AHA! goes interbook and beyond

Santic, T.

Award date:
2003

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

MASTER'S THESIS

AHA! goes Interbook and beyond

by

T. Santic

Supervisors: prof. dr. P. Brusilovsky
prof. dr. P.M.E. De Bra

juni 2003

AHA! goes Interbook and beyond

Eindhoven University of Technology

Project : AHA! goes Interbook and beyond

Author : Tomislav Santic

Department : Department of Mathematics and Computer Science

Date : June 2003

Supervisors : Prof. Dr. Peter Brusilovsky
Prof. Dr. Paul De Bra,

Summary

Adaptive hypermedia is a relatively new area of research that became more popular in the last ten years when the first Web based adaptive hypermedia systems (AHS) were developed. Among these Web based adaptive hypermedia systems were Interbook, developed by Peter Brusilovsky and Adaptive Hypermedia For All developed at the Eindhoven University of Technology.

The original goal of the project was to extend AHA! to the point to be able to simulate Interbook's user interface. During the project, when the primary goal was achieved we decided to go one step further. The final results can be divided into two main parts:

- Generic user model for AHA!
- Interbook to AHA! Compiler

The first and most important outcome of this project is a new user interface model for AHA!. We tried to extend a general-purpose adaptive hypermedia (AH) architecture AHA! with flexible and rich interface by re-using some ideas explored in a more specific AH system InterBook. To address the lack of user interface possibilities in AHA! we have developed the *Layout model* that employs the strong points of Interbook user interface. The dynamic structures of the *Layout model* are easily extendible and give the author the power to adapt the user interface to the nature of the course.

The second outcome of the project is the '*Interbook to AHA!*' compiler. The compiler was developed to test the Layout model and to achieve total simulation of Interbook courses. With the new user interface model and the compiler AHA! is able to serve Interbook courses in Interbook style. The results of this project prove that AHA! indeed can serve as a generic adaptive hypermedia system not only on the conceptual level but on the user interface level as well.

Preface

This report is the result of a nine month research project partly completed at the University of Pittsburgh and party completed at the Eindhoven University of Technology. The main research area of the project was adaptive hypermedia, more specifically two adaptive hypermedia systems, Interbook and AHA!.

The great majority of the project was done at the University of Pittsburgh. A debt of gratitude is owned to my supervisor in Pittsburgh Prof. Dr. Peter Brusilovsky. He has guided me during this research project and helped me making important decisions. He is an open minded and very pleasant person to work with. I would also like to thank Prof. Dr. Paul De Bra for fulfilling my wish of doing my Master's thesis abroad and making this project possible. Finally I would like to thank University of Pittsburgh for the hospitality extended to me during my stay in Pittsburgh and the NLnet Foundation for financial support.

Eindhoven, June 2003

Tomislav Santic

Contents

Introduction

Numerous Web-based adaptive hypermedia systems have been developed within the last 10 years (Brusilovsky, 2001). These systems all have different "look and feel" and offer different ways of adaptation. Yet, behind this diversity an expert can find a reasonably limited set of methods and techniques (Brusilovsky, 1996; De Bra, Brusilovsky & Houben, 1999a). A major motivation behind the AHA! project (De Bra & Calvi, 1998) was developing a flexible adaptive hypermedia architecture that can be used for implementing a wide variety of adaptation methods. AHA! was created as an "assembly language" of adaptive hypermedia in the sense that any higher-level adaptation paradigm can be expressed in terms of AHA! and simulated by the AHA! engine. The most recent AHA! version (De Bra et al., 2002) has shown to be very powerful in this respect. As was recently demonstrated, a reasonably advanced adaptation paradigm implemented in the InterBook system (Brusilovsky, Eklund & Schwarz, 1998) can be almost completely simulated by AHA! (De Bra, Houben & Wu, 1999b; Wu, De Kort & De Bra, 2001). Yet, with all this power, AHA! is not completely ready to serve as a universal simulator for an arbitrary AH system.

The problem is that each AH system is characterized not only by its unique model of adaptation, but also by its unique interface. While the current version of AHA! is quite ready to simulate the adaptation behavior of any AH system, simulating their interface is a lot more difficult. The AHA! engine operates in a single-window mode - in a good tradition of classic hypertext systems. In contrast, most of modern adaptive Web-based hypermedia systems use rich multi-frame and multi-window interfaces. InterBook is a good example here. It uses several multi-frame windows (textbook, glossary, and table of contents). An example of InterBook's Textbook and Glossary windows is provided on Figure 1. Other advanced AH systems also use complex multi-frame windows (Grigoriadou et al., 2001; Henze & Nejdl, 2001; Melis et al., 2001; Weber & Brusilovsky, 2001). Creating such multi-frame and/or multi-window interfaces in AHA! requires that the author define the frame structure and use Javascript code to synchronize the presentation in all the frames.

The goal of the project was to resolve the problem by developing a flexible interface model for the AHA! engine. Following the idea of AHA! that can be used to describe a variety adaptation functionalities, we wanted to develop an interface model that is used to describe a variety of Web adaptive hypermedia interfaces. The primary goal of our project was reasonably modest: we wanted to extend the AHA! engine to the extent where it can simulate the InterBook adaptation mechanism and its multi-frame interface. In doing so, we wanted to avoid narrow-minded solutions and hacks (like the Javascript hack previously used with AHA!) developing a reasonably universal approach that can be used to implement the InterBook interface along with many other interfaces.

