate or deactivate each other and their actions commute. So we conclude that rule set *Rule* is confluent. But Constraint 4.7 still limits the possible triggering: triggering is only allowed among the rules using the same relationship type. Constraint 4.8 replaces Constraint 4.7 by enforcing a pre-defined execution order between different relationship types. The propagation through different relationship graphs has a priority order, so the final result of the UM from the same event for the same user will be always be the same. Hence rule set *Rule* is confluent.

Constraints 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8 are fairly simple constraints, easy to understand for authors, are satisfied by some current AHS (as we will see later) and can be used in future AHS.

4.8.3 Complexity of static analysis

Because the system has to check at least Constraint 4.3 to guarantee reasonable adaptation behavior, and because Constraint 4.4 and Constraint 4.8 on the DM are known before analyzing a set of rules, we neglect checking of Constraints 4.3, 4.4 and 4.8 when considering the complexity of the whole checking algorithm for termination and confluence. The checking time is mainly spent on checking Constraints 4.6 and 4.7. The complexity for Constraint 4.6 is $O(L \times E \times M^2)$ where L is the total number of relationship graphs used

in all rules, E is the maximum number of edges in every relationship graph, and M is the total number of all attributes. The complexity for Constraint 4.7 is $O(N^2 \times M^2)$, N is number of rules and M is the total number of all attributes. So the complexity of this checking algorithm for termination and confluence is $O((N^2 + L \times E) \times M^2)$.

4.9 Summary of AE

We have defined a general purpose adaptation engine. We focused on the design goals of providing powerful expressiveness for writing adaptation strategies, making authoring easy, and providing static analysis for termination and confluence.

To show how an AE works exactly, we designed an abstract adaptation engine. This abstract machine uses AHAM-CA rules as a subset of CA rules from active database theory, because the database CA rules are close to the adaptation requirement of AHS, and CA rules are well-studied in active database theory. For CA rules better static analysis results for termination and confluence are available than for the more general ECA rules.

We have illustrated two widely used static analysis methods for termination and confluence, one is the TR (Triggering Graph) for ECA rules and one is AG (Activation Graph) for CA rules. Considering the special situation of AHS: that we know the DM instance, we have proposed a less conservative static analysis of termination and confluence (less conservative for AHS than for active databases). This analysis has the same time complexity as the Propagation Algorithm [Baralis and Widom, 2000]. The complexity of the static analysis is exponential in general, but for AHS the PA is simple and efficient.

Static analysis of termination and confluence is based on the assumption that the AE randomly selects an active rule to execute. Different systems may uses different conflict resolution methods. For the case where termination and confluence is not guaranteed, static enforcement can be used to force the properties. For example, the AE can associate a priority to every rule, concept, concept relationship and/or attribute, thereby specifying the choice the AE has to make at runtime when two rules become eligible for execution. However, such tricks are masking problems rather than solving them. When the adaptation result depends on a system- or author-defined priority of one rule over another (apart from the priorities used to distinguish the execution *phases*) it becomes difficult for authors to predict the behavior of the adaptive application they are developing. We again prefer a static analysis to predict when *confluence* is no longer guaranteed.

Unfortunately, the sufficient conditions used in the static analysis are too strict: they do not allow rules to trigger each other. We have proposed sufficient conditions for simple adaptive hypermedia systems that guarantee termination and confluence while still allowing rules to trigger each other.

In this chapter we answered research question 4. We have defined behavioral semantics for AHS and have illustrated how AHS work exactly. We have discussed different ways to guarantee termination and confluence for AHS.

Chapter 5

Validation of AHAM: InterBook

This chapter describes InterBook in terms of our reference architecture for adaptive hypermedia applications. InterBook [Brusilovsky et al., 1996b] is an authoring tool to develop electronic textbooks on the World Wide Web. It is a typical AHS that provides adaptive content and adaptive navigation support according to users' features. (However, the content adaptation is very limited.) Section 5.1 introduces the user interface of InterBook by characterizing all its different types of windows and user interaction with those windows. Section 5.2 describes the structure of the domain model of InterBook. Section 5.3 describes the structure of the user model of InterBook. Section 5.4 describes the adaptation of InterBook using AHAM-CA rules. Section 5.5 describes the issues of termination and confluence in the case of InterBook. Section 5.6 summarizes the validation of our reference model by expressing the functionality of InterBook.

5.1 Introduction

An electronic textbook (ET) is a popular form of on-line learning material that replicates classic printed textbooks in digital form. Adaptivity makes the book personalized. Adaptive ET systems can be useful in any situation when the ET is expected to be used by people with different goals and knowledge and where the hyperspace is reasonably large. Knowing users' goals and knowledge, an adaptive ET can support users in their navigation by limiting the browsing space, suggesting the most relevant links to follow, or by providing adaptive annotations to visible links. Users with different goals and knowledge may be interested in different pieces of information presented on a hypermedia page and may use different kinds of links for navigation.

InterBook, designed by Peter Brusilovsky, is an authoring tool to develop adaptive electronic textbooks on the World Wide Web. It builds on the ELM-ART system [Brusilovsky et al., 1996a] which was used for an interactive Lisp course. We studied InterBook's production version, which is available on-line and very stable. There is also a development version with richer functionality, which we do not discuss in this dissertation. The limited production version is sufficient to show how to describe InterBook's adaptation functional-

ity in AHAM. It should not be difficult to extend our description to the richer development version (but we have not tried this).

InterBook is a typical AHS that consists of a domain model, a user model, and an adaptation model. It has only built-in adaptation rules. There is no adaptation language for authors to use to describe adaptation rules. Authors only define the domain model which consists of a set of concepts and concept relationships. The system maintains a user model for each individual user and uses the domain model and user model to provide adaptation. Before we describe these models of InterBook, we first introduce the interface of InterBook.

InterBook uses multiple windows and multiple frames within windows. There are various types of windows in InterBook: *registration window*, *section window*, *domain-concept window*, *content window*, *glossary window*, *search window*, and *help window*. We concentrate on the section window, domain-concept window, content window, glossary window, and help window, because InterBook provides adaptation in these windows and adaptation is what we are most interested in. We use the term *section* for *Textbook Section (TS)* in the remainder of this chapter.

Figure 5.1: A screen shot of the section window (1)

When the user "clicks" on a *section link* a *section window* appears as shown in Figure 5.1 or Figure 5.2. The *section window* is the most important window in InterBook's interface.

This window is designed to view the textbook section by section. A *section window* has three frames: *text, navigation bar*, and *tool box*. Several textbooks can be viewed at the same time; each textbook will be shown in a separate window. All textbooks on the same bookshelf share a common domain-concept structure and a common user model. As a result, in our description we use a single user model for each user to store the user's knowledge of all domain-concepts about all textbooks in the bookshelf.

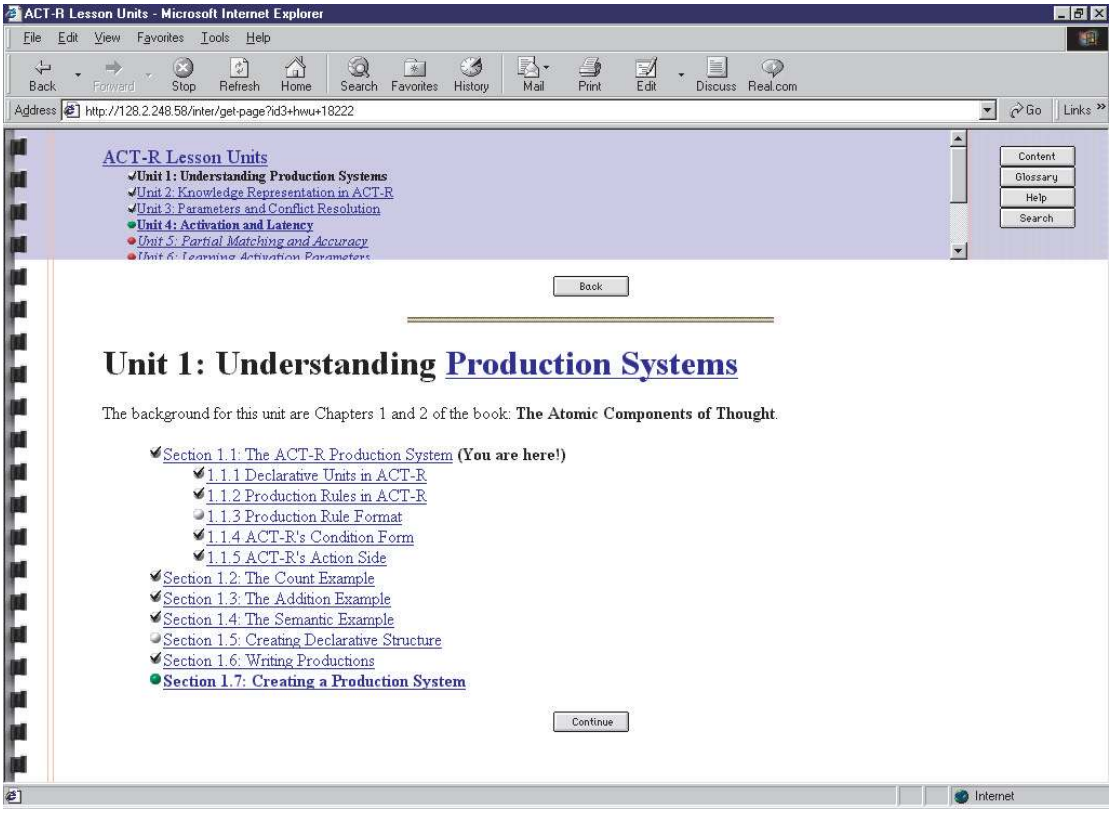

Figure 5.2: A screen shot of the section window (2)

- The *text* frame is the lower sub-window of a section window. This frame shows a particular section of the textbook that is called the current section. It can be a *terminal section* or a *high-level section* (the most high-level section is the book itself). For a terminal section (see Figure 5.1) the *text* frame shows the "ready bar", the title of the section, and then possibly some pictures, text and domain-concept links. Note that "domain-concept" is a term used in InterBook to represent a piece of knowledge about a textbook. A domain-concept is an *abstract concept* in terms of AHAM. In the text frame, domain-concept links are treated normally as a part of text; these domain-concept links are not adaptive (not annotated adaptively) in InterBook. (They can be annotated in InterBook but this feature was disabled in the production version we describe.) For a high-level section (see Figure 5.2) the *text*

frame shows the "ready bar", the title of the section, an introduction to the section (if that exists) and a list of subsection links. It provides content adaptation and link adaptation. If one of its subsections is the last section visited and this subsection is not a terminal section, the system shows "(You are here!)" and shows hierarchical subsection links for this subsection. The system also shows "(You were here!)" for the link to the section that was visited before last. Technically speaking these notes are a form of content adaptation. However, their function is to annotate the links to sections. You can see these annotations also in Figure 5.4 which is a content window, not a section window.

Another form of content adaptation is the change in color of the "ready bar". The green bar indicates that the user is ready to read this page. For "real" link adaptation InterBook uses colored bullets as we explain below.

- The *navigation bar* frame is the upper left sub-window of the section window. This frame shows the position of the current section in the textbook: it lists the titles of all direct ancestor links (father, grandfather, etc.) and all sibling links (links at the same level) of the current section. It also shows the current section title, with annotation, but this is not a link. The system represents links using the same color metaphor as used for adaptive annotation to show the educational status for the sections. The *navigation bar* frame serves for both orientation and navigation.

- The *tool box* frame is the upper right sub-window of a section window. It provides a set of buttons that are used to call additional windows (tools). It does not provide adaptation.

InterBook distinguishes two "read" states of a section. It uses a *check mark* to show that the user has read this section or not. InterBook has a feature to show a *concept bar* in which links to concepts are shown with check marks of different sizes. This feature was disabled in the production version we describe.

InterBook distinguishes three "ready" states of a section. It uses the link annotation technique to show the relevance of links to sections, which is a combination of color and font. The color "green" and font "bold" means the section is "recommended" to be read. The color "red" and font "italic" means the section is "not recommended". The color "white" and font "normal" means the section is "not interesting" (because its outcome concepts are already known). The adaptation for section links is the same in all the windows. The current section is always shown in "black" and with a "bold" font. It is not a link.

When the user "clicks" on a *domain-concept link* a *domain-concept window* is opened as shown in Figure 5.3. A domain-concept window has three parts: a *description frame* describes the domain-concept itself; an *introduction frame* contains a list of links to sections that introduce this domain-concept; and a *requirement frame* gives a list of links to sections for which knowledge of this domain-concept is required. The section links are presented in the same way as in the section window.

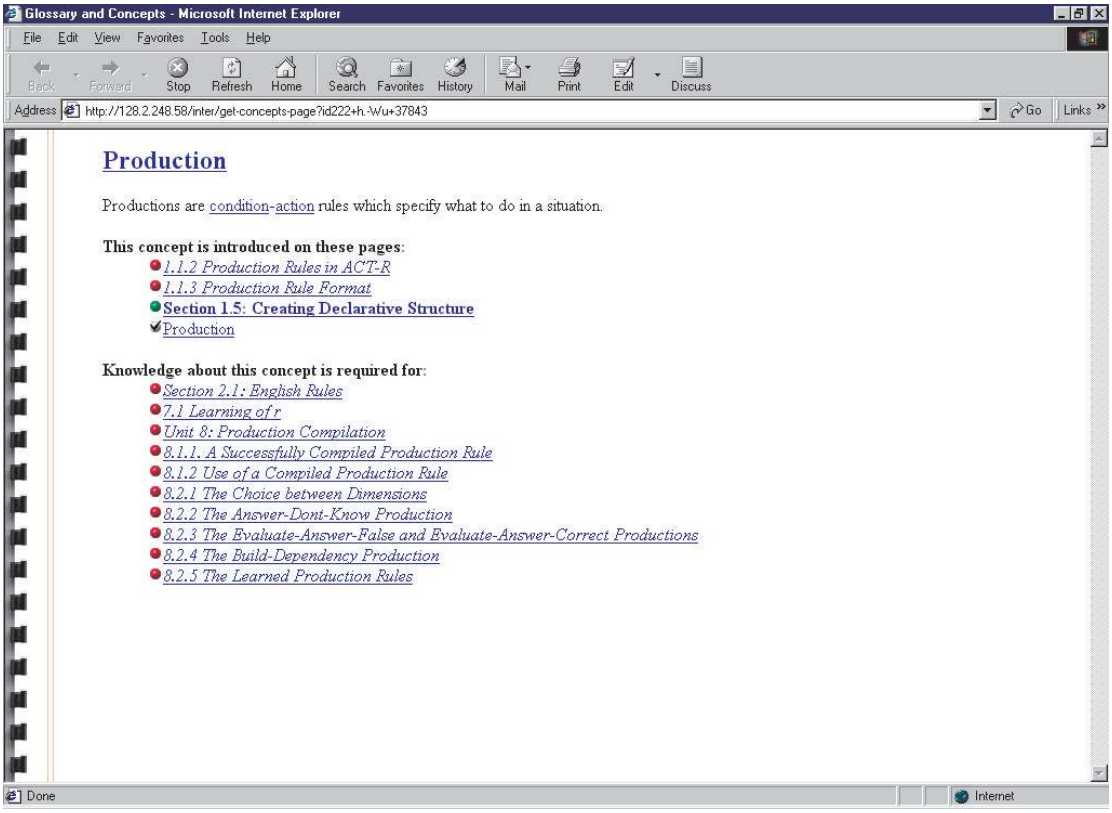

Figure 5.3: A screen shot of the domain-concept window

A "click" on the *content button* in the tool box frame of the section window creates a *content window* as shown in Figure 5.4. The *content window* shows two tables of content (in two frames): The *bookshelf* frame shows the list of book links on the bookshelf of this site; the *index* frame shows links to the sections of the current book. It also shows recently visited subsections and adds the "(You are here!)" and "(You were here!)" comments. The system represents links using the same link annotation techniques as in other windows.

A "click" on the *glossary button* in the section window creates a *glossary window* as shown in Figure 5.5. The *glossary window* has three frames: *alphabet*, *glossary*, and *content*. The *alphabet* frame shows a list of 26 letters as an index for the glossary. A "click" on a letter causes the system to update the *glossary* frame. There is no adaptation in the *alphabet* frame. The *glossary* frame shows a list of domain-concept links. The information in this frame is adaptively annotated. It only shows domain-concept links that have the selected letter as the first character in their name. InterBook does not need "read" states for the domain-concepts (unlike for the sections), because the domain concepts are used to represent a piece of knowledge about the subject domain, not to represent the content of the textbook. Each "click" on a domain-concept link in the glossary frame updates the content frame. The *content* frame shows the same information as the *domain-concept window*. (In fact, the *domain-concept window* **is** a glossary window with the alphabet and

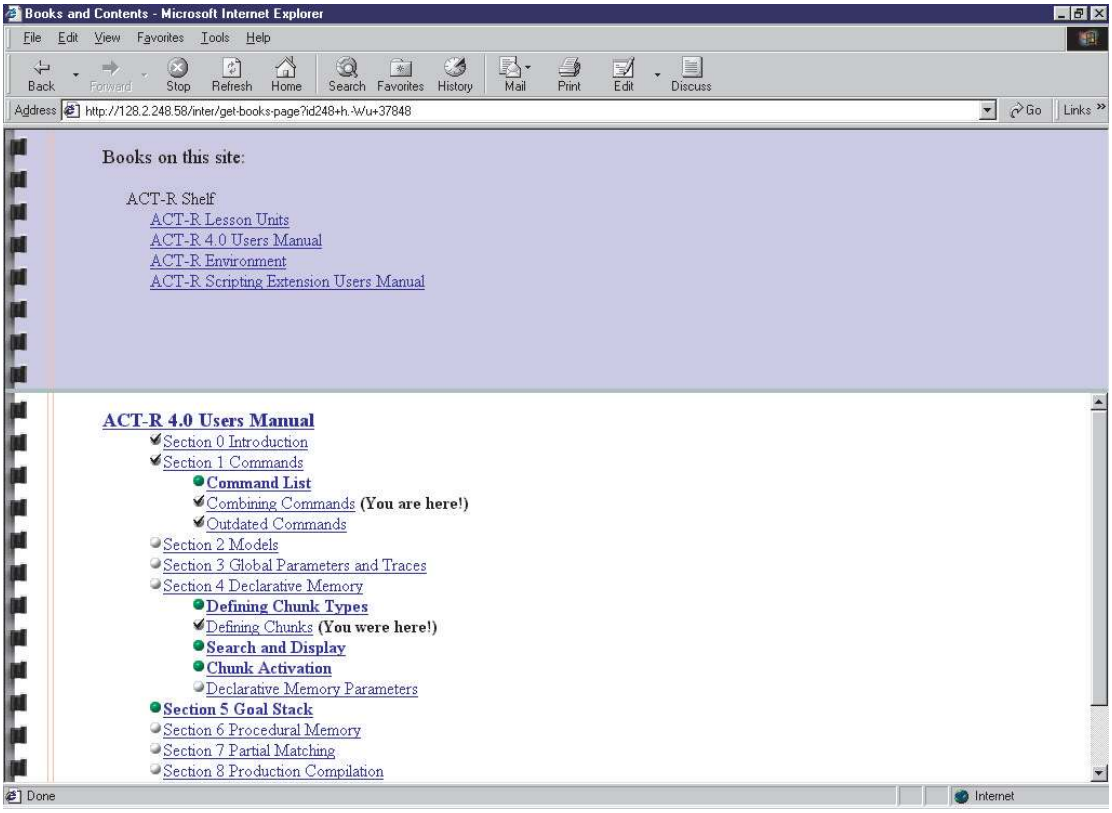

Figure 5.4: A screen shot of the content window

glossary frames disabled.) The information in the content frame is adaptive. The frame shows the domain-concepts that are selected in the glossary frame. It uses the same link annotation techniques as in the *domain-concept window*.

InterBook offers a *help window* to show to users which pages to read to prepare themselves to read a section. The system knows the structure of prerequisites and outcome domain-concepts. When a student enters a section page that is not yet ready to be learned, he or she can "click" on the help button in the tool box frame. The system then opens a *help window* as shown in Figure 5.6. A *help window* shows an explanation of the problem the student may have, and the links to sections that explain a domain-concept. The section links are adapted using link annotation techniques.

We concentrate on the various types of links described above from an adaptation point of view. There are also other kinds of links that are not relevant for the discussion of adaptation in InterBook (because these links have no adaptation associated with them). We will not discuss these links in this dissertation.

- *test* – pop up a solution feedback window and pop up an information window.
- *back* – go to previous text window (without using the browser's back button).

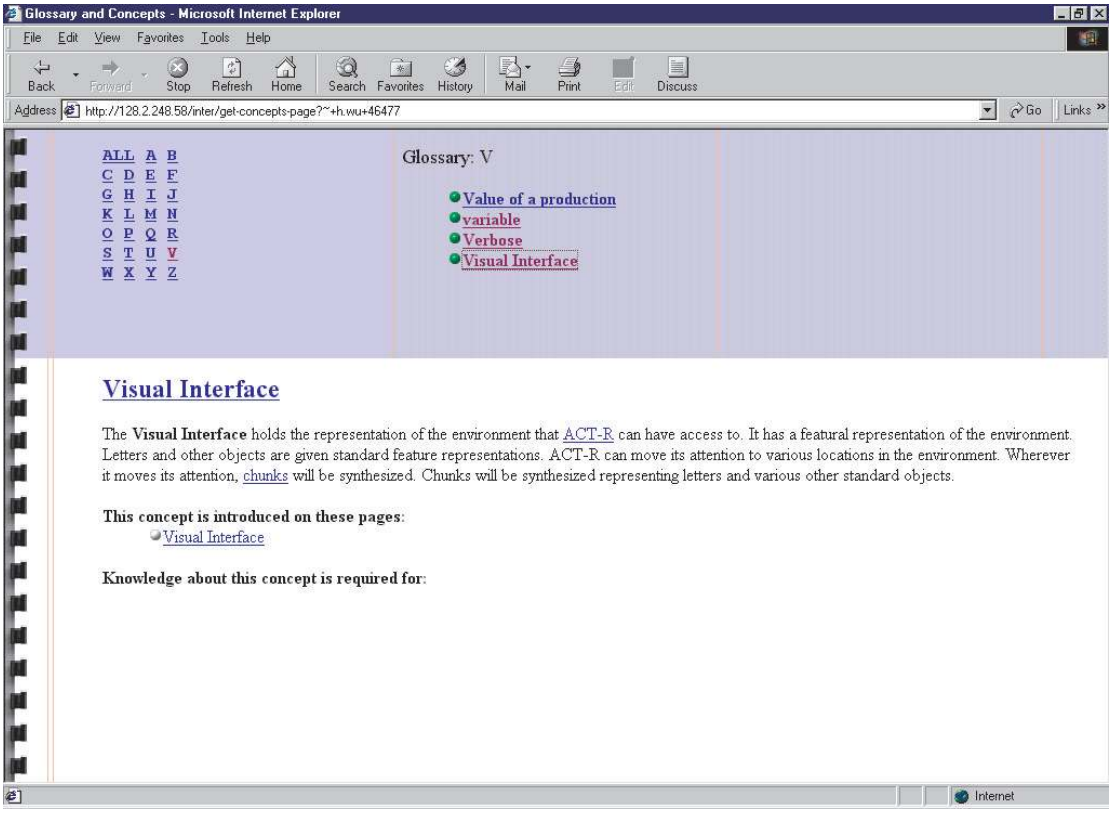

Figure 5.5: A screen shot of the glossary window

- *continue* – go to next text window.
- *search* – pop up a window with a search form.

5.2 InterBook: Domain Model

As we described above, InterBook uses multiple windows and frames. The author does not specify this rich functionality. An author only defines sections, domain-concepts, a hierarchical relationship between sections, and a structure of outcome and requirement relationships between sections and domain-concepts. It is the system that generates all windows and frames. AHAM focuses on describing adaptation functionality based on a static domain model. In our description of InterBook we assume all windows and pages have been designed, authored or generated. We concentrate on how to update the user model and how to generate the adaptation in the windows and frames. Each type of window in InterBook corresponds to a set of pages (either authored or generated from different books, sections, and domain-concepts). Each page consists of a set of fragments. Some fragments are always included and some are conditionally included, based on the

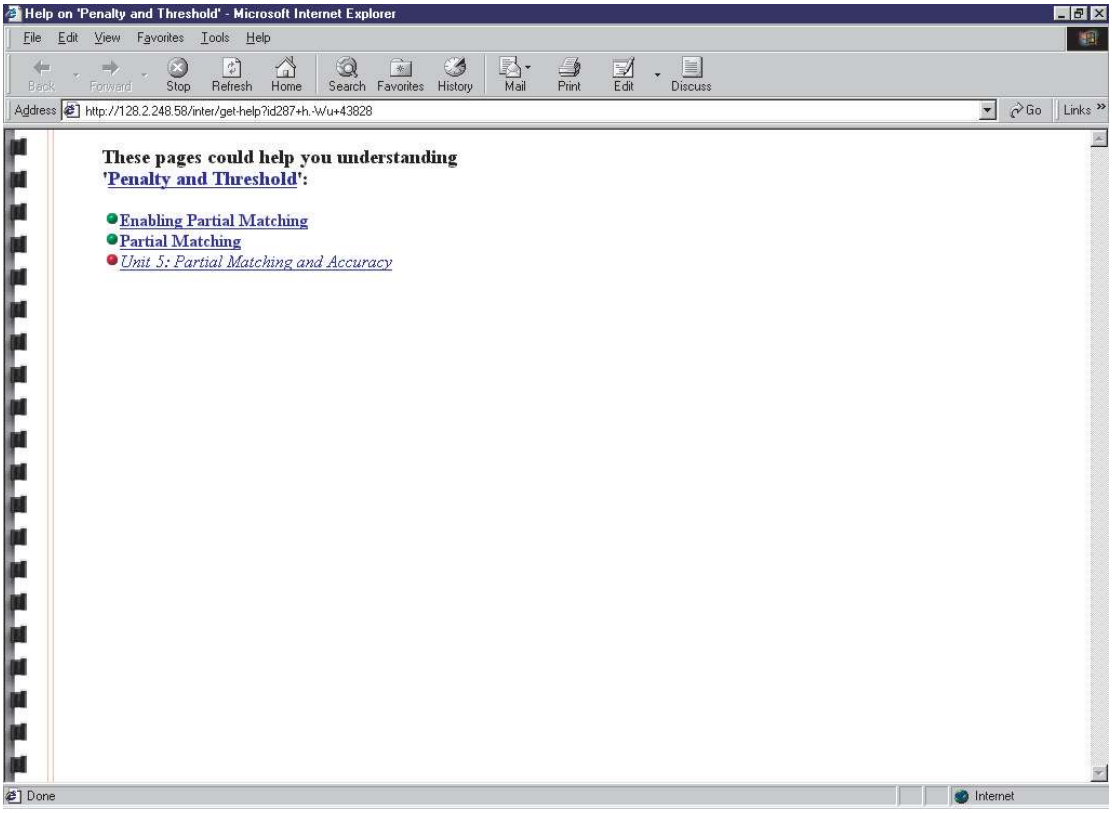

Figure 5.6: A screen shot of the help window

user model instance. Some link fragments are adaptively annotated and some are not. We only describe adaptation rules for things that depend on the user model.

Like traditional printed textbooks InterBook uses hierarchically structured presentations of a subject. Units at different levels are called sections in InterBook. The highest level unit is the electronic textbook. A book is divided into sections. Each section is structured further into smaller units (subsections). A *terminal section* presents text, links and maybe also pictures. A *high level section* usually provides some introduction or preface that introduces the reader into its subsections. Both terminal sections and high-level sections have a physical page to present it.

InterBook uses a *domain-concept* to describe a piece of knowledge about the subject domain. Depending on the application area, domain-concepts represent smaller or larger pieces of domain knowledge. A section is usually centered on a specific subset of the *domain-concepts*.

In addition to the main content of the textbook, InterBook provides navigation aids just as good textbooks do. Examples are the *table of contents*, the *index of the glossary* and the *glossary* itself. The *table of contents* contains a hierarchy of sections of a textbook. Each section is presented as a link to its physical page. The *index* contains the first letter of the names of the most important terms and keywords of the book. Each term and keyword

links to a physical page. The *glossary* contains a more extensive description for a number of important domain-concepts that link to a detailed explanation of the domain-concept and where the domain-concept is presented. All sections are indexed by domain-concepts that appear in the glossary.

5.2.1 Concepts in the DM-InterBook

The domain model describes the information domain at a conceptual level. In AHAM it consists of a set of concepts and concept relationships. Suppose we use DM-InterBook to represent the domain model of InterBook. The term concept used in AHAM is a general term, it can represent low level fragments, pages and high level abstract concepts. The DM-InterBook has several kinds of concepts. At the lowest level there are *fragments* which are pieces of text, pictures, links which may link to other windows (pages), e. g. a section, the content of a book, a domain-concept, an external source, etc. Above fragments there are *pages* that can be distinguished by the types of windows described below. To simplify the page description, we introduce *frames* to group fragments in the page according to different use. We do not actually use frames in our rules to describe InterBook. At the highest level there are *concepts* of which a user can have knowledge (stored in the user model).

Section windows (SW)

We assume each section has a corresponding section window. A section window is generated when a user "clicks" on a section link. A section window has three frames, a *navigation bar frame, text frame, and tool box frame*:

- The *navigation bar frame* consists of section link fragments. These sections can be chapters, sections, or subsections.
- The *text frame* consists of fragments. The first fragment is a colored "ready bar" presented at the top of the frame. Below the colored bar is a fragment to describe the content of this section. The other fragments can be either links to subsections if the current section is not a terminal section, or can be text and links to other sections or domain concepts (or to external sources) if the current section is a terminal section.
- The *tool box frame* contains a content button linking to a content window and a glossary button linking to a glossary window.

Domain-concept windows (DCW)

We assume that each domain-concept has a corresponding domain-concept window. A domain-concept window is generated when the user "clicks" on a domain concept link in the text frame of a section window. A domain-concept window consists of three frames, a *description frame, outcome frame and requirement frame*:

- The *description frame* is a text fragment to explain this domain-concept.
- The *outcome frame* consists of link fragments linking to terminal sections that provide knowledge on the current domain-concept.
- The *requirement frame* consists of link fragments linking to sections for which the knowledge about this domain-concept is required.

Content windows (CW)

We assume each textbook has a corresponding content window. A content window is generated when the user "clicks" on the Content button in the section window. A content window consists of two frames, *bookshelf frame and index frame*:

- The *bookshelf frame* consists of link fragments pointing to electronic books stored on this site.
- The *index frame* consists of link fragments. Each fragment is a section link that points to one section of the current textbook. Links to subsections are also shown for the sections containing the two most recently visited subsections.

Glossary windows (GW)

We assume each domain-concept has a corresponding glossary window. A glossary window is generated when the user "clicks" on the Glossary button in the section window. It consists of three frames, an *alphabet frame, glossary frame, and content frame*. Each frame is generated by a different event.

- The *alphabet frame* consists of 26 link fragments labelled with the letters 'A' to 'Z' and an "All" link fragment. Each letter links to a list of domain-concept links which have this letter as the first letter of their names. The "All" link fragment links to a list of all domain-concept links.
- The *glossary frame* consists of fragments. The first fragment contains the text "Glossary" which is presented when the user "clicks" on a glossary button. The second fragment is the selected link fragment in the alphabet frame which is presented when the user "clicks" on a link fragment in the alphabet frame. A list of domain-concept link fragments is presented below the first and the second fragment. When the user "clicks" on an alphabet link, the list contains domain-concepts having the selected letter as the first letter of their names. When the user "clicks" on the "All" link, the list contains all domain-concepts.
- The *concept frame* consists of a set of fragments. The first fragment explains this domain-concept. It is followed by fragments that are links to sections providing knowledge about this domain-concept. Below these are fragments that are links to sections that require knowledge of this concept. The content frame contains the same

information as the domain concept window. The content of this frame is generated by a "click" on a domain-concept link in the glossary frame. It is empty when users just "click" on the glossary button.

Help windows (HW)

We assume each section has a corresponding help window. A help window consists of two frames: a *description frame* and a *suggestion frame*:

- The *description frame* is a fragment to describe the section page for which the user requests help.
- The *suggestion frame* consists of link fragments linking to sections that provide required (prerequisite) knowledge.

5.2.2 Concept relationships in the DM-InterBook

Concepts in the DM-InterBook are related to each other in a certain way. InterBook requires authors to define the following relationships:

- *Firstpage*$(Book, P)$ describes that page P is the first page of the $Book$.
- *Subsection*(TS_1, TS_2) describes the hierarchical structure of the textbook, e. g. section TS_1 has a subsection TS_2. The $Book$ itself is the top of this hierarchy.
- *Domain_concept*$(Bookshelf, C)$ describes that C is a domain-concept of the *Bookshelf*. The *Bookshelf* is a predefined entity (concept component). We use it to get access to all the domain-concepts used in the *Bookshelf*.
- *Bookset*$(Bookshelf, Book)$ describes that the $Book$ is on the $Bookshelf$.
- *Loginpage*$(Bookshelf, CW)$ describes that CW is the the first page to visit after a user registers or logs in to the system. The first page of the system is a content window (CW) which only contains a *Bookshelf*, the index frame of the CW is empty.
- *Requirement*(TS, DC) describes that the knowledge of domain-concept DC is required (prerequisite) for the section TS.
- *Outcome*(TS, DC) describes that domain-concept DC is an outcome of section TS. (Reading TS generates knowledge about DC. A section may have several outcome concepts.)
- *Fragment*(P, F) describes that F is a fragment of section P. Here P represents a window (page), e. g. SW, DCW, CW, GW, and HW.

To simplify the rule description for InterBook, we introduce the following relationships that can be deduced from above relationships.

- *Descendant*$(Book, TS)$ describes that section TS is a descendant of the $Book$, TS is a direct or indirect subsection of the $Book$. This can be derived from *Subsection* relationship.

- *Descendant*$^{-1}(TS, Book)$ is the inverse relationship of *Descendant(Book, TS)*.

- *Requirement*$^{-1}(DC, TS)$ is the inverse relationship of *Requirement*. It describes that section TS requires the knowledge of the domain-concept DC.

- *Outcome*$^{-1}(DC, TS)$ is the inverse relationship of *Outcome*. It describes that DC is introduced in TS.

- *Subsection*$^{-1}(TS_2, TS_1)$ is the inverse relationship of *Subsection*.

- *GAlphabet*$(Letter, DC)$ describes that domain concept DC has *Letter* as the first character in its name.

These relationships are binary tuples. They can only be applied from the first parameter to the second parameter. We introduced inverse relationships to circumvent this restriction. The restriction is not necessary for AHAM, but we use it on AHAM-CA rules in order to control termination and confluence.

5.3 InterBook: User Model

For every user InterBook creates and maintains an individual model. The UM of InterBook, called UM-InterBook, is an overlay model of the DM-InterBook. For each concept in the DM-InterBook, a set of attribute values are stored in the UM-InterBook, e. g. it can store independently the user's knowledge of different topics. The DM-InterBook may have many kind of concepts, but we focus on *domain-concept*, *section*, *glossary*, *content*, *help*, and *(alphabet) letter*, which we described in the previous section.

We can use attribute-value pairs to describe all the user features. For each concept in DM, several attribute values are stored in the user model. Also the other user features can be stored in the UM. We distinguish four types of attributes used in the UM of InterBook:

- user features (including knowledge)

- system related features, e. g. to remember the history or temporary results

- the adaptation attributes (of presentation specifications) used in the run time layer

- (simulations of) external events

5.3.1 Attributes for users' features

InterBook stores and/or uses a number of attributes about concepts and pages that represent what the user "did" about these concepts and pages.

- For each domain-concept, a *knowledge* attribute stores a value that is an estimate of the student's knowledge level of this domain-concept. InterBook distinguishes four knowledge states: "new", "known", "well known", and "learned". We use a numeric representation of these states: 0 for "new", 1 for "known", 2 for "well known", and 3 for "learned". We will see how these different knowledge states are used in adaptation rules in Section 5.4. We assume that the data type of knowledge is *real* because there are adaptation rules that increase the knowledge by 0.1 under certain conditions.

- For each domain-concept, a *test* attribute stores the result of a test performed by the user. If the user passes the test the function returns **true**, otherwise it returns **false**.

- For each section a *ready* attribute stores the state of the section: whether it is ready to be read or not. This attribute could be calculated on the fly instead of storing it in the user model. We chose to make it persistent in our representation because it simplifies the adaptation model. The data type for this attribute is enumerated: "recommended", "not recommended", "not interesting".

- For each section a *read* attribute represents whether the section has been accessed by the user. Its value can be **true** or **false**.

5.3.2 Attributes for system related features

Temporary results are generated during the rule execution. They may be used later during the same transaction and are preserved for use during the next transaction. They can be (re-)calculated on the fly, or stored in the UM. We chose the latter in our representation of InterBook because it greatly simplifies the adaptation model.

- For each domain-concept a *page* attribute stores the (uid of the) first page that made the knowledge of this domain-concept become "known". The data type is string. Initially the value is **null**. This attribute is used to avoid updating the knowledge a second time when a user revisits this page.

- For each book a *last* attribute remembers the section last visited. Initially it is **null**. This is used to generate the "(You are here!)" annotation.

- For each book a *before_last* attribute remembers the section visited before the *last* section. Initially it is **null**. This is used to generate the "(You were here!)" annotation.

5.3.3 Attributes for presentation specifications

The adaptation attributes used for presentation specifications store the adaptation results. These will be used by the Run-time Layer according to the definition of AHAM.

For each section several attributes are used to indicate different features:

- An *indicator* attribute stores a (color) value for the bullet in front of the section link fragments. The same indicator is also used for the color of the "ready bar" in the text frame of a section window. InterBook uses the link annotation technique to indicate the different relevance levels of these links (for the current user). The relevance is represented by the colors "green", "red" and "white" for the states "recommended", "not recommended", and "not interesting".

- A *font* attribute stores a value for the text of a link fragment. It can be "bold", "italic" or "normal" to indicate that this link leads to a section with state "recommended", "not recommended", and "not interesting". It is used together with the indicator (bullet) in InterBook to indicate the relevance of section links.

- A *pres* attribute stores a value for the presentation specification of a section link fragment. It can be "show" or "hide".

- A *link* attribute stores a value to indicate whether to hide this link or not. When a link is hidden the text of the link anchor is shown but it is not a link. The value can be "show" or "hide". When a link is hidden the anchor will appear in black, whereas a link that is shown appears in blue (or purple but that is determined by the browser).

- A *mark* attribute stores a value for the check mark in front of the link fragment. It can be "show" or "hide" to indicate whether to show a mark or not.

- A *fa* attribute stores a value to indicate whether to show the text fragment "(You are here!)" after the link section fragment. It serves as a reminder of the current reading place. The value can be "show" or "hide".

- A *fw* attribute stores a value to indicate whether to show the text fragment "(You were here!)" after the link fragment. It serves as a reminder of the previous reading place. The value can be "show" or "hide".