We will start with an introduction about Hypermedia and Adaptive Hypermedia systems. After that we will describe the problematic start of the project and introduce the

background for our work, we will give a brief introduction of the InterBook and AHA! interfaces, followed by extensions on the conceptual level and our first attempt for user interface extensions. After that we present the most important outcome of the project, the Layout Model that extends AHA! and demonstrate how it can be used to simulate the InterBook interface in AHA!. Finally we describe the 'Interbook to AHA!' compiler that we have developed to test the flexibility of the Layout model and to achieve complete simulation of Interbook courses.

Hypermedia

Traditional text documents provide a linear organization of information. There is one particular order in which the information should be retrieved. This kind of information organization is not sufficient for a great number of applications. It is hard to determine the best reading order for the information that is not organized in a sequential way. People have different styles of learning and information gathering. Therefore there is a great variety of reading orders. This is where hypermedia/hypertext fits in.
Hypermedia is an acronym which combines the words *hypertext* and *multimedia.* Shneiderman (Shneiderman, 1989) defines hypertext/ hypermedia as *"a database that has active cross-references and allows the reader to "jump" to other parts of the database as desired".*
A Hypermedia system (HS) consists of Hyper-documents. Hyper-documents are nodes that contain embedded links to other documents (De Bra, 1998). The user can use these embedded links to navigate to other nodes (documents) to retrieve the required information. This architecture allows the information in the HS to be accessed in a non-linear way. In the last 40 years number of Hypermedia Systems had been developed (Xanadu, ZOG, Aspen Movie Map, Hypercard etc.), but the real breakthrough came with World Wide Web (WWW) which is by far the most popular example of a HS.

Adaptive Hypermedia

One of the biggest problems with Hypermedia Systems is the phenomenon called "Lost in Hyperspace". Hypermedia nodes usually contain a number of links to other nodes and the user can follow any of the presented links. This has a disadvantage that after a while the user can loose the orientation in Hyperspace and loose confidence in the system.

To address this and other problems in Hypermedia Systems, Adaptive Hypermedia Systems have been developed. The basis of Adaptive Hypermedia Systems is that the pages served by the system are adapted to user's needs.

Brusilovsky (Brusilovsky, 1996) defines Adaptive Hypermedia as:
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.

From this definition we can derive that in order to be an Adaptive Hypermedia System a Hypermedia System needs to keep the information about the user in some kind of user model and use this user model to adapt the pages served by the system to the user's needs. There are many systems available that can adapt the pages to the user, but not all of these systems can be called adaptive hypermedia systems. Yahoo for example does not use a user model to adapt the pages to user's changing wishes. The user can fill out a form before using the system which is then used to adapt the pages to his/hers needs, but the system is not capable of using any data about the user gathered on the fly.
An arbitrary AHS distinguishes three functional parts:

1. *Domain model*- contains the content of application including concept relationships
2. *User model* – contains the data about the user which can be used for page adaptation
3. *Adaptation rules* – set of rules that describe how the user model has to updated based on the user's actions during the working session

Internally adaptive hypermedia systems can use different architectures but functionally all AHS can be described using these three functional parts.

Adaptive Hypermedia Systems use a number of different techniques for page adaptation, however all these techniques can be divided in two main groups (Brusilovsky, 1996):

- Adaptive presentation (conditional including of fragments, fragments sorting, strechtext, etc.)
- Adaptive navigation (adaptive link annotation, adaptive link sorting, direct guidance, etc.)

The basis of adaptive presentation is that the actual content of the page is adapted to the user's knowledge, interests, goals etc. Pieces of text/media (fragments) are shown or not shown dependant on the user model. Adaptive navigation is mainly used to solve the "Lost in Hyperspace" problem. Links are usually annotated by a color or icons to give the user more information about the documents the links are referring to. There are also other techniques like link sorting where the most important links are shown at the top, or link hiding where links that are not recommended to follow at the moment are hidden or disabled. Adaptive navigation helps the user to navigate through the Domain Hyperspace in the most efficient way.

The project start

The start of the project was quite slow and problematic. An initial objective was porting Interbook to the PC. This 'mini project' was unrelated to the rest of the project. Interbook exists for some time but one of the things that is standing in the way of its proliferation is the fact that Interbook is only available for Macintosh platform. Interbook is implemented in Lisp and uses the Common LISP Hypermedia Server (CL-HTTP) as a Web server. In order to run Interbook on a PC two things needed to be done:

1. The CL-HTTP web server should be compiled and run on a PC using one of the Lisp compilers for the PC.
2. The Interbook source code should be compiled on a PC using one of the Lisp compilers for the PC.

Two well known Lisp programming environments for the PC are: Allegro Lisp and LispWorks. The problem was that we had neither of the two and therefore we were forced to try to do the job by using the free versions of these environments. After some trouble we finally managed to run CL-HTTP server on a PC.

The second task was more complex. Interbook uses a lot of MCL (Macintosh Lisp programming environment) specific symbols and functions. These symbols are part of the CCL package which is the MCL specific package thus not available in other Lisp implementations (LispWorks, Allegro CL,...). Lisp programs that do not use this MCL specific package can be more easily ported to PC/Windows. There are however two problems with porting a MCL program to PC/Windows:

1. Finding equivalents for the CCL symbols and functions in other Lisp implementations
2. GUI (specific Graphical User Interface code).