For domain-concepts some attributes are used as follows:

- An *indicator* attribute stores a value for the bullet in front of the domain-concept link fragments (which appear in a glossary window). InterBook uses the annotation technique to indicate the relevance of these links to the user. The value can be "red", "green" or "white" depending on the knowledge state (for the concept): "red" for smaller than 1, "green" for between 1 and 2, and "white" for bigger than 2. Note that the green-red-white color metaphor used for domain-concepts is used with a different meaning than for links to sections.

- A *pres* attribute stores a value for the presentation specification of the domain-concept link fragment (in the glossary window). The value can be "show" or "hide".
- A *font* attribute stores a value for the text of a link to a domain-concept. It can be "italic", "bold" or "normal" to indicate that this link leads to a domain-concept with knowledge values smaller than 1, between 1 and 2, or bigger than 2. It is used together with an indicator to indicate the relevance of domain-concept links.

5.3.4 Attributes for simulating events

We use "pseudo" user model attributes to represent that an event has happened. We use three such events: for accessing a page, for logging onto the system and for registering as a new user.

- We use a Boolean *access* attribute to represent the event of accessing a page (which is done by "clicking" on a link anchor). For example, $P.access$ represents that "clicking" on a section (window) link; $GW.access$ represent "clicking" on a glossary (window) link; $C.access$ represents "clicking" on a domain-concept (window) link; $Letter.access$ represents "clicking" on a letter in the alphabet frame of the glossary window; $Book.access$ represents "clicking" on a book link the $Book$ on the $Bookshelf$ (in the content window).
- We use a Boolean *new_access* attribute to represent the special event of "registering" with the system as a new user.
- We use a Boolean *old_access* attribute represents the special event of "registering" with the system as an existing user.

5.4 InterBook: Adaptation Model

The adaptation model describes how to update the user model and how to generate the adaptation based on that user model and the domain model. We use AHAM-CA rules to describe the adaptation model of InterBook, called AM-InterBook. As we described in Chapter 3 the adaptation rules are divided into four execution phases: IU, UU-pre, GA, and UU-post. These four phases are executed in a linear order. In each phase there are several transitions, where each transition is the execution of one active rule.

5.4.1 Adaptation rules in the IU phase

InterBook allows the creation of a bookshelf containing different electronic textbooks. We will take the existence of the bookshelf into account (using the name *Bookshelf*, for instance when initializing new_access values and old_access values of books). Apart from that, we only deal with the "current" book. Each page belongs to a book; each book has a last visited page and a before-last visited page. We treat the action of registering (or logging

in) and selecting a book to study as different events. Registering will give access to the *Bookshelf*. The user must then choose a book from the *Bookshelf* to study.

When a user logs in on the system, the system reacts differently depending on whether the user is a "new" or existing ("old") user. If the user is new, the system creates the UM, initializes all attributes with default values, and generates a first page (CW) of the system to show to the user. If the user is old, then system retrieves the existing user's UM and generates the same first page (CW) for the user. Assume we use the *Bookshelf* concept to represent the registration window of the system. The following rules set the default values for the registration. Section variable P represents a section (window) link. We write the initialization rules for InterBook as follows:

Rules 1 to Rule 6 are generic rules for initializing the user model when a user registers with the system for the first time. These rules set default values for each section P of each book on the *Bookshelf*.

The rules correctly initialize the user model only under the following assumptions:

- Every page of a book contributes (knowledge) to at least one outcome concept. (Otherwise InterBook will consider the page not interesting and users are likely to then never read that page.)
- The first page of a book does not have prerequisites. Otherwise the page with which a user starts will not be recommended.
- The first page of a book contributes knowledge to an "initial" concept that is a prerequisite for all other pages. As a result, when the user registers and accesses the first page the "initial" concept becomes known, and that knowledge change will make pages become "recommended" that have no other prerequisites.

Rule 1:
C: **select** *Bookshelf.new_access*
A: **update** *Book.new_access* := **true**; *Book.access* := **false**
where *Bookset*(*Bookshelf*, *Book*)

Rule 2:
C: **select** *Bookshelf.new_access*
A: **update** *CW.access* := **true**
where *Loginpage*(*Bookshelf*, *CW*)

Rule 3 is a generic rule to set the knowledge of all concepts to 0 when the user registers as a new user.

Rule 3:
C: **select** *Bookshelf.new_access*
A: **update** *C.knowledge* := 0
where *Domain_concept*(*Bookshelf*, *C*)

Rule 4:
C: **select** *Book.new_access*
A: **update** *P.ready* = "recommended"; *P.fa* := "hide";
P.fw := "hide"; *P.read* := **false**;
P.mark := "hide"; *P.link* := "show"
where *Firstpage*(*Book*, *P*)

Rule 5:
C: **select** *Book.new_access*
A: **update** *P.ready* = "not recommended"; *P.fa* := "hide";
P.fw := "hide"; *P.read* := **false**;
P.mark := "hide"; *P.link* := "show"
where *Descendant*(*Book*, *P*) **and not** *Firstpage*(*Book*, *P*)

Rule 6 is a generic rule that initializes the *last* and *before_last* attributes of each book on the *Bookshelf*.

Rule 6:
C: **select** *Book.new_access*
A: **update** *Book.before_last* := **null**;
Book.last := **null**

Rules 7–12 handle events that can occur more than once. Therefore we verify that the "event" attribute has become **true**. (We will also have to reset the value to **false** at the end of the rule processing.)

Rule 7:
C: **select** *Bookshelf.old_access*
where *Bookshelf.old_access* = **true**
A: **update** *Book.old_access* := **true**; *Book.access* := **false**
where *Bookset*(*Bookshelf*, *Book*)

Rule 8:
C: **select** *Bookshelf.old_access*
where *Bookshelf.old_access* = **true**
A: **update** *CW.access* := **true**
where *Loginpage*(*Bookshelf*, *CW*)

Rule 9–10 describe when a user accesses another book on the *Bookshelf*, the system shows the first page of that book, and hide the others.

Rule 9:
C: **select** *Book.access*
where *Book.access* = **true**
A: **update** *P.access* := **true**
where *Descendant*(*Book*, *P*) **and** *Firstpage*(*Book*, *P*)

Rule 10:
C: **select** *Book.access*
where $Book.access = \textbf{true}$
A: **update** $P.access := \textbf{false}$
where $Descendant(Book, P)$ **and not** $Firstpage(Book, P)$

Rule 11 sets the *pres* adaptation attribute for all domain-concepts to "hide" for the glossary window when a user *access* a glossary window.

Rule 11:
C: **select** *GW.access*
where $GW.access = \textbf{true}$
A: **update** $C.pres :=$ "hide"
where $Domain\_concept(Bookshelf, C)$

Rule 12 is a generic rule to set the *access* attribute to **false** for all Letters (and "All") in the alphabet for the glossary window when a user *access* a glossary window.

Rule 12:
C: **select** *GW.access*
where $GW.access = \textbf{true}$
A: **update** $Letter.access := \textbf{false}$
where $Letter \in \{A, \ldots, Z, All\}$

This is not exactly AHAM-CA rule syntax, but simply represents 27 specific adaptation rules, one for each letter and one for *All*.

5.4.2 Adaptation rules in the UU-pre phase

The following rules describe how InterBook updates the user model before generating the values of adaptation attributes of the presentation specifications.

Rules 1 through 4 define how the knowledge of concepts changes (increases) when the user accesses section pages or performs (and passes) a test. Rule 1 says that if a user reads a "recommended" section page P, the knowledge of all domain-concepts C that are outcome concepts of P become 1 ("known") if P is the first "recommended" section page that has been visited for domain-concept C.

Rule 1:
C: **select** *P.access*
where $P.access = \textbf{true}$
A: **update** $C.knowledge := 1$; $C.page := P$
where $Outcome(P, C)$ **and** $P.ready =$ "recommended" **and** $C.page = \textbf{null}$

Rule 2 says that if a user reads a "not recommended" section page P, the knowledge of all domain-concepts C that are outcome concepts of P is increased by 0.1 if *C.knowledge* is smaller than 1.

Rule 2:
C: **select** *P.access*
where $P.access = \textbf{true}$
A: **update** $C.knowledge := C.knowledge + 0.1$
where $Outcome(P, C)$ **and** *P.ready* =“not recommended” **and** $C.knowledge < 1$

Rule 3 says that if a user reads a second (different) “recommended” section page of the outcome domain-concept C, then *C.knowledge* becomes 2 (“well known”). Note that *C.page* does not change because it keeps track of the *first* recommended page the user read for concept C.

Rule 3:
C: **select** *P.access*
where $P.access = \textbf{true}$
A: **update** $C.knowledge := 2$
where $Outcome(P, C)$ **and** *P.ready* = “recommended”
and $C.knowledge = 1$ **and** $C.page \neq \textbf{null}$ **and** $C.page \neq P$

Rule 4 says that if a user accesses a section page, then the system believes that the user read the page.

Rule 4:
C: **select** *P.access*
where $P.access = \textbf{true}$
A: **update** $P.read := \textbf{true}$

Rule 5 says that if a user passes a test for domain-concept C (*C.test* is **true**), then *C.knowledge* becomes 3 (“learned”).

Rule 5:
C: **select** *C.test*
where $C.test = \textbf{true}$
A: **update** $C.knowledge := 3$

Rules 6 through 8 define how the *ready* status of section pages changes depending on the knowledge of prerequisite and outcome concepts. Rule 6 says that if all the “prerequisites” for page P are satisfied, meaning that the knowledge for these prerequisite concepts is at least 1, and if P contributes knowledge to at least one still unknown outcome domain-concept, meaning the knowledge of that outcome concept is less than one, then P is “recommended” to be read.

Rule 6:
C: **select** *C.knowledge*
A: **update** *P.ready* := “recommended”
where $Requirement^{-1}(C, P)$ **and** $Outcome(P, C_1)$ **and** $C_1.knowledge < 1$ **and**
not exists (**select** C_2
where $Requirement(P, C_2)$ **and** $C_2.knowledge < 1$)

It is also possible that all the "prerequisites" for page P become satisfied when P can only contribute knowledge to an outcome concept for which the knowledge is already 1 (but not yet 2). This means that the page must not be the "first recommended page" for this outcome concept. Rule 7 sets the recommended state in this case.

Rule 7:
C: **select** *C.knowledge*
A: **update** *P.ready* := "recommended"
where $Requirement^{-1}(C, P)$ **and** $Outcome(P, C_1)$ **and** $C_1.knowledge < 2$ **and**
$C_1.page \neq P$ **and**
not exists (**select** C_2
where $Requirement(P, C_2)$ **and** $C_2.knowledge < 1$)

Note that we could have combined rules 6 and 7 into one rule but we kept them separate to make them easier to read and understand.

Rule 8 says that if reading a recommended page P does not contribute any knowledge to any outcome concept of P then it becomes "not interesting". This can happen when there is no outcome concept that is still unknown or which has knowledge already 1 but when that knowledge was obtained by reading page P.

Rule 8:
C: **select** *C.knowledge*
A: **update** *P.ready* :="not interesting"
where $Outcome^{-1}(C, P)$ **and**
not exists (**select** C_1
where $Outcome(P, C_1)$ **and**
($C_1.knowledge < 1$ **or** ($C_1.knowledge = 1$ **and** $C_1.page \neq P$)))

Note that unlike with rules 6 and 7 we cannot split this rule in the same way to make it easier to read.

5.4.3 Adaptation rules in the GA phase

In this section we describe how to generate the adaptation. InterBook does not provide content adaptation, or *adaptive presentation*, in the sense of Chapter 2. Such adaptation would normally be expected in the text frame of a section window, and would provide additional, prerequisite or comparative explanations. A common reason for not performing content adaptation is that the students may not like it when the same page can look different each time it is visited. However, there are cases where InterBook provides some conditional text (like "(You are here!)" or "(You were here!)"). So in the rules below we will encounter conditional inclusion of fragments in a number of places even though this does not represent the typical use of adaptive presentation mentioned above.

To support users navigating through the course, InterBook provides adaptive navigation support through the adaptive link annotation technique. Adaptive link annotation

means that the system uses visual cues (icons, fonts, colors) to show the type and the educational state of each link. InterBook distinguishes three educational states for each page of material: "recommended", "not recommended", and "not interesting". The annotation is done through a colored bullet where "green" means "recommended", "red" means "not recommended", and "white" means "not interesting". The same type of annotation is used for links to domain-concepts, but the meaning of the colors is different: "red" for "new", "green" for "known", "white" for "well known". Another annotation, using a check mark, indicates that a section has been accessed already.

We study the adaptation offered by InterBook for each type of window separately.

Section windows

We already know there are three frames in this window. The *tool box frame* contains no adaptation. The *text frame* provides content adaptation and link adaptation. The *navigation bar frame* provides link adaptation.

In the *text frame*, if the section is a terminal section then InterBook shows the contents of the section (the text, possibly with images, and links). The only adaptation in the text frame is the color of the "ready bar" which can be green, white or red. (See Figure 5.1 for an example). This adaptation is expressed through the *indicator* attribute. If the section is not a terminal section, InterBook shows a list of links to subsections. It performs content adaptation (to show certain subsubsections and the "(You are here!)" message) and link adaptation to indicate the *ready* status of the subsections. Also in this case a "ready bar" is shown with the appropriate color. (See Figure 5.2 for an example of such a section window).

Since the content of the text frame for a terminal section is not adaptive there is no adaptation rule for the inclusion of this content. For text frames (for terminal or non-terminal sections) we have the following rules to determine the color of the "ready bar":

Rule 1:
C: **select** *P.access*
where *P.access* = **true and** *P.ready* = "recommended"
A: **update** *P.indicator* := "green"

Rule 2:
C: **select** *P.access*
where *P.access* = **true and** *P.ready* = "not recommended"
A: **update** *P.indicator* := "red"

Rule 3:
C: **select** *P.access*
where *P.access* = **true and** *P.ready* = "not interesting"
A: **update** *P.indicator* := "white"

For non-terminal sections InterBook shows a list of links to subsections. This list is not adaptive (and hence we do not have an adaptation rule for that). If the last or before-last visited subsection is in this list or is a subsubsection of an item on the list then InterBook shows links to the subsubsections of that item. We need an adaptation rule to indicate for these subsubsection links whether they are shown or hidden.

Rule 4:
C: **select** *P.access*
where $P.access = \textbf{true}$
A: **update** $M.pres :=$ "show"
where $Fragment(P, F)$ **and** $Subsection(F, M)$ **and**
$Descendant^{-1}(P, Book)$ **and** $(Book.last = F$ **or** $Book.before\_last = F$ **or**
$(Fragment(F, N)$ **and** $(Book.last = N$ **or** $Book.before\_last = N)))$

This rule is complicated because there are four different conditions that may make a subsubsection link be presented, and because this depends on the last or before-last visited section, which is stored in an attribute of a "global" variable for the book.

Rule 5 says not to show subsubsection links for other subsections.

Rule 5:
C: **select** *P.access*
where $P.access = \textbf{true}$
A: **update** $M.pres :=$ "hide"
where $Fragment(P, F)$ **and** $Subsection(F, M)$ **and not** (
$Descendant^{-1}(P, Book)$ **and** $(Book.last = F$ **or** $Book.before\_last = F$ **or**
$(Fragment(F, N)$ **and** $(Book.last = N$ **or** $Book.before\_last = N))))$

Rules 6–8 determine the bullet color and font for link fragments. To greatly simplify the adaptation model we describe rules that determine the bullet color and font when the *ready* status of a section changes. Whenever an annotated link to a section is shown (in any type of window) the annotation is exactly the same. Therefore it makes sense to calculate the annotation when it changes instead of when it needs to be presented.

Rule 6:
C: **select** *P.ready*
where $P.ready =$ "recommended"
A: **update** $P.indicator :=$ "green"; $P.font :=$ "bold"

Rule 7:
C: **select** *P.ready*
where $P.ready =$ "not recommended"
A: **update** $P.indicator :=$ "red"; $P.font :=$ "italic"

Rule 8:
C: **select** *P.ready*
where $P.ready =$ "not interesting"
A: **update** $P.indicator :=$ "white"; $P.font :=$ "normal"

Rule 9 determines whether to show the check mark (*mark* attribute) for the sections that have been read. Initially (through Rules 4 and 5 of the IU phase) the *mark* attribute for every section has the value "hide". Rule 9 sets the check mark for future use. All subsection links that are shown either have their check mark or not depending on whether they were read before or not.

Rule 9:
C: **select** $P.read$
where $P.read = \textbf{true}$
A: **update** $P.mark :=$ "show"

Note that when a page is read, the rule to set the *read* attribute belongs in the UU-pre phase, and the rule to set the *mark* attribute (which is an adaptation attribute) belongs in the GA phase.

InterBook annotates (links to) the most recently visited section (in this book) by "(You are here!)". The section most recently visited section is annotated by "(You were here!)". This annotation only appears if these sections are a subsection or a subsubsection of the current section. Rules 10–13 describe this.

Rule 10:
C: **select** $P_1.access$
where $P_1.access = \textbf{true}$
A: **update** $P_2.fa :=$ "show"
where $Fragment(P_1, P_2)$ **and** $Descendant^{-1}(P_1, Book)$ **and** $Book.last = P_2$

Rule 11:
C: **select** $P_1.access$
where $P_1.access = \textbf{true}$
A: **update** $P_3.fa :=$ "show"
where $Fragment(P_1, P_2)$ **and** $Subsection(P_2, P_3)$ **and** $Descendant^{-1}(P_1, Book)$
and $Book.last = P_3$

Rule 12:
C: **select** $P_1.access$
where $P_1.access = \textbf{true}$
A: **update** $P_2.fw :=$ "show"
where $Fragment(P_1, P_2)$ **and** $Descendant^{-1}(P_1, Book)$ **and** $Book.before\_last = P_2$

Rule 13:
C: **select** $P_1.access$
where $P_1.access = \textbf{true}$
A: **update** $P_3.fw :=$ "show"
where $Fragment(P_1, P_2)$ **and** $Subsection(P_2, P_3)$ **and** $Descendant^{-1}(P_1, Book)$
and $Book.before\_last = P_3$

In the *navigation bar frame*, InterBook shows links to all siblings and all ancestors of the current section. (We assume that a frame with this information is present, without requiring any rules.) The link annotation does not require any additional rules (since we already calculated the bullet color and check mark). But in the navigation bar frame the link to the current page is disabled. Rule 14 describes how this is done:

Rule 14:
C: **select** *P.access*
where *P.access* = **true**
A: **update** *P.link* := "hide"; *P.font* := "black bold"

Domain-concept windows

A domain-concept window contains annotated links to pages. This annotation uses colored bullets and check marks as calculated before. So no additional adaptation rules are needed for these windows.

Content windows

The content window contains a bookshelf frame (without adaptation) and a frame showing a table of contents of the current book. The sections and subsections shown, and the "(You are here!)" and "(You were here!)" notes are the same as on a section page (for a non-terminal section). So no additional adaptation rules are needed for content windows.

Glossary windows

The *glossary* is an important part of InterBook. It is a visualized (and externalized) domain-concept network. Each node of the domain network is represented by a glossary entry in the glossary frame. Vice versa, *each glossary entry corresponds to one of the domain-concepts*. Each glossary entry provides links to all pages that introduce the domain-concept and pages for which the knowledge of the concepts is required. There is no adaptation in the alphabet frame, but there is adaptation in the glossary frame and the content frame. The adaptation in the content frame does not require additional rules because this frame contains annotated links to sections.

In the *glossary frame* InterBook provides domain-concept link annotation which is similar to the section link annotation in the text frame of a section window, described by Rules 6–8. In the glossary frame, however, the annotation is based on the *knowledge* about concepts, not the *ready* status of sections.