Interbook uses CCL specific symbols and functions, which are only available on the Macintosh, in 190 different places. There are some duplicates but the fact is that Interbook heavily relies on the CCL package and to port it to PC/Windows all these CCL symbols should be replaced by their LispWorks/AllegroCL equivalents. There is not something like a standard list of CCL symbols equivalents for other Lisp implementations which means that for the CCL symbols used in Interbook equivalents should be found on the case by case basis. Porting a GUI implemented in MCL to Windows/PC also seems to be a (standard) problem.

It was obvious that porting Interbook to the PC would be a very time-consuming matter and that it could endanger the completion of the rest of the project. Therefore we decided to continue further with the main part of the project which is extending AHA! with Interbook features.

The InterBook Interface Paradigm

Introduction

InterBook is a tool for authoring and delivering adaptive electronic textbooks (ET) on the World Wide Web. It has been developed by Peter Brusilovsky and Elmar Schwarz in 1996. Interbook supports adaptive navigation but no adaptive presentation is supported. The user is being guided through the course domain hyperspace by annotating the links (using colored icons) to glossary concepts and text pages. Interbook supports a number of features that can be turned on and off by the user. The system comes in different flavors. Dependent on the user's level a simple or more complicated user interface is used. Interbook supports the "Incremental Interface" technique where new user interface features are being introduced by the system dependent on the user knowledge. The internal architecture of Interbook is based on a specific concept-based approach. The basis of this approach is that the course domain is expressed as a set of concepts where the user is gathering the knowledge about these concepts by reading text pages served by the system. Every text page introduces knowledge about certain concepts. In turn the readiness attribute (whether the user should read the page or not) of text pages is influenced by the "background concepts". Background concepts are concepts that need to be familiar to the user in order to understand the content of the text page.
One of the strong points of Interbook is the authoring process. It is relatively easy to prepare a course in Interbook. The author designs a course in the form of a specially structured Word file. This hierarchically structured file is being transformed during a number of transformation steps into a Interbook format file which is a valid HTML file with some special Interbook codes. Another strong point of Interbook is its rich user interface.

Interbook user interface paradigm

InterBook has two main kinds of windows - a Textbook window (left on Figure 1) and a Glossary Window (right on Figure 1). These windows correspond to two major kinds of information items supported by InterBook - a book page and a domain knowledge concept. Each window in InterBook can include multiple links to concepts and pages. A click on any page link causes the linked page to be loaded in the Textbook window. A click on any concept link causes the information about the linked concept to be loaded in the glossary Window.

Despite of its complicated interface, InterBook attempted to support a simple metaphor - one window shows one and only one information item - i.e., a textbook window shows exactly one page of a textbook at a time. While each of these two windows includes several frames, they are considered not as independent windows, but as multiple views on

the same concept or page. For example: the text frame (bottom left) presents the text of the page; the navigation bar (top) presents the location of the current page among its ancestors and siblings, and the concept bar (bottom right) presents prerequisite and outcome concepts for the current page. All four frames of the textbook window are updated at the same time. Technically, a link to a textbook page is called a whole *page frameset* to be loaded into the textbook window. This frameset, in turn, pulls several frames associated with the requested page. The frameset approach is simple to understand and also works well with most browsers' standard way of navigation using back and forward buttons and history.