Rule 15:
C: **select** *Letter.access*
where *Letter.access* = **true**
A: **update** *C.indicator* := "green"; *C.font* := "bold"
where *GAlphabet*(*Letter*, *C*) **and** *C.knowledge* = 1

Rule 16:
C: **select** *Letter.access*
where *Letter.access* = **true**
A: **update** *C.indicator* := "red"; *C.font* := "italic"
where *GAlphabet*(*Letter*, *C*) **and** *C.knowledge* < 1

Rule 17:
C: **select** *Letter.access*
where *Letter.access* = **true**
A: **update** *C.indicator* := "white"; *C.font* := "normal"
where *GAlphabet*(*Letter*, *C*) **and** *C.knowledge* ≥ 2

Help windows

When the user "clicks" on the *help button* when reading a section, the system generates a help window that shows the user which pages to read to learn the prerequisite concepts for this section. (This list is not adaptive. It includes all relevant pages, including pages the user has read before.) The links that are shown are annotated in the same way as in a section window, so no additional adaptation rules are needed.

5.4.4 Adaptation rules in the UU-post phase

To be able to update the user model after generating the adaptation, and after the system has sent the adapted page to the user's browser, AHAM provides the UU-post phase. InterBook is very well suited for illustrating the use of the UU-post phase. The following rules in the UU-post phase undo some user model changes made in the GA phase.

In the *navigation bar frame* of a section window the link to the current section is disabled. It is shown in black, using a bold font. After the presentation we need to calculate its normal font again. The following rules "counteract" Rule 14 of the GA phase by repeating some of the work performed by GA Rules 6–8:

Rule 1:
C: **select** *P.access*
where *P.access* = **true and** *P.ready* = "recommended"
A: **update** *P.link* := "show"; *P.font* := "bold"

Rule 2:
C: **select** *P.access*
where *P.access* = **true and** *P.ready* = "not recommended"
A: **update** *P.link* := "show"; *P.font* := "italic"

Rule 3:
C: **select** *P.access*
where *P.access* = **true and** *P.ready* = "not interesting"
A: **update** *P.link* := "show"; *P.font* := "normal"

Note that these rules do not have to change the bullet color or the check mark because GA Rule 14 does not change these.

Another change that has to be undone is the indication where to show "(You are here!)" and "(You were here!)". This is done through four rules that are essentially the same as GA Rules 10–13, but this time setting the *fa* and *fw* attributes to "hide":

Rule 4:
C: **select** $P_1.access$
where $P_1.access = \textbf{true}$
A: **update** $P_2.fa :=$ "hide"
where $Fragment(P_1, P_2)$ **and** $Descendant^{-1}(P_1, Book)$ **and** $Book.last = P_2$

Rule 5:
C: **select** $P_1.access$
where $P_1.access = \textbf{true}$
A: **update** $P_3.fa :=$ "hide"
where $Fragment(P_1, P_2)$ **and** $Subsection(P_2, P_3)$ **and** $Descendant^{-1}(P_1, Book)$
and $Book.last = P_3$

Rule 6:
C: **select** $P_1.access$
where $P_1.access = \textbf{true}$
A: **update** $P_2.fw :=$ "hide"
where $Fragment(P_1, P_2)$ **and** $Descendant^{-1}(P_1, Book)$ **and** $Book.before\_last = P_2$

Rule 7:
C: **select** $P_1.access$
where $P_1.access = \textbf{true}$
A: **update** $P_3.fw :=$ "hide"
where $Fragment(P_1, P_2)$ **and** $Subsection(P_2, P_3)$ **and** $Descendant^{-1}(P_1, Book)$
and $Book.before\_last = P_3$

The presentation of the "(You are here!)" and "(You were here!)" notes requires that InterBook remember these two sections after presenting the page. The following rule must not only be executed after generating the adaptation (because otherwise the wrong pages would be annotated) but also after Rules 4–7 because these rules use the *last* and *before_last* attributes that are assigned to in Rule 8. We omit the "tricks" needed to enforce this execution order here.

Rule 8:
C: **select** $P.access$
where $P.access = \textbf{true}$
A: **update** $Book.before\_last := Book.last$; $Book.last := P$
where $Descendant^{-1}(P, Book)$

The final action to be performed in UU-post is resetting the *access* attribute:

Rule 9:
C: **select** *P.access*
where *P.access* = **true**
A: **update** *P.access* := **false**

Here again a "trick" is needed to enforce that these rules are executed last.

Rule 10:
C: **select** *Book.access*
where *Book.access* = **true**
A: **update** *Book.access* := **false**

Rule 11:
C: **select** *Letter.access*
where *Letter.access* = **true**
A: **update** *Letter.access* := **false**

Rule 12:
C: **select** *Bookshelf.access*
where *Bookshelf.access* = **true**
A: **update** *Bookshelf.access* := **false**

Rule 13:
C: **select** *GW.access*
where *GW.access* = **true**
A: **update** *GW.access* := **false**

5.5 InterBook: Termination and Confluence

The previous section showed that we can describe InterBook using AHAM-CA rules. The rules in the UU-pre phase trigger each other, so in principle there may be termination and confluence problems. InterBook, however, provides timely and deterministic result to the user. We will discuss below how rules trigger each other in InterBook.

The rules the IU, GA and UU-post phases do not trigger each other in the same phase. (To have a simple description in IU phase, we do use triggers, e. g. Rule 1 triggers Rules 4–6, but this is just to transfer *Bookshelf.new_access* to *Book.new_access*. This happens in an acyclic way, so it obviously terminates.) Rules in the UU-pre phase do trigger each other, in the same execution phase. So for termination we only have to verify the UU-pre phase.

Rules 1–3 transfer *access* into *knowledge*. Rule 4 transfers *access* into *read*. Rule 5 transfers *test* into *knowledge*. Rules 6–8 transfer *knowledge* to *ready*. From the fact

that information is passed between attributes (of pages or concepts) in an acyclic way we immediately conclude that the Activation Graph, for every possible domain model instance, must be acyclic. According to Theorem 4.3 in Section 4.3.2 InterBook guarantees termination.

The theory in Chapter 4 has conditions for confluence that are too strict. Still, we can verify confluence in InterBook by examining the **where** clauses and updates of rules that have the same **select** clause. Many rules depend on *P.access* becoming **true**. However, we have carefully written these rules in such a way that the **where** clauses (in the C or the A part) are mutually exclusive (as in UU-pre Rules 1, 2 and 3 for instance, or 6 and 7, or GA Rules 1, 2 and 3, etc.) whenever the same attribute of the same concept or page is assigned a value.

The only place where we left a "conflict" is in the UU-post phase where priorities (or a simulation trick) are needed to ensure that Rules 1–7 are executed before Rule 8 and Rule 8 before Rule 9. So InterBook guarantees confluence, although our description leaves an incomplete detail at the end.

5.6 Summary of Validation 1

InterBook is a special-purpose AHS: it is intended only for electronic textbooks. InterBook offers a versatile user interface with multiple windows and a frame-based presentation. It combines the information with helpful navigation support. It provides little adaptive content but extensive adaptive navigation support. This navigation support consists of a partial table of contents, a glossary, an overview of required prerequisite knowledge for the current book page, and an overview of "outcome" knowledge.

InterBook can be described in AHAM, as shown by our definition of DM-InterBook, UM-InterBook, and AM-InterBook. DM-InterBook consists of a concept hierarchy and concept relationships. The concept hierarchy consists of fragments, frames, pages and abstract concepts. (The frames have been used in the informal description but do not appear in DM-InterBook.)

UM-InterBook uses an overlay model, with several attribute-value pairs per concept or page. InterBook most likely does not use all these attributes. We introduced some artificial attributes in order to be able to describe the adaptation model.

AM-InterBook consists of predefined rules to perform knowledge updates and generate adaptive link annotation. We have not described the generation of pages, which is not adaptive, but only the annotation, which is adaptive. Even though InterBook only uses two concept relationship types: "requirement" (or prerequisite) relations and "outcome" relations, a rich set of adaptation possibilities is offered.

AM-InterBook contains several cases of rules that trigger each other. The rule triggering is limited, and termination and confluence are guaranteed. Still, even this relatively small example has revealed one shortcoming of the "standard" way in which Chapters 3 and 4 suggest structuring the adaptation model: the use of only four execution phases (IU, UU-pre, GA and UU-post) is not enough to allow the description of a "moderate" AHS without

the need for priorities or other ways of subdividing rules into sets that are executed one after the other.

The description of InterBook in AHAM shows that the model can express the adaptation functionality of a relatively complex AHS, not designed by us.

Chapter 6

Validation of AHAM: the AHA! system

This chapter describes the second Version 1.0 of the AHA! system [De Bra et al., 2000] using the AHAM reference architecture described in this dissertation. AHA! is the Adaptive Hypermedia Architecture, used for distance learning applications and for a "kiosk" information system at the Eindhoven University of Technology (TU/e). AHA! is a tool for providing adaptive hypermedia applications on the World Wide Web. It claims to be "general purpose" but has mainly been used to develop on-line courses.

AHA! exists in three main versions. AHA! Version 0 [De Bra and Calvi, 1998a] was the first adaptive hypermedia system built at the TU/e. It has very simple rules using one attribute with a Boolean value for every course page or concept. It does not allow rules to trigger each other. AHA! Version 1 allows rules that trigger each other in a certain way. It still uses one attribute in its rules, but the attribute value can be an integer ranging from 0 to 100. AHA! Version 2 is the current development version which uses many features in its rules, inspired by the AHAM model. We chose AHA! Version 1 [De Bra et al., 2000] to validate our reference architecture for adaptive hypermedia applications in this chapter. AHA! Version 1 is the "production" version. As with InterBook (see Chapter 5) we prefer to describe the production version which is stable (both in terms of the software and in terms of the features it offers). Section 6.1 introduces the user interface of AHA! and the user interaction with AHA!. Section 6.2 describes the domain model of AHA!. Section 6.3 describes the user model of AHA!. Section 6.4 describes the adaptation model of AHA! using AHAM-CA rules. In Section 6.5 we study the termination and confluence properties of AHA!. Section 6.6 summarizes the validation of our reference model that was done by describing AHA!.

6.1 Introduction

AHA! (Adaptive Hypermedia Architecture) is a Web-based adaptive hypermedia system intended to serve many different purposes. As such AHA! is able to perform adaptation

that is based on the user's browsing actions, regardless of the interpretation of browsing as a learning process, exploration or other purpose. AHA! provides adaptive content and adaptive navigation support.

To illustrate the adaptation features of AHA! we show two applications built using AHA!: the online courses 2L690 and 2R350, dealing with Hypermedia and Graphical User-Interfaces respectively.

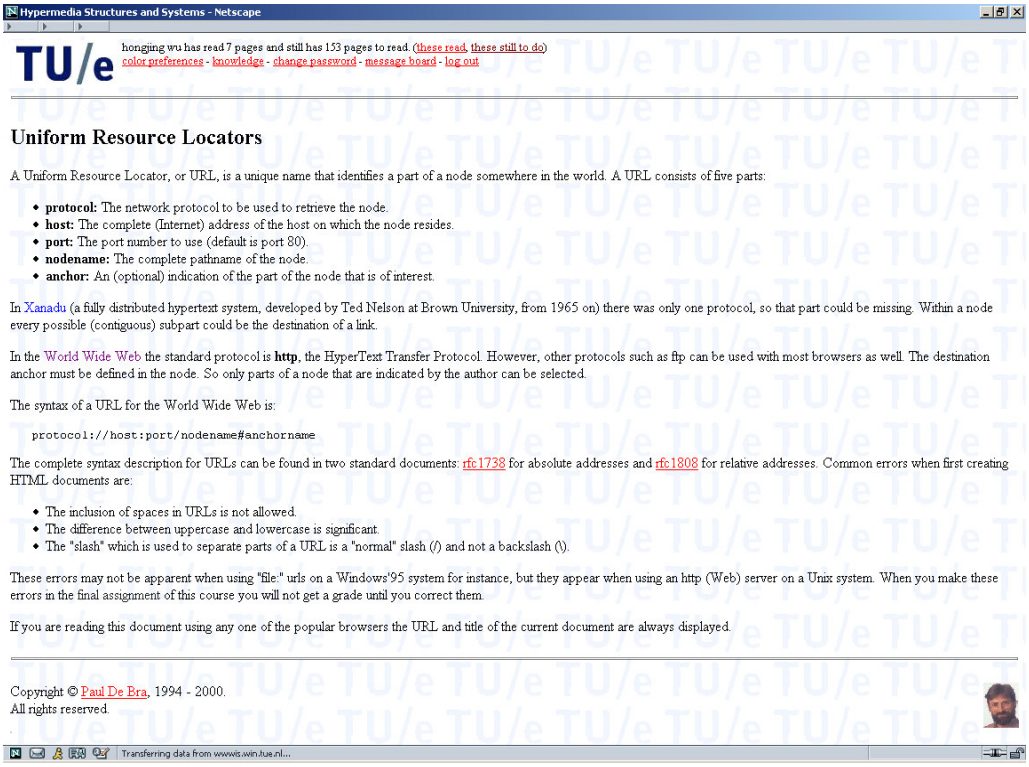

Figure 6.1: A screen shot of a 2L690 window

A page from course 2L690 is shown in the Figure 6.1. Every page of 2L690 has a *header*, a *footer*, and a *section* part. We will refer to these parts as "frames" but they are not frames in the HTML sense (unlike in InterBook where the frames are HTML frames).

The *header* frame is an interface to special AHA! functions. Users can see how much they have learned already, how much they still have to do, and they can change the knowledge values and color preferences in their user model. The *footer* frame gives a copyright notice and shows a photo of the author of the course. The header and footer are created by the author of the course. The special functions of AHA! (like number or list of pages read and pages still to do) can be included in the header or footer as desired by the author. In order to keep our description of AHA! simple we do not describe these special functions of AHA! using the AHAM model. A list of pages already read (or still to read) is a simple page listing all the pages, each in a conditional fragment that is shown or hidden

depending on the "visited" status of the page. The forms to change knowledge values and color preferences are similar and the events they generate require special handlers that do not interact with the rules for handling a simple page access.

AHA! provides adaptive content by conditionally including fragments, and adaptive navigation support by annotating (actually coloring) links. In Figure 6.1 the brief explanation on Xanadu (shown between parentheses) is conditionally included. It is only shown to users who have not read the page about Xanadu. The link to Xanadu is blue, indicating that this page is recommended to be read.

As shown in Figure 6.2, every page of the course 2R350 consists of a *header*, a *footer*, a *section* part, and a *site-map* frame. The first three parts of 2R350 are similar to the layout in the 2L690 course. The difference is that in 2R350 there is a site-map frame that serves as an additional navigation support tool. The site-map shows the structure of chapters and sections of this course. It works in more or less the same way as a menu or the "Folders" part of the Windows explorer. When a user clicks on a menu item, the system opens a submenu for this item if it is not a leaf-level item; at the same time the system closes other submenu(s) that might be open. Each menu item is a link to a section page of the course, and the menu link presentation is adapted to the user's knowledge by using the link annotation technique. Figure 6.2 shows links in three colors: blue, black and purple, corresponding to the states "recommended", "not recommended" and "not interesting".

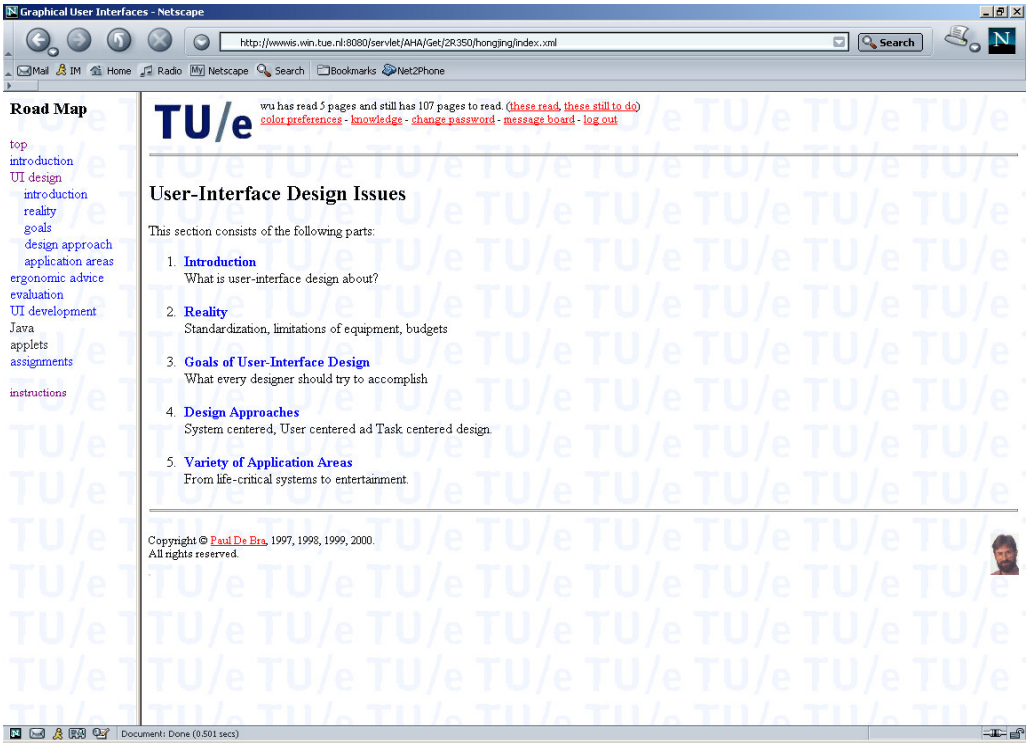

Figure 6.2: A screen shot of a 2R350 window

In the AHAM description of AHA!, we will refer to the courses 2L690 and 2R350, to illustrate interesting ways in which the functionality of AHA! can be used. The description of AHA! is not entirely accurate. We use the following simplification:

- *actual AHA! use*: In course 2R350 when a user clicks on a link (in the site-map frame or the section frame) the destination of the link is opened in the right frame. The destination is an HTML page that contains the section part and includes the header and footer. Every page in 2R350 contains a 1 line Javascript function that orders the browser to reload the site-map frame.

- *modeled AHA! functionality*: When a user clicks on a link we assume that a page consisting of header, footer, section part and site-map frame is accessed. We thus assume that the result of clicking on a link is that only one new page is loaded.

AHA! contains some "magic" to generate text dynamically. The header for instance shows the name of the user and the number of pages the user has read and the number of pages still to do. We model AHA! headers and footers as fragments. In AHAM fragments are atomic concepts, of which the internals are hidden in the Within-Component layer.

6.2 AHA!: Domain Model

Let us call the domain model of AHA!, DM-AHA. The DM-AHA consists of a set of concepts and concept relationships, according to AHAM. In DM-AHA the lowest level concepts are fragments. In AHA! fragments can contain other fragments. In order to satisfy the restrictions of AHAM we assume that an application does not contain fragments within fragments. This is no real restriction because a conditional fragment within a fragment can be modeled as a fragment that has the conjunction of its own and its parent's (or ancestors') condition. We can thus "flatten" the structure of fragments within fragments in order to eliminate it. Fragments within fragments are most useful for the conditional opening and closing of menus and submenus. One level higher in the concept hierarchy we have the page level. Pages consist of conditional and unconditional fragments. One of the properties of AHA! that makes it somewhat difficult for authors to use is that the conditions are associated with the fragments and written down on the pages containing the fragments, but that conceptually they represent specific adaptation rules and are thus not part of DM-AHA. AHA! also supports abstract concepts. Normally a concept is connected to a page with the same name. Concepts with a name that does not correspond to a page are automatically abstract concepts. The only way in which abstract concepts have a different function from page concepts is that there is no "access" event for an abstract concept.

In order to connect concepts in AHA! using AHAM constructs we define the following types of concept relationships:

- $Fragment(P, F)$ means F is a fragment of page P. Through the $Fragment$ relationship we can decide on the inclusion of fragments of a page.

- $Link(P_1, P_2)$ means that there is a hyperlink from P_1 to P_2. Through the *Link* relationship we can perform adaptation of link anchors based on the desirability of the link destination.

- $Contains(C, P)$ means that the abstract concept C "contains" page or concept p. The *Contains* relationship is used to be able to perform an action on user model attributes of all pages of an application (or course) at once.

- $Contains^{-1}(P, C)$ is the inverse of *Contains*. We only use this relationship type to find out to which application (or course) a page belongs.

These relationships are binary, and can only be applied from the first parameter to the second in adaptation rules, because of the restriction of AHAM introduced in Chapter 4. We introduced $Contains^{-1}$ because of this restriction.

It may seem strange that AHA! provides adaptive hypermedia but does not offer concept relationship types such as prerequisites. The reason is that AHA! only offers authors the ability to write specific adaptation rules. It has no generic adaptation rules based on relationship types.

The presentation specification *pres* of a fragment can have the values "show" and "hide". Instead of using anchor values we will use the presentation specification *pres* of a page to indicate the "desirability" of page as the destination of a link. The desirability can be "good", "bad" or "neutral", representing that the page is "recommended", "not recommended" or "not interesting".

6.3 AHA!: User Model

AHA! maintains a user model for each individual user, called UM-AHA. The real user model in AHA! contains only two attributes: one representing *knowledge* and one representing the *visited* status. The visited status is used by the system but cannot be used in author-defined adaptation rules. Only the knowledge can be used by the author. AHA! simply uses the page name to represent this attribute. If an author wants to associate more attributes with a page this can only be done by creating new concepts, and by using, or abusing, the knowledge attribute for these other purposes.

In UM-AHA we introduce additional attributes to represent temporary information about concepts. (The temporary nature of this information can be seen from its initialization at the start of the execution of the adaptation rules.) Below is the list of attributes:

- A *knowledge* attribute stores user's knowledge state about a page (concept). It has an integer value between 0 and 100.

- A *visited* attribute keeps track of whether a page was visited. It has a Boolean value. (Actually in AHA! the value is an integer but only the numbers 0 and 100 are used to mean **false** and **true**.)

- A *change* attribute stores the amount by which the knowledge has changed. This is used in the propagation of knowledge from pages to sections and from sections to chapters.

- A *ready* attribute represents whether this page is "recommended" to be read. It uses values of "recommended", "not recommended" and "not interesting". We use an additional value "unknown" for initialization purposes, and a value "needed" to trigger the computation of the **true** *ready* value. This attribute must be computed each time after receiving an event because it is not stored in the user model in AHA!.

- An *access* attribute stores the event of "clicking" on a link to the page. It changes from **false** to **true** when the user clicks on a link to the page, and it resets to **false** after the presentation generation has been performed.

6.4 AHA!: Adaptation Model

The adaptation model of AHA!, called AM-AHA, describes how to update the user model and how to generate the adaptation based on that user model and the domain model. In AHA! authors write specific adaptation rules to define how reading pages generates user model updates. The system offers generic adaptation rules for initialization, and for generating the adaptation from the current user model (and access event). AHA! performs all user model updates before generating the adaptation. As a result it does not have rules in the UU-post phase. However, we can use the UU-post phase to reset the *access* attribute of the page that was accessed.

In this section we will assume that the top-level abstract concept that *Contains* all other concepts and pages is called *Book*. In the examples we gave earlier this would be either 2L690 or 2R350.

6.4.1 Adaptation rules in the IU phase

When a user visits an adaptive (AHA!) application for the first time the system builds and initializes a user model. The initialization is performed in what we call the IU phase.

The IU phase requires that an application *Book* contains a special attribute $FirstPage$ that indicates the page to be accessed when a user logs in.

Rule 1 says that when a new user logs on the knowledge for all pages and concepts is set to 0.

Rule 1:
C: **select** $Book.register$
A: **update** $P.knowledge := 0$
where $Contains(Book, P)$

Rules 2 and 3 set the *access* attribute. All pages except *FirstPage* are assigned the value **false**, except for *FirstPage* which is assigned the value **true**. By making *FirstPage.access* **true**, the rules in other execution phases are triggered for *FirstPage*.

Rule 2:
C: **select** $Book.register$
A: **update** $P.access := \textbf{false}$
where $Contains(Book, P)$ **and** $P \neq Book.FirstPage$

Rule 3:
C: **select** $Book.register$
A: **update** $P.access := \textbf{true}$
where $Contains(Book, P)$ **and** $P = Book.FirstPage$

AHA! only stores the *knowledge* and *visited* attribute. The other attributes in UM-AHA need to be initialized when an event happens and before the rules are executed that are triggered by that event. The rules in IU initialize the attributes *change* and *ready*, not just when the user registers or logs in, but also at the beginning of rule execution after "clicking" on a link.

Rule 4 initializes the *change* attribute to 0 for all concepts of the application to which the accessed page belongs.

Rule 4:
C: **select** $P.access$
where $P.access = \textbf{true}$
A: **update** $Q.change := 0$
where $Contains^{-1}(P, Book)$ **and** $Contains(Book, Q)$

Rules 5 and 6 initialize the *ready* attribute to "unknown" for all pages that can be reached through a link from the accessed page, and for all fragments of the accessed page. Whenever the real value of *ready* is calculated (in the UU-pre phase) it will be a change from "unknown" and thus trigger other rules that depend on the *ready* attribute.

Rule 5:
C: **select** $P.access$
where $P.access = \textbf{true}$
A: **update** $Q.ready :=$ "unknown"
where $Link(P, Q)$

Rule 6:
C: **select** $P.access$
where $P.access = \textbf{true}$
A: **update** $F.ready :=$ "unknown"
where $Fragment(P, F)$

Note that rules in the IU phase are not functionally different from rules in the subsequent phases. The updates to the *change* and *ready* attributes that are performed here may try to trigger rules in other phases just like any update to any attribute. Therefore all adaptation rules in the other phases will check the value of *change* or *ready*. The conditions will check that $change \neq 0$ and $ready \neq$ "unknown".

In the UU-pre phase there are rules that use the "visited" status of pages. In AHA! a page is considered visited in adaptation rules for the access of the page itself. It is thus important to set the visited status before entering the UU-pre phase:

Rule 7:
C: **select** *P.access*
where $P.access =$ **true**
A: **update** $P.visited :=$ **true**

6.4.2 Adaptation rules in the UU-pre phase

AHA! updates the user model when the user "clicks" on a link and thereby accesses a page. Part of the definition of how the update process works is *generic*, meaning independent of the accessed page, and part is *specific*, meaning written by the author. We will describe the adaptation rules in such a way that the generic part is contained in generic rules and the specific part in specific rules. We hope that this leads to the clearest albeit not shortest description.

The first rule that is executed when the user accesses a page determines the desirability of the page. This desirability is defined in AHA! through a *requirement*, specified in XML in a file "xmlreqlist". An example of such a requirement is:

```
<concept>
  <conceptname>
    dexter_runtime
  </conceptname>
  <relationexpression>
    advanced>70 and dexter>70
  </relationexpression>
</concept>
```

It expresses that the page *dexter_runtime* is "recommended" if the knowledge of the concepts *advanced* and *dexter* is higher than 70. In AHAM we can represent the processing of this rule through three specific adaptation rules:

Rule 1:
C: **select** *dexter_runtime.access*
where $dexter\_runtime.access =$ **true**
A: **update** *dexter_runtime.ready* := "recommended"
where $dexter\_runtime.visited =$ **false and**
$advanced.knowledge > 70$ **and** $dexter.knowledge > 70$

Rule 2:
C: **select** *dexter_runtime.access*
where *dexter_runtime.access* = **true**
A: **update** *dexter_runtime.ready* := "not interesting"
where *dexter_runtime.visited* = **true and**
advanced.knowledge > 70 **and** *dexter.knowledge* > 70

Rule 3:
C: **select** *dexter_runtime.access*
where *dexter_runtime.access* = **true**
A: **update** *dexter_runtime.ready* := "not recommended"
where not (*advanced.knowledge* > 70 **and** *dexter.knowledge* > 70)

In order to later generate the presentation of a page we need to know not just the "ready" status of the presented page but also of every fragment of that page and every page to which there is a link. Strictly speaking this can be determined in the GA phase because *ready* is not a user model attribute in AHA! (although we use it in UM-AHA). We include it here (in UU-pre) because otherwise we encounter rule execution order problems in the GA phase.

The rules for determining the status of fragments and link destinations work in two phases: first we determine the fragments and pages of which we need to calculate the status. We can do this through a *generic* rule. Then we determine the status itself. For that we need *specific* rules because in AHA! the author writes the requirements for pages and fragments as specific rules.

Rule 4:
C: **select** *P.access*
where *P.access* = **true**
A: **update** *F.ready* := "needed"
where $Fragment(P, F)$

Rule 5:
C: **select** *P.access*
where *P.access* = **true**
A: **update** *Q.ready* := "needed"
where $Link(P, Q)$

The "needed" status is used as the trigger for the specific rules. Consider a fragment "withmenu" that shows a submenu in the course 2R350. In AHA! it appears as follows:

```
<if expr="evalmenu>0 and withmenu>0">
<block>
  ...
</block>
```

Actually the menu "evaluation with users" appears within a larger menu "evaluation", but as we flatten the fragments within fragments in AHAM we obtain the conjunction of the conditions for both menus. Assuming the name *withmenu* for this fragment, the pair of rules for determining the status of the fragment is:

Rule 6:
C: **select** *withmenu.ready*
where *withmenu.ready* = "needed"
A: **update** *withmenu.ready* := "recommended"
where *evalmenu.knowledge* > 0 **and** *withmenu.knowledge* > 0

Rule 7:
C: **select** *withmenu.ready*
where *withmenu.ready* = "needed"
A: **update** *withmenu.ready* := "not recommended"
where not (*evalmenu.knowledge* > 0 **and** *withmenu.knowledge* > 0)

For pages that appear as link destinations the rules are very similar as for the accessed page. But instead of *access* it is now the *ready* attribute that triggers the rules. Recall the condition for the concept *dexter_runtime*. The rules for determining the status of this concept when it is the destination of a link are:

Rule 8:
C: **select** *dexter_runtime.ready*
where *dexter_runtime.ready* = "needed"
A: **update** *dexter_runtime.ready* := "recommended"
where *dexter_runtime.visited* = **false and**
advanced.knowledge > 70 **and** *dexter.knowledge* > 70

Rule 9:
C: **select** *dexter_runtime.ready*
where *dexter_runtime.ready* = "needed"
A: **update** *dexter_runtime.ready* := "not interesting"
where *dexter_runtime.visited* = **true and**
advanced.knowledge > 70 **and** *dexter.knowledge* > 70

Rule 10:
C: **select** *dexter_runtime.ready*
where *dexter_runtime.ready* = "needed"
A: **update** *dexter_runtime.ready* := "not recommended"
where not (*advanced.knowledge* > 70 **and** *dexter.knowledge* > 70)

After determining the "ready" status of the accessed page we can determine the knowledge update that occurs when the page is accessed. This is hard-coded behavior of AHA!

and can thus be described by generic adaptation rules. Note that we cannot have these rules be triggered by *P.access* because then they might be evaluated before the rules that determine the "ready" status. But since the "ready" status was "unknown" and is changed to "recommended", "not interesting" or "not recommended", and this only for the accessed page, we can use this change to trigger the knowledge update rules.

Rule 11:
C: **select** *P.ready*
where *P.ready* = "recommended" **or** *P.ready* = "not interesting"
A: **update** $P.change := 100 - P.knowledge$; $P.knowledge := P.knowledge + P.change$

Rule 12:
C: **select** *P.ready*
where *P.ready* = "not recommended" **and** $P.knowledge < 35$
A: **update** $P.change := 35 - P.knowledge$; $P.knowledge := P.knowledge + P.change$

From these rules we see that AHA! can change the knowledge of a page in two ways: the knowledge becomes 100 if the page was not "not recommended"; it becomes 35 if the page was "not recommended" except when it was already higher than 35 in which case the value does not change. The rules could have been simplified a bit (for instance because we know in Rule 4 that *P.knowledge* becomes 100, but we want to illustrate the typical construct that we first determine the *change* and then add the *change* to *knowledge*.

The change to the knowledge value of the page can be propagated to other concepts in AHA! through rules that are written by the author. These rules are XML constructs in a file called "xmlgenlist". AHA! limits the recursive propagation to monotonic updates. We need to incorporate this limitation in the adaptation rules.

Suppose an author writes the following adaptation rules in AHA! (adapted from rules in course 2R350):

```
<genitem>
  <name>eval_with</name>
  <genlist>eval_with_knowledge:+100 eval_with:0
           withmenu:100 withoutmenu:0</genlist>
</genitem>
<genitem>
  <name>eval_with_knowledge</name>
  <genlist>evaluation:+50</genlist>
</genitem>
```

The rules are motivated as follows: There is a menu dealing with the topic of "evaluation". The topic consists of "evaluation with users" and "evaluation without users". There are submenus corresponding to these items. When the user "clicks" on the "eval_with" link and accesses the corresponding page the following actions take place:

- The "eval_with" knowledge first becomes 100. Let us assume that the value was 0 before and that the page was recommended.
- The knowledge of "eval_with" is copied to another concept "eval_with_knowledge" in order to remember it.
- The attribute of "eval_with" is reset to 0 so that the rule will still have effect the next time the user clicks on "eval_with".
- The attribute of "withmenu" is set to 100. This will cause the submenu for "evaluation with users" to be shown.
- The attribute of "withoutmenu" is set to 0. This will cause the submenu for "evaluation without users" to be hidden.
- The update to "eval_with_knowledge" is monotonic (indicated by +100). This means that the rule for "eval_with_knowledge" will also be executed. This rule propagates 50% of the update to "eval_with_knowledge" to the concept "evaluation".

In AHAM the AHA! adaptation rules are translated to specific adaptation rules. The updates all depend on the *change* to the knowledge of "eval_with". (Note that the previous generic rules create this *change* value.)

Rule 13 performs the monotonic relative update to "eval_with_knowledge". In AHA! only monotonic relative updates are propagated. We represent this in AHAM by only setting the change attribute in the case of a monotonic relative update. Propagation in our representation depends on the change attribute, so the update to "eval_with_knowledge" will be propagated. We do not set the change attribute for the other updates (Rules 14, 15 and 16) so these do not cause propagation.

Rule 13:
C: **select** *eval_with.change*
where $eval\_with.change \neq 0$
A: **update** $eval\_with\_knowledge.change :=$
$\mathbf{min}(100 - eval\_with\_knowledge.knowledge, 1.0 \times eval\_with.change)$;
$eval\_with\_knowledge.knowledge :=$
$eval\_with\_knowledge.knowledge + eval\_with\_knowledge.change$

Note that we could have written $eval\_with\_knowledge.knowledge := 100$ but we wanted to illustrate that the value is augmented by 100% of the change to the knowledge of eval_with and limited to 100. The given expression is easier to generalize to other values as we shall see in Rule 10 below. Also note that when *change* is set to 0 (as in the IU phase) it does not trigger this rule.

Rule 14:
C: **select** *eval_with.change*
where $eval\_with.change \neq 0$
A: **update** $eval\_with.knowledge := 0$

Rules 15 and 16 are used to "play" with menus in the site-map frame of course 2R350. They set the knowledge attribute of "withmenu" and "withoutmenu" to 100 and 0 respectively. Of course they do not represent knowledge, but the knowledge attribute is the only attribute available to the author.

Rule 15:
C: **select** *eval_with.change*
where $eval\_with.change \neq 0$
A: **update** $withmenu.knowledge := 100$

Rule 16:
C: **select** *eval_with.change*
where $eval\_with.change \neq 0$
A: **update** $withoutmenu.knowledge := 0$

Rule 17 describes the propagation of knowledge that is achieved through the rule associated with concept "eval_with_knowledge". It illustrates the propagation in AHA! in general through its similarity with Rule 13:

Rule 17:
C: **select** *eval_with_knowledge.change*
where $eval\_with.change \neq 0$
A: **update** $evaluation.change :=$
$\mathbf{min}(100 - evaluation.knowledge, 0.5 \times eval\_with\_knowledge.change)$;
$evaluation.knowledge := evaluation.knowledge + evaluation.change$

Note that since the update to "evaluation" is again a monotonic relative update it may propagate further through other rules.

6.4.3 Adaptation rules in the GA phase

In AHA! fragments are shown when their condition is **true**. In the UU-pre phase we have calculated the "ready" status of fragments included in the accessed page. A pair of rules in the GA phase is sufficient to generate the required presentation specification:

Rule 1:
C: **select** *P.access*
where $P.access =$ **true**
A: **update** $F.pres :=$ "show"
where $Fragment(P, F)$ **and** $F.ready =$ "recommended"

Rule 2:
C: **select** *P.access*
where $P.access =$ **true**
A: **update** $F.pres :=$ "hide"
where $Fragment(P, F)$ **and** $F.ready =$ "not recommended"

In AHA! an author marks conditional links (i. e. links for which there is adaptation of the link anchor) through an HTML link class called "conditional". AHA! changes the class to "good", "bad", and "neutral" depending on whether the destination of the link is a "recommended", "not recommended" or "not interesting" page. Through a style sheet these link classes are coupled with link colors, which are "blue", "black" and "purple" by default but which can be changed by the user.

Rule 3:
C: **select** $P.access$
   where $P.access =$ **true**
A: **update** $Q.pres :=$ "good"
   where $Link(P, Q)$ **and** $Q.ready =$ "recommended"

Rule 4:
C: **select** $P.access$
   where $P.access =$ **true**
A: **update** $Q.pres :=$ "bad"
   where $Link(P, Q)$ **and** $Q.ready =$ "not recommended"

Rule 5:
C: **select** $P.access$
   where $P.access =$ **true**
A: **update** $Q.pres :=$ "neutral"
   where $Link(P, Q)$ **and** $Q.ready =$ "not interesting"

6.4.4 Adaptation rules in the UU-post phase

AHA! does not use a UU-post phase. However, we can use this phase to make the AHA! representation in AHAM independent of how the system deals with the attributes that represent events. When the user "clicks" on a link to page P we represent this as the $P.access$ attribute becoming **true**. After executing the rules $P.access$ has to become **false** again. We can actually arrange for this in the UU-post phase:

Rule 1:
C: **select** $P.access$
   where $P.access =$ **true**
A: **update** $P.access :=$ **false**

This rule explains why in all rules we tested for $P.access$ being **true**. If we allowed rules to also be triggered when $P.access$ becomes **false** then this rule in UU-post would trigger rules in previous phases again, thus potentially causing infinite loops, but certainly causing undesired rule executions and results.

6.5 AHA!: Termination and Confluence

In order to determine *termination* and *confluence* we unfortunately cannot use the constraints given in Section 4.8 because these assume the AHS works with *generic* adaptation rules (of different types), whereas AHA! only uses *specific* rules. But even if we could somehow group the rules to make them generic a constraint such as Constraint 4.5 would not hold anyway.

So in order to determine *termination* we turn to the conservative approach by using the Propagation Algorithm PA from Section 4.6.1 to build an activation graph AG. The basic PA creates an edge between two (instantiated) rules if there is an attribute of the same concept that appears in the action of the first rule and in the condition of the second rule. It is immediately clear that the basic PA creates cycles because Rule 1 of the UU-post phase updates *P.access* and its condition uses *P.access* so we have a cycle right there. More cycles can be found in Rules 6 through 10 of the UU-pre phase: These rules assign values to the *ready* attribute and use the same attribute in their condition. So we have to augment the PA with the intelligence to at least look at the values assigned in the action of the first rule and the values used in the condition of the second rule. Rules 6 through 10 of UU-pre compare the *ready* attribute with value "needed" and only assign values "recommended", "not recommended" or "not interesting", so they cannot cause an infinite loop.