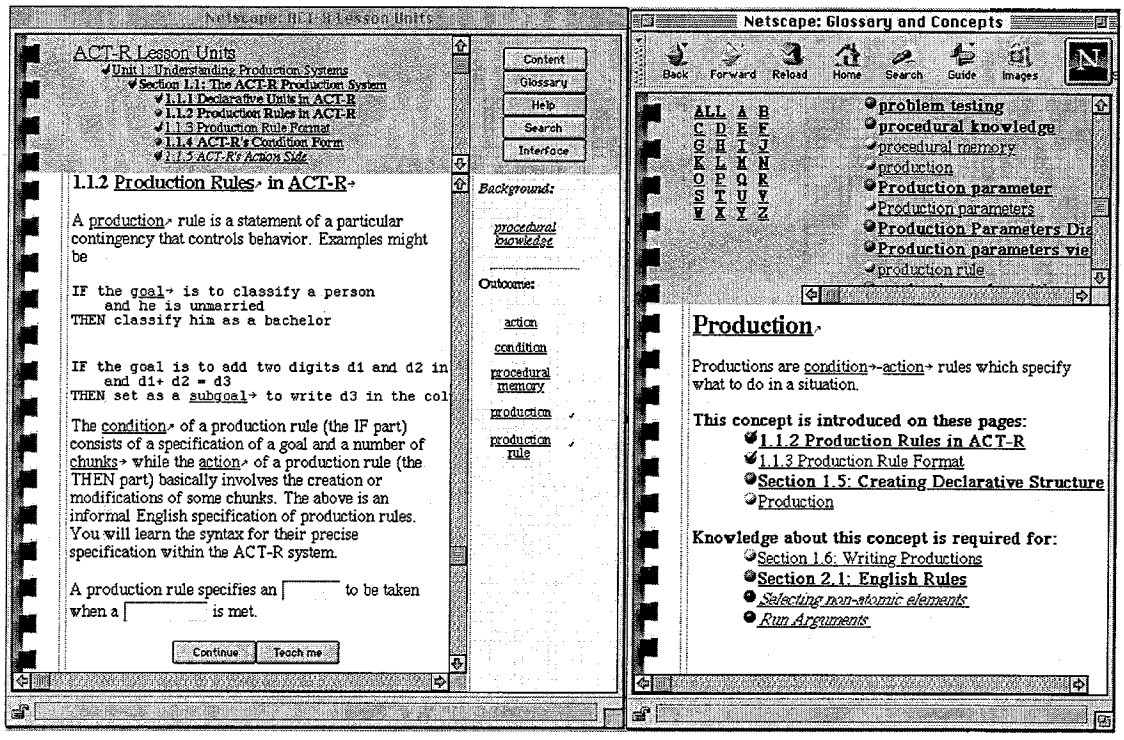


Figure 1: InterBook interface

The AHA! Interface Paradigm

Introduction

The first version of the Adaptive Hypermedia Architecture (AHA!) was released in 2000. AHA! 1.0 differed from other existing AHS because of its simplicity and its generic approach. Soon AHA! received the name of being an "assembler language" of adaptive hypermedia. AHA! can be seen as a low level language for developing of all kinds of adaptive hypermedia applications. In the meantime AHA! evolved from version 1.0 to version 2.0. Version 3.0 is being developed at the moment. AHA! is partly based on AHAM (Adaptive Hypermedia Architecture Model) model (De Bra, Houben & Wu, 1999b) which serves as a reference model for the description of AH applications. AHA! supports only the low level adaptation methods presented in Brusilovsky's taxonomy (Brusilovsky, 1996):

- Adaptive hiding or link annotation
- Conditional inclusion of fragments

Adaptive link annotation is supported by the AHA! engine in the form of colored links. The color of a link depends on the user model where a certain color means that it is recommended for the user to follow the link and some other color means that the user is still not ready to read the document the link is referring to. Adaptive hiding is implemented in the same way by using almost the same color for the link as for the regular text. This makes the link difficult to be noticed. Adaptive presentation is supported in the form of conditional fragments. The author can attach conditions to certain fragments of the text page. This enables the AHA! engine to conditionally show pieces of the page dependent on the user's knowledge about the domain model.

The basis of AHA!'s internal structure is its concept structure. AHA! 'sees' everything as a concept. Text pages presented to the user are generated using the concept's resource element. This element refers to a XHTML (or XML) resource file which is (after processing by the AHA! engine) presented to the user. The AHA! engine uses the user model and the internal concept model during the resource processing to adapt the content of the resource file to the user.

AHA!'s internal structure makes no real separation of Domain model and Adaptation rules. Concepts are saved as XML files that describe the concepts behavior and their relations with other concepts. The AHA! engine can also use a MySQL database to store the concept information, but XML version of the database is being used by default. AHA!'s user interface is quite simple.

AHA! User interface paradigm

AHA! was initially created to add adaptation to the course on hypermedia at the Eindhoven University of Technology (http://wwwis.win.tue.nl/2L690/). This course predates the general availability of frames in browsers. The course was therefore written using a single frame layout. The browser showed one course page at a time, with adapted links and conditionally included fragments. AHA! also added an optional header (with a progress report) and footer (with copyright statement). Header and footer were created by the author as html fragments. In Figure 7 a page with header can be seen (albeit embedded in the new Interbook-like multi-frame environment). Multi-frame applications are possible in AHA!. Figure 2 shows part of the multi-frame interface paradigm used in a course on graphical user interfaces.