Careful analysis of the rules for AHA! reveals that UU-pre rules such as Rule 13 and 17 can generate what appears to be an infinite loop if the author creates a structure in which there is a cycle of monotonic relative knowledge updates. (The AHA! system has no authoring tool to warn authors of this danger.) Fortunately, such loops cannot be infinite loops because the knowledge values cannot be monotonically increased forever as the values cannot exceed 100. In the worst case, a loop must end when the knowledge value of all concepts reaches 100. The number of rule execution steps is thus (in the worst case) linear in the number of concepts. AHA! thus enforces termination through the limit of the value range of the knowledge attribute and the propagation through monotonic relative updates.

Regarding confluence it is easy to see that AHA! with the rules defined earlier in this chapter cannot guarantee confluence. If concept A has a rule that propagates a monotonic relative update to B and C, and B and C each perform a different absolute update on D then the outcome depends on the order in which the updates are performed. (This order is predetermined by AHA!'s adaptation engine, but we wish to guarantee confluence without using that implementation dependent execution order.) In [De Bra et al., 2000] two additional constraints are defined on AHA! rules in order to enforce confluence:

- Only the rules associated with a page may contain absolute updates.
- Relative updates (monotonic or not) are only allowed on abstract concepts, not on pages.

In the example given above the updates on B and C mean that according to the second constraint B and C must be abstract concepts. The absolute updates originating from B

and C require that B and C are pages because of the first constraint. So this example that violates confluence is forbidden in AHA!.

In De Bra et al. [2000] we claimed that the above condition is sufficient to make AHA! guarantee confluence. Unfortunately this claim is false because of the possibility of rounding errors. Consider the following set of monotonic relative updates in Figure 6.3:

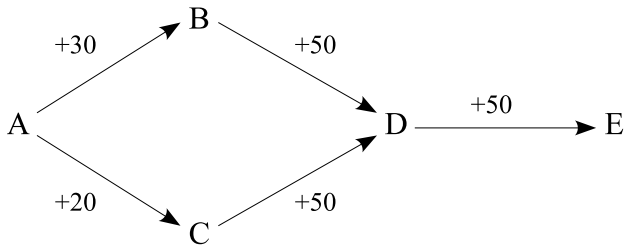

Figure 6.3: An example of AHA! rule propagation

Suppose the knowledge of A is changed from 0 to 100. The knowledge of B is then increased by 30 and the knowledge of C is increased by 20. These two updates propagate to D: The first increases the knowledge of D by 15, the second by 10. These updates propagate to E as an increase of 7 (50% of 15 rounded down) and an increase of 5. However, assume that the knowledge of D is already 80 when this process starts. This knowledge value cannot be increased by 15 and 10 because then it would exceed 100. The following possibilities exist:

- If the update from B occurs first then the knowledge of D is first increased by 15, and then by only 5 (instead of 10) because the limit of 100 is reached. The propagation to E is first 7 and then 2 (both updates being rounded down). E is thus increased by 9.

- If the update from C occurs first then the knowledge of D is first increased by 10, and then by only 10 (instead of 15) because the limit of 100 is reached. The propagation to E is first 5 and then 5 again. E is thus increased by 10.

So, although in most cases AHA!'s behavior is confluent there are exceptional cases in which confluence is not guaranteed.

6.6 Summary of Validation 2

AHA! is clearly a lower-level AHS than InterBook, relying heavily on the use of *specific* adaptation rules. As such it is more difficult to use for authors. AHA! is also more difficult to model in AHAM. We needed some artificial constructs to describe the built-in behavior of AHA! through *generic* adaptation rules and combine it with the *specific* rules an author writes.

We have hinted at the possibilities of AHA! to create different presentation styles, in particular to create menus that function as an adaptive site-map. Such creations require inventive use of conditional inclusion of fragments and the invention of additional "concepts" to make the user model remember which menu should be open.

In order to describe AHA! we have "invented" concept relationships in DM-AHA that do not really exist in AHA! and attributes in UM-AHA (*change*, *ready* and *access*) that do not exist in AHA!. This shows that it is certainly an overstatement that AHAM would be specifically designed as a model of AHA!. (Some people have informally told us that they got this impression because the name AHAM suggests that it is the AHA! Model, which it is not.)

From our AHAM representation of AHA! we could not conclude on *termination* in a straightforward way using the theory of Chapter 4. A refined Propagation Algorithm (considering the values used in the conditions and actions of rules) left one unresolved problem with monotonic relative updates. Fortunately we know that this problem does not really cause infinite loops because the knowledge values have an upper bound of 100. So AHA! guarantees termination through what we referred to as *static enforcement* in Chapter 4.

Regarding *confluence* the theory again did not lead to the conclusion that AHA! would guarantee confluence. Theorem 4.4 requires all pairs of rules to commute, which is definitely not guaranteed (just look at the rules for menus). The rules of Section 4.8 could not be used because AHA! uses *specific* adaptation rules. AHA! tries to use *static enforcement* for confluence through constraints introduced in [De Bra et al., 2000]. Unfortunately these constraints still leave cases where the execution order chosen by the AE matters.

As a validation of AHAM the case of AHA! is positive in the sense that we have been able to represent the behavior of AHA! in the AHAM model. However, some tricks were needed, the most important one being that some part of the GA phase was moved to the UU-pre phase in order to avoid order dependence or the need for sub-phases within the GA phase. Indeed: first determining the status of fragments and link destinations and then generating the presentation specifications are two distinct phases, and the first one does not really belong in UU-pre because it determines values that are not part of the user model in AHA!. Another such strange rule is the update of *P.visited* in the IU phase which is done in that phase to avoid order dependence in the UU-pre phase where this rule really belongs. These observations seem to indicate that our choice to divide the execution into an IU, UU-pre, GA and UU-post phase (and no other phases) may be too severe. (It is an arbitrary choice anyway.) A more natural representation of AHA!, with fewer tricks to avoid order dependence, would be possible if AHAM simply suggested to use an arbitrary number of phases (instead of four).

Chapter 7

Concluding Remarks

In this chapter we summarize the results of the previous chapters and give some pointers for future research.

7.1 Conclusions

In this dissertation we have studied the field of adaptive hypermedia. We started in Chapter 1 by asking five research questions about this research area. We will now summarize our findings and give the answers that we came up with in the different chapters of this dissertation.

Question 1: How do hypertext systems (and applications) attempt to personalize the information?
In Chapter 2 we studied the history of hypermedia, and found out that true personalization does not exist in hypermedia. It first appeared in adaptive hypermedia systems in the early 1990's.

Question 2: How can adaptive hypermedia systems solve the problems of (non-adaptive) hypermedia systems?
In Chapter 2 we discussed the methods and techniques used in adaptive hypermedia, and concluded that personalization is offered in two ways:

- **adaptive presentation**: this solves the problem that users may not understand a page they can navigate to; missing foreknowledge is compensated by inserting additional explanations.

- **adaptive navigation support**: this solves the problem that users may not know which of the available links to follow in order to reach their goals without encountering pages that are not relevant or that they cannot understand; the link anchors are adaptively annotated, hidden or sorted to guide users to the most appropriate information.

In order to perform such adaptation the systems maintain a user model that is updated every time the user interacts with the system.

Question 3: Can we describe the functionality of AHS at an abstract level? Can we describe adaptive hypermedia systems as extensions to hypermedia systems in general?
In Chapter 3 we introduced our reference model, called AHAM (Adaptive Hypermedia Application Model). AHAM describes the functionality of Adaptive Hypermedia Systems (AHS) at an abstract level. AHAM is an extension of the Dexter model for hypermedia systems. As a result AHAM describes adaptive hypermedia as an extension of hypermedia systems in general.

AHAM decomposes an adaptive hypermedia application into a domain model (DM), a user model (UM) and an adaptation model (AM).

- The DM supports composite concepts and concept relationships. A concept hierarchy describes the conceptual structure of the application domain. The concept relationships describe how concepts fit together. One concept may be a *prerequisite* for another concept for instance. The concept relationships play an important role in providing adaptation to users. Prerequisite relationships, for instance, can be used to suggest or even enforce a certain reading order.

- The UM supports all kinds of user features, e. g. knowledge, goals, background, hyperspace experience, preferences, interests and individual traits. The UM keeps a user's browsing history information which can be used to provide adaptation. Stereotyping and overlay techniques are commonly used to structure and initialize the UM.

- The AM provides the strategies to perform actual adaptation. It describes the updates to the UM and the generation of adaptation. We have given examples of adaptation rules and an adaptation rule langauge, but AHAM does not impose a certain rule language. In fact, it does not even require that an AHS is uses rule-based adaptation.

Question 4: Can we define behavioral semantics for AHS and analyze how AHS work exactly?
A reference architecture for adaptive hypermedia applications should describe how adaptation really works in AHS. In Chapter 4 we have defined an Adaptation Engine (AE) and studied its properties. To illustrate how adaptation works in AHS, we have studied three aspects:

- We have defined a rule language, AHAM-CA, based on a subset of *condition-action* rules as studied in the field of active databases.

- We have defined the AHAM-CA rule execution model that explains how rules can activate and deactivate each other.

- We have discussed termination and confluence problems with AHAM-CA rules. We have developed a theory for performing conservative static analysis (at authoring

time) of a given rule set, and a theory of constraints to guarantee termination and confluence.

Question 5: Can we easily describe the adaptation functionality of some well known existing AHS in our model?
In Chapters 5 and 6 we have described the adaptation functionality of two well-known AHS: InterBook and AHA!. We have described the production version of InterBook, with limited functionality, and the production version 1.0 of AHA!. Newer versions of these systems, with richer functionality, exist. Our description of InterBook and AHA! shows that we can describe the adaptation in AHAM. It also shows that an exact description of a fairly simple AHS already leads to a fairly complicated set of adaptation rules. Several parts of the description of InterBook and AHA! are somewhat similar. Understanding the AHAM description of one system is probably helpful for understanding the AHAM description of another AHS.

7.2 Future work

This dissertation provides a framework to understand and compare AHS, and to build well-behaved new AHS. There are several possible directions to extend this work. One is to formalize the description of adaptation functions at a more formal abstract level (e. g. using Z, Object-Z or some other formal specification language).

Another direction is to extend the use of AHAM to more application areas and a more advanced adaptation level. The following are possible extensions:

- To broaden the use of AHAM to application areas which contain "context" of the information domain, we need to incorporate a description of the "context" of the information domain to AHAM, perhaps by building a context model (CM). The AM can then provide adaptation based on the DM, UM, and CM.

- To allow AHAM to support very large information applications where the DM is unknown or partly unknown, the system needs to include functions to handle dynamic parts of the DM.

- To make the use of adaptive hypermedia applications more convenient, the system needs to provide advanced navigation support, e. g. link-independent navigation support and cross-reference navigation support. We have published some research in this direction in [Wu and De Bra, 2002; Wu and de Kort, 2002]. The system needs functions to present composite concepts, e. g. by showing an index page of a composite concept (either generated on-the-fly or beforehand).

- To have more ways to trace a user's behavior, the system needs to deal with many different input events. In AHAM we have concentrated on the "follow link" operation, but many other events or metrics can be added to gather information.

Another possible direction is to build authoring tools to facilitate the creation of (well-behaved) adaptive hypermedia applications. To make authoring easy, we published some ideas for authoring support in [Wu et al., 1999a]. We give the following wish list as an example for authoring support:

- AHAM
  - Tools for creating and maintaining the DM, UM, and AM are needed. Especially the separation of these three parts during the authoring process is missing in the current generation of systems.
  - Tools in the AM consist of two parts: tools for adaptation and tools for maintaining or extending the AM. The tools for adaptation should provide several adaptation patterns (for content adaptation and link adaptation). More research is needed on tools for defining page selectors and page constructors.
- AE
  - Authoring tools need to implement and apply the algorithms for verifying sufficient conditions for termination and confluence. They need to be conservative but as accurate as possible in order to avoid giving too many false warnings.
  - Authoring tools need to allow authors to select constraints to impose on the rule sets that can be created. The author can thereby select the termination and confluence "dangers" the system will automatically guard against.

Bibliography

M. Agosti, M. Melucci, and F. Crestani. Automatic authoring and construction of hypermedia for information retrieval. *ACM Multimedia Systems*, 3:15–24, 1995.

A. Aiken, J. M. Hellerstein, and J. Widom. Static analysis techniques for predicting the behavior of active database rules. *ACM Transactions on Database Systems*, 20(1):3–41, 1995.

A. Aiken, J. Widom, and J. M. Hellerstein. Behavior of database production rules: Termination, confluence and observable determinism. In *Proc. of the ACM SIGMOD International Conference on Management of Data, San Diego, CA, USA*, pages 59–68, June 1992.

R. M. Akscyn, D. L. McCracken, and E. A. Yoder. KMS: A distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM*, 31(7):820–835, 1988.

L. Ardissono and A. Goy. Tailoring the interaction with users in electronic shops. In *Proc. 7th International Conference on User Modeling (UM'99), Banff, Canada*, pages 35–44, June 1999.

R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. WebWatcher: A learning apprentice for the world wide web. In *Proc. of AAAI Spring Symposium on Information Gathering, Stanford University, CA, USA*, pages 6–12, March 1995.

F. Asnicar and C. Tasso. ifWeb: A prototype of user modelbased intelligent agent for document filtering and navigation in the World Wide Web. In *Proc. of Workshop on Adaptive Systems and User Modeling on the World Wide Web at UM'97 Conference, Chia Laguna, Sardinia, Italy*, pages 3–12, June 1997.

J. Bailey. *On the Foundations of Termination Analysis of Active Database Rules.* PhD thesis, Department of Computer Science, University of Melbourne, Australia, 1997.

M. Balabanovic and Y. Shoham. Fab: Content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

E. Baralis. *An Algebraic Approach to the Analysis and Optimization of Active Database Rules.* PhD thesis, Politecnico di Torino, Torino, Italy, 1994.

E. Baralis and J. Widom. An algebraic approach to rule analysis in expert database systems. In *Proc. of International Conference on Very Large Databases (VLDB'94), Santiago, Chile*, pages 475–486, September 1994. URL:http://citeseer.nj.nec.com/baralis94algebraic.html.

E. Baralis and J. Widom. An algebraic approach to static analysis of active database rules. *ACM Transactions on Database Systems*, 25(3):269–332, 2000.

I. Beaumont. User modeling in the interactive anatomy tutoring system ANATOM-TUTOR. *User Modeling and User-Adapted Interaction*, 4(1):21–45, 1994.

M. Bernstein, J. D. Bolter, M. Joyce, and E. Mylonas. Architectures for volatile hypertext. In *Proc. of the ACM Conference on Hypertext (Hypertext'91), San Antonio, TX, USA*, pages 243–260, December 1991.

A. C. Bertoletti and C. da Rocha Costa. SAGRES - A virtual museum. In *Proc. of Museums and WebConference, New Orleans, LA, USA*, March 1999. URL:http://www.archimuse.com/mw99/papers/bertoletti/bertoletti.html.

D. Billsus, M. J. Pazzani, and J. Chen. A learning agent for wireless news access. In *Proc. of International Conference on Intelligent User Interfaces (IUI'00), New Orleans, LA, USA*, pages 33–36, January 2000.

H. Böcker, H. Hohl, and T. Schwab. pADAPTer: Individualizing hypertext. In *Proc. of Human Computer Interaction (INTERACT'90), Amsterdam, The Netherlands*, pages 931–936, August 1990.

G. A. Boy. On-line user model acquisition in hypertext documentation. In *Proc. of Workshop on Agent Modeling for Intelligent Interaction at IJCAI'91 Conference, Sydney, Australia*, pages 34–42, 1991.

C. Boyle and A. O. Encarnacion. MetaDoc: An adaptive hypertext reading system. *User Modeling and User-Adapted Interaction*, 4(1):1–19, 1994.

P. J. Brown. Turning ideas into products: The guide system. In *Proc. of the ACM Conference on Hypertext (Hypertext'87), Chapel Hill, NC, USA*, pages 33–40, November 1987.

P. Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3):87–129, 1996. URL:http://citeseer.nj.nec.com/brusilovsky96methods.html.

P. Brusilovsky. Adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 11: 87–110, 2001.

P. Brusilovsky and D. W. Cooper. ADAPTS: Adaptive hypermedia for a Web-based performance support system. In *Proc. of Workshop on Adaptive Systems and User Modeling on World Wide Web at WWW'99 Conference, Toronto, Canada*, pages 41–47, May 1999.

P. Brusilovsky and J. Eklund. A study of user model based link annotation in educational hypermedia. *Journal of Universal Computer Science*, 4(4):429–448, 1998. `URL:http//citeseer.nj.nec.com/161601.html`.

P. Brusilovsky, J. Eklund, and E. Schwarz. Web-based education for all: A tool for development adaptive courseware. *Computer Networks and ISDN Systems*, 30(1-7):291–300, 1998.

P. Brusilovsky and L. Pesin. ISIS-Tutor: An intelligent learning environment for CDS/ISIS users. In *Online Proc. of Interdisciplinary Workshop on Complex Learning in Computer Environments (CLCE'94), Joensuu, Finland*, May 1994. `URL:http://cs.joensuu.fi/~mtuki/www_clce.270296/Brusilov.html`.

P. Brusilovsky and L. Pesin. Visual annotation of links in adaptive hypermedia. In *Proc. of Short Papers: Agents and Anthropomorphism of ACM Conference on Human Factors in Computing Systems (CHI'95), Denver, CO, USA*, pages 222–223, May 1995.

P. Brusilovsky, L. Pesin, and M. Zyryanov. Towards an adaptive hypermedia component for an intelligent learning environment. In *Human Computer Interaction*, volume 753 of *LNCS*, pages 348–358. 1993.

P. Brusilovsky, E. Schwarz, and G. Weber. ELM-ART: An intelligent tutoring system on world wide web. In *Proc. of International Conference on Intelligent Tutoring Systems (ITS'96), Montreal, Canada*, pages 261–269, June 1996a.

P. Brusilovsky, E. Schwarz, and G. Weber. A tool for developing adaptive electronic textbooks on world wide web. In *Proc. of World Conference of the WWW, Internet, and Intranet (WebNet'96), San Francisco, CA, USA*, pages 64–69, October 1996b.

P. Brusilovsky, E. Schwarz, and G. Weber. World-Wide intelligent textbooks. In *Proc. of World Conference on Educational Telecommunications, Boston, MA, USA*, pages 302–307, June 1996c.

P. Brusilovsky and G. Weber. Collaborative example selection in an intelligent example-based programming environment. In *Proc. of International Conference on Learning Sciences (ICLS'96), Evanston, IL, USA*, pages 357–362, July 1996.

P. Brusilovsky and M. Zyryanov. Intelligent tutor, environment and manual for physical geography. In *Proc. of International PEG Conference, Edinburgh, Scotland, UK*, pages 63–73, July 1993.

P. D. Bruza. Hyperindices: A novel aid for searching in hypermedia. In *Proc. of the ACM European Conference on Hypertext (ECHT'90), Paris, France*, pages 109–122, November 1990.

V. Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, 1945.

B. Campbell and J. M. Goodman. HAM: A general purpose hypertext abstract machine. *Communications of the ACM*, 31(7):856–861, 1988.