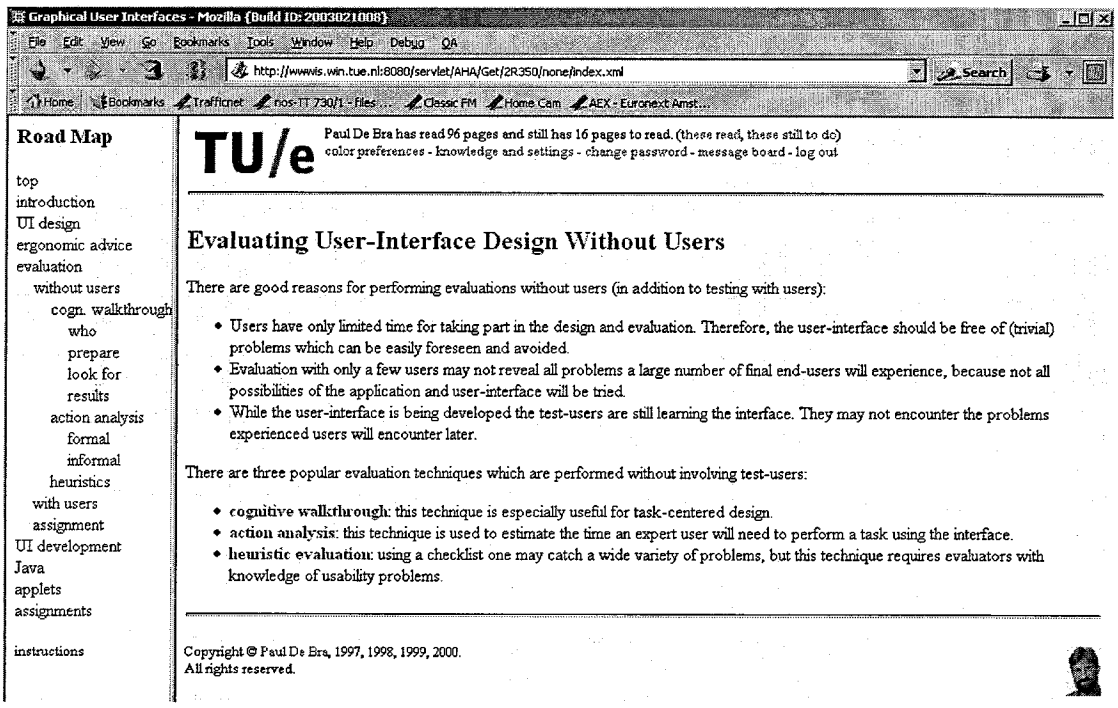


Figure 2: AHA! multi-frame interface

In order to make this interface work in AHA!, every page must include the following piece of Javascript code:

```
<script language="JavaScript">
parent.frames[0].location="content.xml"
```

```
</script>
```

The result is that when a link to a page is followed the leftmost frame is reloaded. It contains the "content.xml" file, which is a navigation menu in which submenus are conditionally shown, based on which page is displayed in the rightmost frame. The access to a page and the subsequent access to the menu are treated separately by AHA!. Whereas in Interbook following a link requests a complete frameset from the server in AHA! following a link requests a page, to be shown in a complete browser window or inside a frame. The AHA! engine does not "know" about a possible use of frames.

Bringing Interbook and AHA! together

The original goal of this project was extending AHA!'s user interface to the point to be able to simulate Interbook's user interface paradigm. In order to simulate Interbook's user interface AHA!'s user interface needs to support multiple windows and multiple frames. More specifically AHA! needs to support frames and two kinds of windows used by Interbook. Interbook uses different frames to represent different aspects of the concept. We can see these frames that contain some information about concepts as different views of course domain.

AHA! extensions on the conceptual level

Interbook' s user interface paradigm and AHA!'s user interface paradigm are quite different. However in order to simulate Interbook not only user interface needed to be improved but also number of things on the conceptual level.

Hierarchy

Interbook has a hierarchical concept structure. A number of views that are used in Interbook use this hierarchy to present some information to the user. The table of contents is probably the best example. It shows the whole hierarchy as a tree of annotated links to text pages. Another example is the Navigation bar where the path from root node to current node is shown also as a tree of annotated links to text pages. In contrast, AHA!'s internal concept structure is not hierarchical. Therefore before extending AHA!'s user interface with views(frames) that use internal concept hierarchy first we needed to introduce hierarchy to AHA! on the conceptual level. Using the following structure for each concept in hierarchy tree we introduced relations between concepts that enable the AHA! engine to construct an internal concept hierarchy:

```
<hierarchy>
    <firstchild> </firstchild>
    <nextsib> </nextsib>
    <parent> </parent>
 </hierarchy>
```

There are three kinds of hierarchical relations between concepts: *firstchild*, *nextsib* and *parent*. The hierarchy tree can also be constructed with only two relations (firstchild and nextsib), but using one extra relation makes the job much easier for the views that use the hierarchy. Figure 3 is showing graphically the hierarchy relations between concepts.

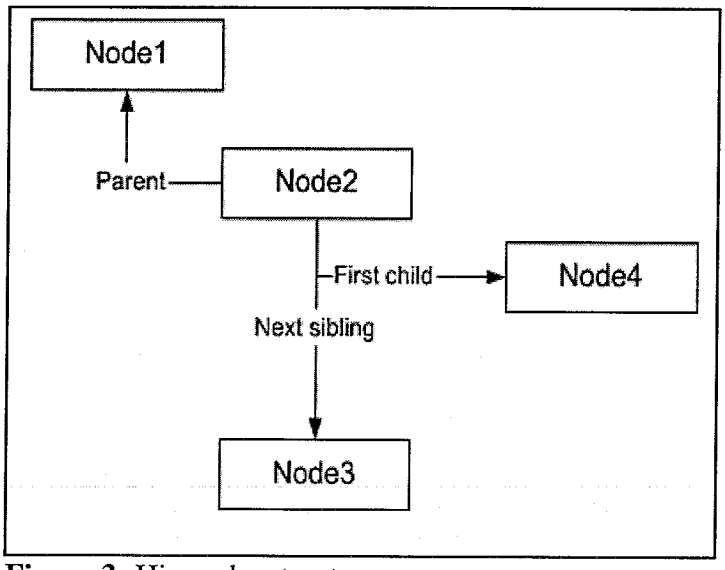


Figure 3: Hierarchy structure

Title

Another extension on the conceptual level is the introduction of titles. Old AHA! courses use the concept name field to identify concept. This field is also used for showing links to concepts. This system is rather inconvenient when Interbook courses, or courses of any other system, have to be served by the AHA! engine (see chapter 'Interbook to AHA! compiler'). Most of these systems use a computer generated concept ID that is not suitable for displaying to the user. Another advantage of separating concept title and concept id is that this system reduces chances for making mistakes.

Concepttypes

The last extension we made on the conceptual level is the introduction of concept types. In AHA! everything is a concept and all concepts are the same. There is no way of making a distinction between concepts. Every concept is treated in the same way. This is not the case in all adaptive hypermedia systems. Some systems treat different types of concepts in a different way. Interbook is a good example of such a system. Therefore we added a simple but powerful mechanism to the AHA! engine. By introducing the '*Concepttype*' field to the concept structure we enabled the AHA! engine to distinguish any number of different kinds of concepts. There are however no predefined concept types. The author of the course can define any desired number of concept types and the engine can treat these concepts in a different way. Chapter 'The View-Based Layout Model' gives an example how concept types are used in the Layout model.