R. Carro, E. Pulido, and P. Rodriguez. TANGOW: Task-based adative learner guidance on the world wide web. In *Proc. of Workshop on Adaptive System and User Modeling on World Wide Web at WWW'99 Conference, Toronto, Canada*, pages 49–57, May 1999.

S. Ceri and J. Widom. Deriving production rules for constraint maintenance. In *Proc. of International Conference on Very Large Data Bases (VLDB'90), Brisbane, Queensland, Australia*, pages 566–577, August 1990.

D. T. Chang. HieNet: A user-centered approach for automatic link generation. In *Proc. of the ACM Conference on Hypertext (Hypertext'93), Seattle, WA, USA*, pages 145–158, November 1993.

K. Clibbon. Conceptually adapted hypertext for learning. In *Proc. of the ACM Conference on Human Factors in Computer Systems (CHI'95), Denver, CO, USA*, pages 224–225, May 1995.

P. De Bra, A. Aerts, G. J. Houben, and H. Wu. Making general-purpose adaptive hypermedia work. In *Proc. of World Conference of the WWW, Internet, and Intranet (WebNet'00), San Antonio, TX, USA*, pages 117–123, October 2000.

P. De Bra and L. Calvi. Creating adaptive hyperdocuments for and on the web. In *Proc. of World Conference of the WWW, Internet, and Intranet (WebNet'97), Toronto, Canada*, pages 149–155, October 1997.

P. De Bra and L. Calvi. AHA!: A generic adaptive hypermedia system. In *Proc. of Workshop on Adaptive Hypertext and Hypermedia (Hypertext'98), Pittsburgh, PA, USA*, pages 20–24, June 1998a.

P. De Bra and L. Calvi. AHA! an open adaptive hypermedia architecture. *The New Review of Hypermedia and Multimedia*, 4:115–139, 1998b.

P. De Bra and L. Calvi. Towards a generic adaptive hypermedia system. In *Proc. of Workshop on Adaptive Hypertext and Hypermedia at Hypertext'98 Conference, Pittsburgh, PA, USA*, pages 5–11, June 1998c.

P. De Bra, G. J. Houben, and H. Wu. AHAM: A Dexter-based reference model for adaptive hypermedia. In *Proc. of the ACM Conference on Hypertext and Hypermedia (Hypertext'99), Darmstadt, Germany*, pages 147–156, February 1999.

P. De Bra, G.J. Houben, and Y. Kornatzky. An extensible data model for hyperdocuments. In *Proc. of the ACM Conference on Hypertext (Hypertext'92), Milan, Italy*, pages 222–231, November 1992.

B. de Carolis, F. de Rosis, C. Andreoli, V. Cavallo, and M. de Cicco. The dynamic generation of hypertext presentations of medical guidelines. *The New Review of Hypermedia and Multimedia*, 4:67–88, 1998.

B. de La Passardiere and A. Dufresne. Adaptive navigational tools for educational hypermedia. *Computer Assisted Learning*, 602:555–567, 1992.

F. de Rosis, B. de Carolis, and S. Pizzutilo. User tailored hypermedia explanations. In *Proc. of INTERCHI'93 Conference (CHI'93 and INTERACT'93) , Amsterdam, The Netherlands*, pages 169–170, April 1993.

N. Delisle and M. Schwartz. Neptune: A hypertext system for CAD applications. In *Proc. of ACM SIGMOD, Washington DC, USA*, pages 132–142, May 1986.

L. DeYoung. Linking considered harmful. In *Proc. of the ACM European Conference on Hypertext (ECHT'90), Paris, France*, pages 238–249, November 1990.

H. Dieterich, U. Malinowski, T. Kuhme, and M. Schneider-Hufschmidt. State of the art in adaptive user interfaces. *Computing and Informatics*, 18:357–366, 1993.

C. Eliot, D. Neiman, and M. Lamar. Medtec: A Web-based intelligent tutor for basic anatomy. In *Proc. of World Conference of the WWW, Internet, and Intranet (WebNet'97), Toronto, Canada*, pages 161–165, October 1997.

L. M. Encarnacao. Adaptivity in graphical user interfaces: An experimental framework. *Computers & Graphics*, 19(6):873–884, 1995.

D. C. Engelbart. A conceptual framework for the augmentation of man's intellect. *Vistas in Information Handling*, 1:1–29, 1963.

J. Fink, A. Kobsa, and A. Nill. Adaptable and adaptive information provision for all users, including disabled and elderly people. *The New Review of Hypermedia and Multimedia*, 4:163–188, 1998.

G. Fischer, T. Mastaglio, B. Reeves, and J. Riemann. Minimalist explanations in knowledge-based systems. In *Proc. of Annual Hawaii International Conference on System Sciences, Kailua-Kona, HI, USA*, pages 309–317, January 1990.

T. Fox, G. Grunst, and K.J. Quast. HyPlan - a contextsensitive hypermedia help system. In *Report 743: Arbeitspapiere der GMD, GMD, Germany*, 1993.

L. Francisco-Revilla and F. Shipman. Adaptive medical information delivery: Combining user. In *Proc. of International Conference on Intelligent User Interfaces (IUI'00), New Orleans, LA, USA*, pages 94–97, January 2000.

X. Fu, J. Budzik, and K. J. Hammond. Mining navigation history for recommendation. In *Proc. of International Conference on Intelligent User Interfaces (IUI'00), New Orleans, LA, USA*, pages 106–112, January 2000.

G. W. Furnas. Generalized fisheye views. In *Proc. of the ACM Conference on Human Factors in Computing Systems (CHI'86), New York, NY, USA*, pages 16–23, April 1986.

R. Furuta and P. D. Stotts. The Trellis hypertext reference model. In *Proc. of NIST Hypertext Standardization Workshop, Gaithersburg, MD, USA*, pages 83–93, January 1990.

S. Garlatti, S. Iksal, and P. Kervella. Adaptive on-line information system by means of a task model and spatial views. In *Proc. of Workshop on Adaptive Systems and User modeling on World Wide Web at WWW'99 Conference, Toronto, Canada*, pages 59–66, May 1999.

K. Gates, P. Lawhead, and D. Wilkins. Toward an adaptive WWW: A case study in customized hypermedia. *The New Review of Hypermedia and Multimedia*, 4:89–113, 1998.

S. Geldof. Con-textual navigation support. *The New Review of Hypermedia and Multimedia*, 4:47–66, 1998.

J. E. Gilbert and C. Y. Han. Arthur: Adapting instruction to accommodate learning style. In *Proc. of World Conference of the WWW, Internet, and Intranet (WebNet'99), Honolulu, HI, USA*, pages 433–438, October 1999.

M. Gonschorek and C. Herzog. Using hypertext for an adaptive helpsystem in an intelligent tutoring system. In *Proc. of World Conference on Artificial Intelligence in Education (AI-ED'95), Washington DC, USA*, pages 274–281, August 1995.

J. Greer and G. McCalla. *Student Modeling: The key to Individualized Knowledge-Based Instruction*, volume 125 of *NATO ASI Serie F.* Springer Verlag, 1991.

K. Grønbæk and R. H. Trigg. Design issues for a Dexter-based hypermedia system. *Communications of the ACM*, 37(2):40–49, 1994.

G. Grunst. Adaptive hypermedia for support systems. *Adaptive User Interfaces: Principles and Practice*, pages 269–283, 1993.

F. Halasz. Reflections on NoteCards: Seven issues for the next hypermedia systems. *Communications of the ACM*, 31(7):836–852, 1988.

F. Halasz, T. P. Moran, and R. H. Trigg. NoteCards in a nutshell. In *Proc. of the ACM conference on Human Factors in Computing Systems abd Graphics Interface (CHI+GI'87), Toronto, Canada*, pages 45–52, April 1987.

F. Halasz and M. Schwartz. The Dexter hypertext reference model. In *Proc. of the NIST Hypertext Standardization Workshop, Gaithersburg, MD, USA*, pages 95–133, January 1990.

F. Halasz and M. Schwartz. The Dexter hypertext reference model: Hypermedia. *Communications of the ACM*, 37(2):30–39, 1994.

L. Helmes, M. Razum, and A. Barth. Concept of a hypertext interface for the information retrieval in complex factual databases. In *Hypertext - Information Retrieval - Multimedia (HIM'95), University of Konstanz, Germany*, pages 175–189, April 1995.

N. Henze, K. Naceur, W. Nejdl, and M. Wolpers. Adaptive hyperbooks for constructivist teaching. *KI - Kunstliche Intelligenz*, 13(4):26–31, 1999.

T. Hirashima, N. Matsuda, T. Nomoto, and J. Toyoda. Context-sensitive filtering for browsing in hypertext. In *Proc. of International Conference on Intelligent User Interfaces (IUI'98), San Francisco, CA, USA*, pages 119–126, January 1998.

T. Hirashima, N. Matsuda, T. Nomoto, and J. Toyoda. Context-sensitive filtering for browsing Web pages. In *Proc. of World Conference of the WWW, Internet, and Intranet (WebNet'99), Honolulu, HI, USA*, pages 294–295, October 1999.

C. Hockemeyer, T. Held, and D. Albert. RATH - A relational adaptive tutoring hypertext WWW-environment based on knowledge space theory. In *Proc. of International Conference on Computer Aided Learning in Science and Engineering (CALISCE'98), Göteborg, Sweden*, pages 417–423, June 1998.

H. Hohl, H. D. Böcker, and R. Gunzenhäuser. Hypadapter: An adaptive hypertext system for exploratory learning and programming. *User Modeling and User-Adapted Interaction*, 6(2-3):131–156, 1996.

K. Höök, J. Karlgren, A. Wærn, N. Dahlbäck, C.G. Jansson, K. Karlgren, and B. Lemaire. A glass box approach to adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3):157–184, 1996.

K. Höök, A. Rudstrom, and A. Wærn. Edited adaptive hypermedia: Combining human and machine intelligence to achieve filtered information. In *Proc. of Workshop on Flexible Hypertext at Hypertext'97 Conference, Southampton, UK*, pages 54–58, April 1997.

J. Hothi and W. Hall. An evaluation of adapted hypermedia techniques using static user modelling. In *Proc. of Workshop on Adaptive Hypertext and Hypermedia at Hypertext'98 Conference, Pittsburgh, PA, USA*, pages 45–50, June 1998.

T. Joachims, D. Freitag, and T. Mitchell. WebWatcher: A tour guide for the WWW. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI'97), Nagoya, Japan*, pages 770–775, August 1997.

T. Joerding. A temporary user modeling approach for adaptive shopping on the Web. In *Proc. of Workshop on Adaptive Systems and User Modeling on World Wide Web at WWW'99 Conference, Toronto, Canada*, pages 75–79, May 1999.

C. Kaplan, J. Fenwick, and J. Chen. Adaptive hypertext navigation based on user goals and context. *User Modeling and User-Adapted Interaction*, 3(3):193–220, 1993.

A. P. Karadimce and S. D. Urban. Conditional term rewriting as a formal basis for analysis of active database rules. In *Proc. of Workshop on Research Issues in Data Engineering, Active Database Systems, Houston, TX, USA*, pages 156–162, Febuary 1994.

M. Katsumoto, M. Fukuda, N. Irie, and Y. Shibata. Dynamic hypermedia system based on perceptional link method for distributed design image database. In *Proc. of International Conference on Information Networking (ICOIN-9), Osaka, Japan*, pages 49–54, December 1994.

M. Katsumoto, M. Fukuda, and Y. Shibata. The kansei link method for multimedia database. In *Proc. of International Conference on Information Networking (ICOIN-10), Kyung-Ju, Korea*, pages 382–389, January 1996.

J. Kay and B. Kummerfeld. Adaptive hypertext for individualised instruction. In *Proc. of Workshop on Adaptive Hypertext and Hypermedia at UM'94 Conference , Hyannis, MA, USA*, August 1994a. `URL:http://www.cs.bgsu.edu/hypertext/adaptive/Kay.html`.

J. Kay and B. Kummerfeld. An individualised course for the C programming language. In *Proc. of International World Wide Web Conference (WWW'94), Chicago, IL, USA*, October 1994b. `URL:http://www.ncsa.uiuc.edu/SDG/IT94/Inproceedings/Educ/kummerfeld/kummerfeld.h%tml`.

M. Kayama and T. Okamoto. Hy-SOM: The semantic mape framwork applied on an example case of navigation. In *Proc. of International Conference on Computers in Education (ICCE'99), Chiba, Japan*, pages 252–259, November 1999.

R. Keller, S. Wolfe, J. Chen, J. Rabinowitz, and N. Mathe. A bookmarking service for organizing and sharing URLs. In *Proc. of International World Wide Web Conference (WWW'97), Santa Clara, CA, USA*, April 1997. `URL:http://www.scope.gmd.de/info/www6/technical/paper189/paper189.html`.

D. Kim. WING-MIT: Das auf einer multimedialen und intelligenten benutzerschnittstelle basierende tutorielle hilfesystem. In *WING-IIR Technical Report 69, University of Regensburg, Germany*, 1995.

A. Kobsa, D. Müller, and A. Nill. KN-AHS: An adaptive hypertext client of the user modeling system BGP-MS. In *Proc. of International Conference on User Modeling (UM'94), Hyannis, MA, USA*, pages 99–106, August 1994.

A. J. Kok. A review and synthesis of user modelling in intelligent systems. *The Knowledge Engineering Review*, 6(1):21–47, 1991.

A. Kushniruk and H. Wang. A hypermedia-based educational system with knowledgebased guidance. In *Proc. of World Conference on Educational Multimedia and Hypermedia (ED-MEDIA'94), Vancouver, Canada*, pages 335–340, June 1994.

P. Lai and U. Manber. Flying through hypertext. In *Proc. of the ACM Conference on Hypertext (Hypertext'91), San Antonio, TX, USA*, pages 123–132, December 1991.

D. B. Lange. A formal model of hypertext. In *Proc. of NIST Hypertext Standardization Workshop, Gaithersburg, MD, USA*, pages 145–166, January 1990.

M. Laroussi and M. Benahmed. Providing an adptive learning through the web case of CAMELEON: Computer aided medium for learning on networks. In *Proc. of International conference on Computer Aided Learning and Instructon in Science and Engineering (CALISCE'98), Göteborg, Sweden*, pages 411–416, June 1998.

W. S. Li, Q. Vu, D. Agrawal, Y. Hara, and H. Takano. PowerBookmarks: A system for personalizable Web information organization, sharing, and management. In *Proc. of International World Wide Web Conference (WWW'99), Toronto, Canada*, pages 297–311, May 1999.

H. Lieberman. Letizia: An agent that assists web browsing. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI'95), Montreal, Canada*, pages 924–929, August 1995.

A. Lippman. Movie maps: An application of the optical videodisc to computer graphics. *Computer Graphics*, 14(3):32–43, 1980.

M. Marinilli, A. Micarelli, and F. Sciarrone. A case-based approach to adaptive information filtering for the WWW. In *Proc. of Workshop on Adaptive Systems and User modeling on World Wide Web at WWW'99 Conference, Toronto, Canada*, pages 81–87, May 1999.

C. C. Marshall and F. M. Shipman. Spatial hypertext: Designing for change. *Communications of the ACM*, 38(8):88–97, 1995.

N. Mathe and J. Chen. A user-centered approach to adaptive hypertext based on an information relevance model. In *Proc. of International Conference on User Modeling (UM'94), Hyannis, MA, USA*, pages 107–114, August 1994.

N. Mathe and J. R. Chen. User-centered indexing for adaptive information access. *User Modeling and User Adapted Interaction*, 6(2-3):225–261, 1996.

N. Meyrowitz. Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework. In *Proc. of Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'86), Portland, OR, USA*, pages 186–201, September 1986.

A. Micarelli and F. Sciarrone. A case-based toolbox for guided hypermedia navigation. In *Proc. of International Conference on User Modeling (UM'96), Kailua-Kona, HI, USA*, pages 129–136, January 1996.

M. Milosavljevic. Augmenting the user's knowledge via comparison. In *Proc. of International Conference on User Modelling (UM'97), Chia Laguna, Sardinia, Italy*, pages 119–134, June 1997.

M. Milosavljevic and J. Oberlander. Dynamic hypertext catalogues: Helping users to help themselves. In *Proc. of the ACM Conference on Hypertext (Hypertext'98), Pittsburgh, PA, USA*, pages 123–131, June 1998.

D. Mladenic. Personal WebWatcher: Implementation and design. In *Technical Report IJS-DP-7472, Department of Intelligent Systems, J. Stefan Institute, Slovenia*, 1996.

M. Montebello, W. A. Gray, and S. Hurley. A personal evolvable advisor for WWW knowledge-based systems. In *Proc. of Workshop on Reuse of Web Information at WWW'98 Conference, Brisbane, Australia*, pages 59–69, April 1998.

S. Mukherjea, J. D. Foley, and S. E. Hudson. Interactive clustering for navigating in hypermedia systems. In *Proc. of the ACM European Conference on Hypertext (ECHT'94), Edinburgh, Scotland, UK*, pages 136–145, September 1994.

T. Murray, C. Condit, and E. Haugsjaa. Metalinks: A preliminary framework for concept-based adaptive hypermedia. In *Proc. of Workshop WWW-Based Tutoring a ITS'98 Conference, San Antonio, TX, USA*, June 1998. `URL:http://www-aml.cs.umass.edu/~stern/webits/itsworkshop/murray.html`.

A. Negro, V. Scarano, and R. Simari. User adaptivity on world wide web through CHEOPS. In *Proc. of Workshop on Adaptive Hypertext and Hypermedia at Hypertext'98 Conference, Pittsburgh, PA, USA*, pages 57–62, June 1998.

T. Nelson. A file structure for the complex, the changing, and the indeterminate. In *Proc. of the ACM National Conference, Cleveland, OH, USA*, pages 84–100, August 1965.

G. Neumann and J. Zirvas. SKILL - a scallable internet-based teaching and learning system. In *Proc. of World Conference of the WWW, Internet, and Intranet (WebNet'98), Orlando, FL, USA*, pages 688–693, November 1998. `URL:http://nestroy.wi-inf.uni-essen.de/Forschung/Publikationen/skill-webnet98.ps`.

S. C. Newell. User models and filtering agent for improving internet information retrieval. *User Modeling and User-Adaptive Interaction*, 7(4):223–237, 1997.

J. Nielsen. *Hypertext and Hypermedia*. Academic Press, 1990.

E. G. Noik. Exploring large hyperdocuments: Fisheye views of nested networks. In *Proc. of the ACM Conference on Hypertext (Hypertext'93), Seattle, WA, USA*, pages 192–205, November 1993.

E. Not, D. Petrelli, M. Sarini, O. Stock, C. Strapparava, and M. Zancanaro. Hypernavigation in the physical space: Adapting presentations to the user and to the situational context. *New Review of Multimedia and Hypermedia*, 4:33–45, 1998.

J. Oberlander, M. O. Donnell, A. Knott, and C. Mellish. Conversation in the museum: Experiments in dynamic hypermedia with the intelligent labelling explorer. *The New Review of Hypermedia and Multimedia*, 4:11–32, 1998.

R. Oppermann and M. Specht. Adaptive information for nomadic activity. a process oriented approach. In *Proc. of Software Ergonomie, Walldorf, Germany*, pages 255–264, March 1999.

J. Paredaens, P. Peelman, and L. Tanca. Merging graph based and rule based computation. In *Proc. of International Workshop on Rules in Database Systems (RIDS'93), Edinburgh, Scotland, UK*, pages 211–233, August 1993.

F. Paterno and C. Mancini. Designing Web user interfaces adaptable to different types of use. In *Proc. of Museums and Web Conference, New Orleans, LA, USA*, March 1999. `URL:http://www.archimuse.com/mw99/papers/paterno/paterno.html`.