AHA! user interface extension-first attempt

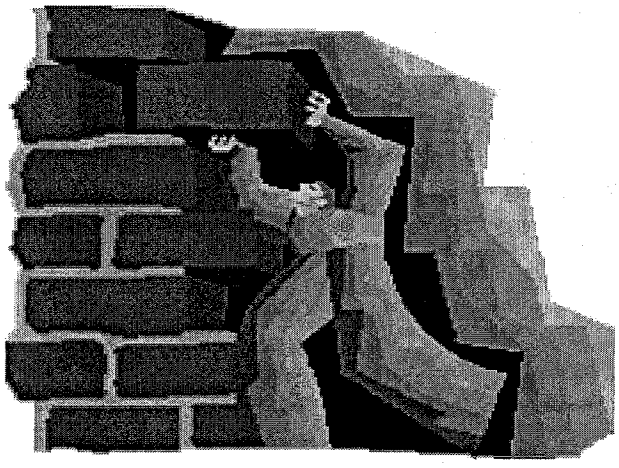
After extensions made on the conceptual level the introductory work for the extension of AHA!'s user interface was completed. We could start with the introduction of Interbook's frames and windows to AHA!. We started with the implementation of the Table of Contents view. The AHA! engine was already extended with hierarchy structures so what we needed to do is to add a frame within the AHA! window which would show the whole hierarchy structure. At first we did the job in a hard coded way, but soon we realized that this kind of narrow minded solutions would stand in a way for the future developments of AHA!. We decided to develop a more generic solution where it would be possible to rearrange views within the window and also place some views in separate windows. Our first solution was somewhere between hard coded and generic. The author was able to configure and rearrange the views in an XML configuration file, but there were still too many hard coded things in views. It was not possible to define any number of window divided in any number of frames. Views still contained too much logic and too much programming was needed to introduce views and to make everything work. While introducing new Interbook views to AHA! we were confronted with number of practical problems. At that moment we decided to go all the way and develop a real generic user interface that could simulate the user interface of any existing Web-based adaptive hypermedia system. Views would be simple objects with one task: displaying data from underlying data structures. Introduction of new views should be quite simple. It should also be possible to define any number of windows divided into any number of frames and linking, view updating and all other logic should be part of the AHA! engine. Finally we came up with the *View-Based Layout Model* which is described in the next chapter.

The View-Based Layout Model

The View-Based Layout Model is the new way in AHA! to present concepts (pages) to the user that was developed to address the lack of user interface possibilities in the earlier versions of the architecture. The *Layout model* combines the strong points of InterBook's rich user interface with the flexibility and customization style that are typical for the AHA!-architecture. This model allows every AH system developer to adapt the user interface to the course nature (without the need for the above mentioned Javascript hack).

To provide a high level of flexibility, the Layout Model was designed as a three-level interface model that is based on the concepts *views*, *viewgroups* and *layouts*. In brief, *views* are considered as atomic interface elements. Views can be grouped in *viewgroups*. One or more viewgroups form a *layout*. These concepts are presented in more detail below.

Views



Views are pieces of information about the course domain. They usually represent some relevant information about the active concept (the concept the user is viewing at the moment). A view can also represent some static information about the course domain. Views are used as pre-fabricated building blocks to construct the user interface for some specific course. Internally views are simply Java classes that generate HTML pages using underlying AHA! data structures. To present a concept to the user the system uses a set of predefined views. These predefined views can be further customized by the author of the course to develop an interface that meets the needs of the course. The author defines and customizes a view using an XML-based description language like the following:

```
<view name="v5" type="ToolboxView" title="Toolbox"
background="IBookbluesq.bmp"  params="leftspace=70">
      <secwnds>
            <secwnd link="TOC" viewgroup="TOC" img="ContentBtn.bmp"/>
            <secwnd link="Glossary" viewgroup="Glossary"
img="GlossaryBtn.bmp"/>
      </secwnds>
</view>
```

At the moment we have already implemented a number of relatively simple basic views. The configuration of these views consists of setting the background picture, the title or changing the page margin to make the view more readable. It is possible however to implement much more complex views that will offer much more tuning possibilities to the author. We are considering parameters that will influence the content of the view page and not only the shape of it.
A view usually displays some information about the active concept including links to other relevant concepts. However a view can also contain links to other views which will offer more information to the user about the active concept. Following one of these links will result in displaying a new set of views. Views that are used directly to represent different aspects of a concept are called *primary views*. Views that present some supplementary information are called *secondary views*. These views are not visible until they are triggered by a link in one of the primary views. The author of the course will usually choose the most important views as primary views and less important views as secondary views. The connection between primary and secondary views can be specified by the author of the course in the XML view structures presented above. In the presented example *ToolboxView* can trigger two secondary viewgroups: *Table of Content* and *Glossary*

Viewgroups and Layouts

As explained above views are the building blocks for constructing a concept representation. Views can be grouped in *viewgroups*. In HTML terms a *viewgroup* corresponds to an independent window and a view corresponds to a page that can be shown in a separate window or in an HTML frame. A set of *viewgroups* forms a *concept layout,* which is used to present a concept. Practically, it means that different aspects of a concept can be presented in several synchronized windows.