M. J. Pazzani, J. Muramatsu, and D. Billsus. Syskill & Webert: Identifying interesting Web sites. In *Proc. of National Conference on Artificial Intelligence (AAAI'96), Portland, OR, USA*, pages 54–61, August 1996.

T. Perez. HyperTutor: From hypermedia to intelligent adaptive hypermedia. In *Proc. of World Conference on Educational Multimedia and Hypermedia (ED-MEDIA'95), Graz, Austria*, pages 529–534, June 1995.

T. Perez, J. Gutirrez, and P. Lopisteguy. An adaptive hypermedia system. In *Proc. of World Conference on Artificial Intelligence in Education (AI-ED'95), Washington DC, USA*, pages 351–358, August 1995.

D. Pilar da Silva. Concepts and documents for adaptive hypermedia: A model and a prototype. In *Proc. of Workshop on Adaptive Hypertext Hypermedia at Hypertext'98 Conference, Pittspurgh, PA, USA*, pages 33–40, June 1998.

V. Quint and I. Vatton. Combining hypertext and structured documents in grif. In *Proc. of the ACM European Conference on Hypertext (ECHT'92), Milano, Italy*, pages 23–32, November 1992.

J. Rucker and M. J. Polanco. Siteseer: Personalized navigation for the Web. *Communications of the ACM*, 40(3):73–76, 1997.

V. Schoch, M. Specht, and G. Weber. ADI - An empirical evaluation of a tutorial agent. In *Proc. of the World Conference on Education Multimedia and Hypermedia (ED-MEDIA'98), Freiburg, Germany*, June 1998. URL:http://apsymac33.uni-trier.de:8080/ADI.html.

I. Schwab, W. Pohl, and I. Koychev. Learning to recommend from positive evidence. In *Proc. of International Conference on Intelligent User Interfaces (IUI'00), New Orleans, LA, USA*, pages 241–247, January 2000.

Y. Shibata and M. Katsumoto. Dynamic hypertext and knowledge agent systems for multimedia information networks. In *Proc. of the ACM Conference on Hypertext (Hypertext'93), Seattle, WA, USA*, pages 82–93, November 1993.

O. Signore, R. Bartoli, and G. Fresta. Tailoring Web pages to user's needs. In *Proc. of Workshop on Adaptive Systems and User Modeling on the World Wide Web at UM'97 Conference, Chia Laguna, Sardinia, Italy*, June 1997. URL:http://www.contrib.andrew.cmu.edu/~plb/UM97_workshop/Signore/Signore.html.

L. Simon and J. Erdmann. SIROG - A responsive hypertext manual. In *Proc. of the ACM European Conference on Hypertext (ECHT'94), Edinburgh, Scotland, UK*, pages 108–116, September 1994.

M. Specht and R. Opermann. ACE - adaptive courseware environment. *The New Review of Hypermedia and Multimedia*, 4:141–161, 1998.

M. Specht, G. Weber, S. Heitmeyer, and V. Schoch. AST: Adaptive WWW-courseware for statistics. In *Proc. of Workshop on Adptive Systems and User Modeling on the World Wide Web at UM'97 Conference, Chia Laguna, Sardinia, Italy*, pages 91–95, June 1997.

J. M. Spivey. *The Z Notation: A reference manual*. Prentice Hall International Series in Computer Science. Prentice-Hall, Inc., 1989.

A. Stefani and C. Strapparava. Exploiting NLP techniques to build user model for Web site: The use of WordNet in SiteIF. In *Proc. of Workshop on Adaptive Systems and User modeling on the World Wide Web at WWW'99 Conference, Toronto, Canada*, pages 95–100, May 1999.

A. Steinacker, C. Seeberg, K. Reichenbacher, S. Fischer, and R. Steinmetz. Dynamically generated tables of contents as guided tours in aadaptive hypermedia systems. In *Proc. of World Conference on Education Multimedia and Hypermedia (ED-MEDIA'99), Seattle, WA, USA*, pages 640–645, June 1999.

C. Süß, B. Freitag, and P. Brössler. Metamodeling for Web-based teachware managment. In *Proc. of Workshop on World Wide Web and Conceptual Modeling (ER'99), Paris, France*, pages 360–373, November 1999.

C. Thomas and G. Fischer. Using agents to improve the usability and usefulness of the world wide web. In *Proc. of International Conference on User Modeling (UM'96), Kailuna-Kona, HI, USA,*, pages 5–12, January 1996.

C. G. Thomas. BASAR: A framework for integrating agents in the world wide web. *IEEE Computer*, 28(5):84–86, 1995.

A. van Dam. Hypertext'87 keynote address. *Communications of the ACM*, 31(7):887–895, 1987.

M. H. van de Voort. *A Design Theory for Database Triggers*. PhD thesis, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, 1994.

J. van Ossenbruggen. *Processing Structured Hypermedia - A Matter of Style*. PhD thesis, Vrije University, Amsterdam, The Netherlands, 2001.

J. Vassileva. A practical architecture for user modeling in a hypermedia-based information system. In *Proc. of International Conference on User Modeling (UM'94), Hyannis, MA, USA*, pages 115–120, August 1994.

J. Vassileva. A task-centered approach for user modeling in a hypermedia office documentation system. *User Modeling and User-Adapted Interaction*, 6(2-3):185–224, 1996.

J. H. Walker. Document examiner: Delivery interface for hypertext documents. In *Proc. of the ACM Conference on Hypertext (Hypertext'87), Chapel Hill, NC, USA*, pages 307–323, November 1987.

J. A. Waterworth. A pattern of islands: Exploring public information space in a private vehicle. In *Proc. of Multimedia, Hypermedia and Virtual Reality (MHVR'94), Moscow, Russia*, pages 265–278, April 1994.

G. Weber. ART-WEB, 1999. `URL:http://www.psychologie.uni-trier.de:8080/projects/ELM/elm.html`.

G. Weber and M. Specht. User modelling and adaptive navigation support in WWW-based tutoring systems. In *Proc. of International Conference on User Modeling (UM'97), Chia Laguna, Sardinia, Italy*, pages 289–300, June 1997.

R. Weiss, B. Velez, M. A. Sheldon, C. Namprempre, P. Szilagyi, A. Duda, and D. K. Gifford. HyPursuit: A hierarchical network search engine that exploits content-link hypertext clustering. In *Proc. of the ACM Conference on Hypertext (Hypertext'96), Washington, DC, USA*, pages 180–193, March 1996.

A. Wesley. *HyperCard Stack Design Guidelines*. Apple Computer Inc, 1989.

J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, Inc., 1996.

E. Wilson. Links and structures in hypertext databases for law. In *Proc. of the ACM European Conference on Hypertext (ECHT'90), Paris, France*, pages 195–211, November 1990.

P. Wright. Cognitive overheads and prostheses: Some issues in evaluating hypertext. In *Proc. of the ACM Conference on Hypertext (Hypertext'91), San Antonio, TX, USA*, pages 1–12, December 1991.

H. Wu and P. De Bra. Suficient conditions for well-behaved adaptive hypermedia systems. In *Proc. of Internation Conference on Web Intelligence, Maebashi, Japan*, pages 148–152, October 2001.

H. Wu and P. De Bra. Link-independent navigation support in web-based adaptive hypermedia. In *Proc. of Web-Engineering Track at WWW'02 Conference, Honolulu, HI, USA*, May 2002. `URL:http://www2002.org/CDROM/alternate/684/`.

H. Wu, P. De Bra, A. Aerts, and G. J. Houben. Adaptation control in adaptive hypermedia systems. In *Proc. of International Conference on Adaptive Hypermedia and Adaptive Web-based Systems (AH'00), Trento, Italy*, pages 250–259, August 2000.

H. Wu and E. de Kort. Cross-references in web-based adaptive hypermedia. In *Proc. of Workshop on Personalization in Electronic Publishing at AH'02, Malaga, Spain*, pages 45–55, May 2002.

H. Wu, E. de Kort, and P. De Bra. Design issues for general-purpose adaptive hypermedia systems. In *Proc. of ACM Conference on Hypertext and Hypermedia (Hypertext'01), Aarhus, Danmark*, pages 141–150, August 2001.

H. Wu, G. J. Houben, and P. De Bra. Authoring support for adaptive hypermedia applications. In *Proc. of World Conference on Education Multimedia and Hypermedia (ED-MEDIA'99), Seattle, WA, USA*, pages 354–369, June 1999a.

H. Wu, G. J. Houben, and P. De Bra. User modeling in adaptive hypermedia applications. In *Proc. of the Interdisciplinaire Conferentie Informatiewetenschap, Amsterdam, The Netherlands*, pages 10–21, November 1999b.

R. Zeiliger. Adaptive testing: Contribution of the SHIVA model. In *Item Banking: Interactive Testing and Self-Assessment*, volume 112 of *NATO ASI Serie F*, pages 54–65. 1993.

Y. Zhou and M. Hsu. A theory for rule triggering systems. In *Proc. of Extending Database Technology (EDBT'90), Berlin, Germany*, pages 407–421, March 1990.

List of Figures

Index

Summary

Web-based hypermedia systems are becoming increasingly popular as tools for user-driven access to information. They typically offer users a lot of freedom to navigate through a large hyperspace. Unfortunately, this rich link structure of the hypermedia applications causes some serious usability problems: navigation problems and comprehension problems. Adaptive hypermedia systems (or AHS for short) aim at overcoming these problems by providing adaptive navigation support and adaptive content. The adaptation (or personalization) is based on a user model that represents relevant aspects of the user such as preferences, knowledge and interests. The system gathers information about the user by observing the use of the application, and in particular by observing the browsing behavior of the user.

This dissertation provides a reference architecture for adaptive hypermedia applications. It describes the adaptation functionality of adaptive hypermedia applications at an abstract level, using a adaptive hypermedia application model (AHAM), and at an implementation level by using an adaptation engine (AE).

Chapter 1 introduces the research questions and approaches and the outline of this dissertation.

Chapter 2 gives an overview of the literature and research issues in the areas of hypermedia and adaptive hypermedia.

Chapter 3 provides a reference model for adaptive hypermedia applications, called AHAM. AHAM should help authors and designers to more easily understand AHS (Adaptive Hypermedia Systems) and be able to compare different AHS. AHAM is an extension of the Dexter reference model for hypermedia applications. AHAM keeps the standard functions of the Dexter model and concentrates on the adaptation part.

AHAM decomposes an adaptive hypermedia application into a domain model (DM), a user model (UM) and an adaptation model (AM).

- The DM supports composite concepts and concept relationships. The concepts form a hierarchical structure, containing high level abstract concepts, concrete page concepts, and small information fragments. Concept relationships are semantic relationships between concepts.

- The UM supports all kinds of user features, e. g., knowledge, goals, background, hyperspace experience, preferences, interests and individual traits. We can use a combination of stereotyping and overlay modeling techniques to model the user fea-

tures. In this dissertation we mainly focus on a simple way of following a link to trace the users' behavior, but AHAM has no built-in limits on how trace users' features. It can easily be extended to other forms of interaction with the user.

- The AM describes the adaptation strategies. It specified how to update the UM and generate the adaptation. As an example we use rules to describe adaptation strategies in this dissertation, but AHAM does not impose rule-based adaptation onto the AM.

Adaptive hypermedia applications consist of AHAM and an Adaptation Engine (AE). The AE is a software environment that performs the actual adaptation. Different systems may have different ways to realize the adaptation. Chapter 4 describes an abstract AE by separating the adaptation description from adaptation techniques. It discusses the functions of the AE, system transactions, termination and confluence problems, and well-behaved AE. To illustrate how adaptation works in AHS, an abstract AE is described by defining a rule language, AHAM-CA, defining the rule execution model and discussing termination and confluence problems therein.

In studying of the behavior of adaptive hypermedia systems, focus lies on the above-mentioned design issues for the rule systems: termination and confluence. The problem of termination is to decide whether the rule execution is guaranteed to terminate. Confluence means that under the same conditions, i. e. the same domain model and same user model instance, the same action always results in the same presentation and the same user model updates. Every AHS expects to have these two properties. Research results for active database are used to perform termination and confluence analysis. These results can even be improved upon by applying domain knowledge of AHS.

Aside from termination and confluence there is also the issue of efficiency, or "fast" determination of termination and confluence, both at authoring time and at run-time. The detection algorithm has an exponential time complexity in general. For that reason sufficient conditions to guarantee termination and confluence for simple adaptive hypermedia applications are proposed. The complexity of the detection algorithm then falls to quadratic order in the number of rules.

Finally, to validate our reference architecture for AHS, we use the above defined AE to describe two existing well known AHS, InterBook (Chapter 5) and AHA! (Chapter 6).

Samenvatting

Inter- en intranet gebaseerde hypermedia applicaties, ook wel web-applicaties genoemd, winnen nog altijd aan populariteit als hulpmiddel voor toegang tot informatie. Zij bieden hun gebruikers veel bewegingsvrijheid in een grote informatieruimte. Helaas leidt de rijk verbonden structuur van deze hypermedia applicaties tot serieuze problemen met de bruikbaarheid: navigatie door een ondoorgrondelijk web van verbindingen. Adaptieve hypermedia systemen (AHS) pogen deze problemen aan te pakken door middel van adaptieve navigatie ondersteuning en adaptieve inhoud. Het adaptieve aspect en de aanpassingen aan persoonlijke wensen zijn gebaseerd op een gebruikersmodel dat de relevante aspecten van een gebruiker representeert zoals voorkeuren, kennis en interesse. Het systeem verzamelt informatie over de gebruiker door het gebruik van de applicatie te analyseren en in het bijzonder het navigatiegedrag.

Deze dissertatie beschrijft een referentie architectuur voor adaptieve hypermedia applicaties. De adaptieve functionaliteit van de AHS wordt op een abstract niveau beschreven met behulp van AHAM (Adaptive Hypermedia Application Model), terwijl op implementatieniveau een adaptieve kernapplicatie (AE, voor Adaptive Engine) beschreven wordt.

Hoofdstuk 1 introduceert de ondezoeksvragen, de aanpak van het onderzoek en een schets van de verdere inhoud van deze dissertatie.

Hoofdstuk 2 bevat een overzicht van de literatuur en onderzoeksproblemen op het gebied van hypermedia en adaptieve hypermedia.

Hoofdstuk 3 beschrijft AHAM, een referentiemodel voor adaptieve hypermedia applicaties. Dit model heeft tot doel auteurs en ontwerpers in staat stellen om gemakkelijker AHS te doorgronden en beter in staat te zijn verschillende AHS te vergelijken. AHAM is een uitbreiding van het Dexter referentiemodel voor hypermedia applicaties. De standaardfunctionaliteit van het Dextermodel wordt gebruikt en uitgebreid met functies die adaptieve applicaties moeten bezitten.

AHAM splitst een adaptieve hypermedia applicatie in een domeinmodel (DM), een gebruikersmodel (UM, voor User Model) en een adaptatiemodel (AM).

- Het DM ondersteunt samengestelde concepten en (semantische) relaties tussen concepten. Tezamen vormt dit een hiërarchische structuur met hoog niveau abstracte concepten, concrete pagina concepten en kleine informatie fragmenten.

- Het UM bevat allerlei gebruikersinformatie, zoals kennis, (leer)doelen, ervaring, voorkeuren, interesses en individuële gewoonten. We kunnen met een combinatie van

stereotypering en overdekkings- (overlay) modelleringstechnieken de gebruikerskenmerken modelleren. Het zwaartepunt in deze dissertatie ligt vooral op de de relatief eenvoudige methode van het volgen van gebruikersnavigatie (het "klikgedrag"), maar AHAM heeft geen ingebouwde beperkingen met betrekking tot de gebruikerskenmerken die gevolgd worden. Andere vormen van interactie kunnen eenvoudig worden toegevoegd aan de hier gepresenteerde voorbeelden.

- Het AM beschrijft de adaptatiestrategieën; het specificeert hoe het UM moet worden bijgewerkt en hoe de pagina's moeten worden aangepast. Bij wijze van voorbeeld wordt een regel-taal gebruikt om de adaptieve strategieën te illustreren, maar merk op dat AHAM een dergelijketaal niet oplegt aan het AM.

Adaptieve hypermedia applicaties bestaan uit AHAM en een adaptieve kernapplicatie (AE). Deze AE is een stuk programmatuur dat de eigenlijke adaptatie uitvoert. Verschillende systemen kunnen verschillende AEs hebben om de adaptatie te verwezenlijken. Hoofdstuk 4 beschrijft een abstracte AE door de adaptatie specificatie te scheiden van de adaptatie technieken; de volgende aspecten komen aan de orde: de functies van een AE, systeem transacties, terminatie en confluentie problemen en gewenst gedrag van een AE. Ter illustratie wordt een voorbeeld AE beschreven. Hierin komt onder andere aan de orde het definiëren van een regel-taal, AHAM-CA, de definitie van het regel-uitvoeringsmodel en het bespreken van terminatie en confluentie problemen daarvan.

In de bestudering van het gedrag van adaptieve hypermedia systemen wordt speciaal aandacht geschonken aan de bovengenoemde ontwerpproblemen voor regelgebaseerde systemen: terminatie en confluentie. Het probleem van terminatie omvat het onderzoek naar de (on)mogelijkheid om te bepalen of een verzameling voorschiften gegarandeerd eindigd binnen een gegeven regelmodel. Confluentie is de eigenschap van een verzameling regels die garandeerd dat uitgaande van dezelfde begintoestand, dat bij dezelfde gebruikersactie dezelfde eindtoestand (presentatie en UM aanpassingen) bereikt wordt, uiteraard weer binnen een vast regelmodel. Ieder AHS moet aan deze twee voorwaarden voldoen. Onderzoeksresultaten voor actieve databases blijken zeer wel bruikbaar voor analyse van terminatie en confluentie. Deze resultaten kunnen zelfs nog enigszins worden verbeterd door de domeinkennis van het AHS te gebruiken.

Behalve terminatie en confluentie moet er ook rekening gehouden worden met de efficiëntie van berekenen, ofwel het "snel" vaststellen van terminatie- en confluentievoorwaarden, zowel in de ontwerpfase als in het actieve gebruik van een AHS. Het detectiealgoritme, geleend van het actieve database onderzoek, heeft in het slechste geval een exponentiëel tijdsgebruik. Om die reden worden beperkende voorwaarden gedefiniëerd die terminatie en confluentie afdwingen. Deze voorwaarden zijn alleen acceptabel voor betrekkelijk eenvoudige adaptieve hypermedia applicaties, maar hebben als voordeel dat de tijdscomplexiteit wordt teruggebracht tot kwadratische orde in het aantal regelinstanties.

Tot slot worden twee bekende AHS beschreven, met gebruikmaking van de beschreven AE, om de referentie architectuur voor AHS te valideren: InterBook (Hoofdstuk 5) and AHA! (Hoofdstuk 6).

Curriculum Vitae

Hongjing Wu was born on November 29, 1962, in Shanghai (China). She started her study in Computer Science at the Department of Computer Science, Harbin Shipbuilding Engineering Institute, Heilongjiang, China, in 1980. Having completed her B.Sc. degree after four years, she continued with the computer science study at the same university. She obtained her M.Sc. degree in 1987.

After her graduation Hongjing worked at the same university for eleven years. She worked there as a lecturer for five years, as an assistant professor for five years and as an associate professor for one year. Her major field for this period time was databases.

She spent one year as a visiting scholar at Parallel Scientific Computing and Simulation Group at the University of Amsterdam, the Netherlands. She started her Ph.D. research in 1998 at the Department of Computer Science, Eindhoven University of Technology, The Netherlands. Her Ph.D. research is on adaptive hypermedia. Her major research interests are web-based information systems, user modeling, databases, and adaptive hypermedia systems. Her email address is h.wu@tue.nl.