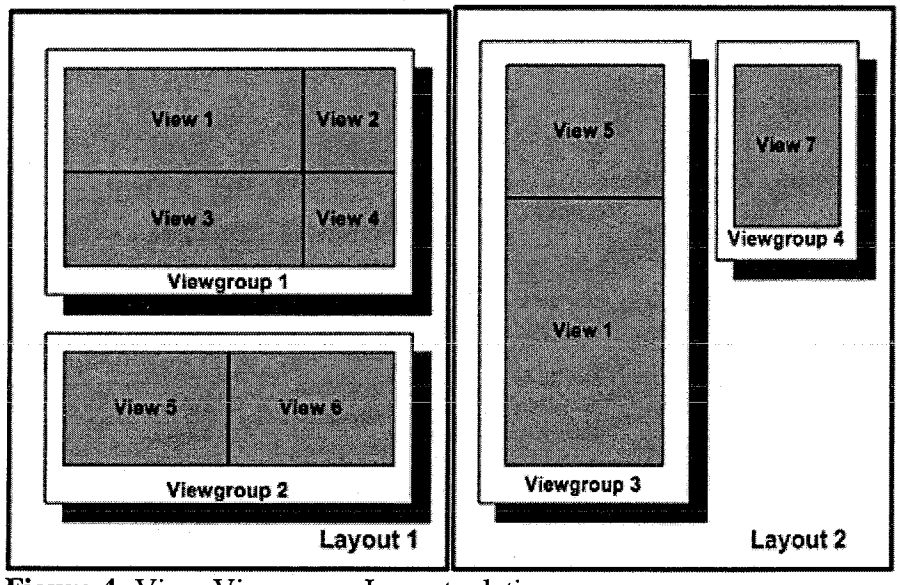


Figure 4: View-Viewgroup-Layout relation

Figure 4 shows examples of two different layouts. The first layout, Layout 1, contains two viewgroups(windows), Viewgroup 1 and Viewgroup 2, which use four and two views respectively. Layout 2 contains two Viewgroups, Viewgroup 3 and Viewgroup4, which use two and one views respectively.

For our simulation of Interbook courses we use three different layouts. Figure 5 shows views used for representation of text pages in Interbook courses. The views are grouped into three Viewgroups of which one Viewgroup with primary views and two Viewgroups with secondary views. The Viewgroup with primary views contains four views: Navigation bar view, Text view, Concept bar view and Toolbox view. Toolbox view can trigger Viewgroups with secondary views. Secondary Viewgroups contain each one view, Table of Contents and Glossary.

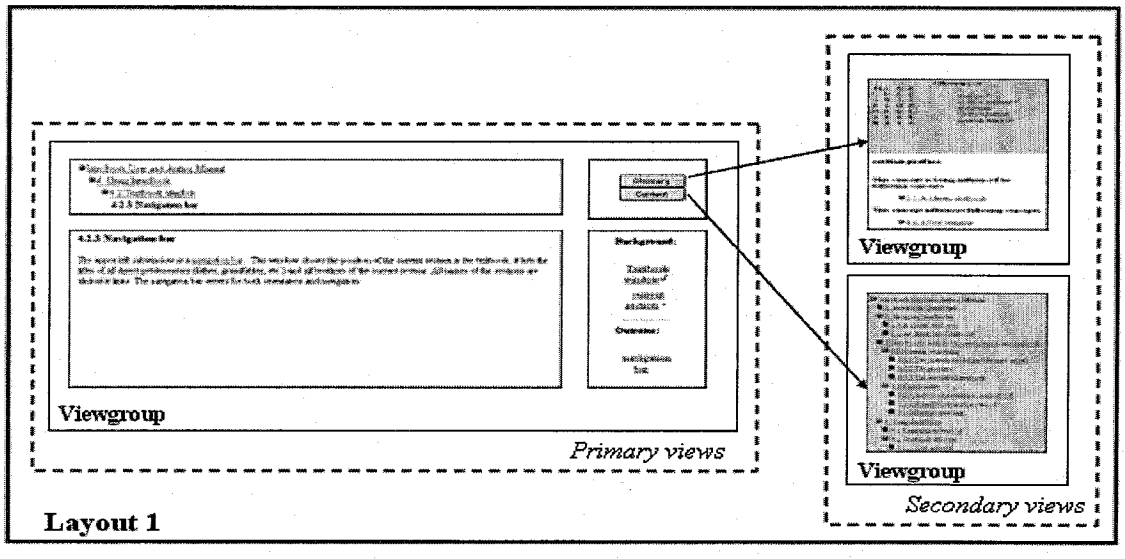


Figure 5: Interbook views of 'page_c_layout'layout

The following XML structure is an example of a layout definition for three layouts that we use to simulate an InterBook style user interface

```
<layoutlist>
      <layout name="page_c_layout" trigger="MAIN">
            <viewgroup name="MAIN" >
                  <compound framestruct="cols=80,*" >
                        <viewref name="v6" />
                        <compound framestruct="rows=50%,*" >
                              <viewref name="v3" />
                              <viewref name="v8" />
                        </compound>
                  </compound>
```

```
            </viewgroup>

            <viewgroup name="TOC" >
                  <compound framestruct="rows=*,300" border="5">
                        <viewref name="v1" />
                        <viewref name="v2" />
                  </compound>
            </viewgroup>

      </layout>

      <layout name="leaf_c_layout" trigger="MAIN">
            <viewgroup name="MAIN" >
                  <compound framestruct="cols=80,*" border="0">
                              <viewref name="v6" />
                              <viewref name="v3" />
                  </compound>
            </viewgroup>
            <viewgroup name="TOC" >
                  <compound framestruct="rows=*,300" border="5">
                        <viewref name="v1" />
                        <viewref name="v2" />
                  </compound>
            </viewgroup>
      </layout>

      <layout name="abst_c_layout" trigger="Glossary" >
            <viewgroup name="Glossary" >
                  <viewref name="v4"/>
            </viewgroup>
      </layout>


</layoutlist>
```

We have defined three layouts each associated with one of the three concept types that we use for the simulation of Interbook courses. As can be seen in the example above each layout consists of a set of viewgroups which contain pointers to predefined views. Viewgroups use *compound* elements to define the place of each of the views within the window. For each viewgroup the author of the course can also define window options for the window in which the viewgroup is placed. The layout structure of layout '*leaf_c_layout*' (without secondary views) above corresponds to the screen presented in Figure 6. It consists of five primary views grouped into one viewgroup, which is shown in the figure, and two secondary views (Glossary and Table of Content) which can be triggered by the buttons in the ToolboxView (upper right corner).

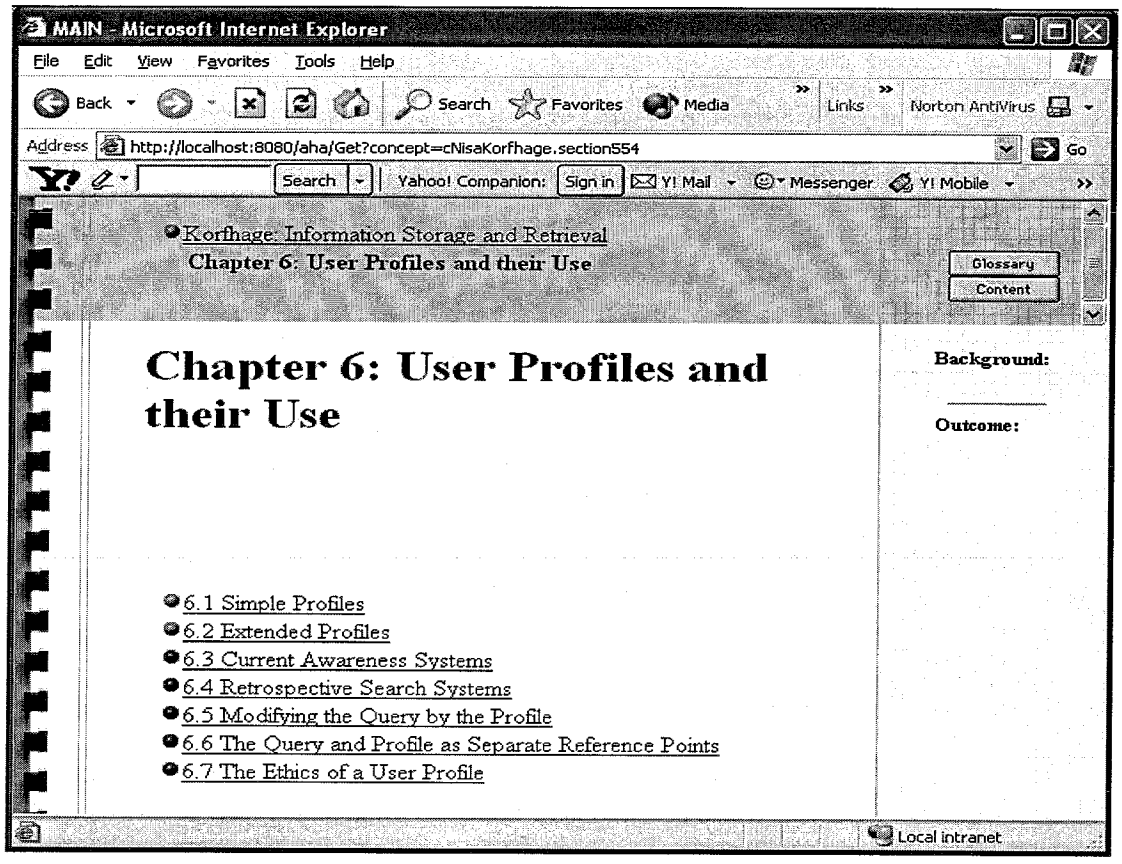


Figure 6: InterBook style concept layout for 'leaf' concepts

Changing the XML configuration structures will change the layout associated with a certain concept type. The following example of an XML configuration structure defines different layout using basically the same views:

```
<layout name="page_c_layout" trigger="MAIN">
<viewgroup name="MAIN" >
                  <compound framestruct="cols=200,*" >
                        <compound framestruct="rows=*,85" >
                              <viewref name="v1" />
                              <viewref name="v5" />
                        </compound>
                        <viewref name="v3" />

                  </compound>
      </viewgroup>
<viewgroup name="Conceptbar" >
                  <viewref name="v2"/>
      </viewgroup>
<viewgroup name="Glossary" secondary="true" >
            <viewref name="v4"/>
      </viewgroup>
</layout>
```

The corresponding screen layout for the XML configuration structure above is shown in figure 7.

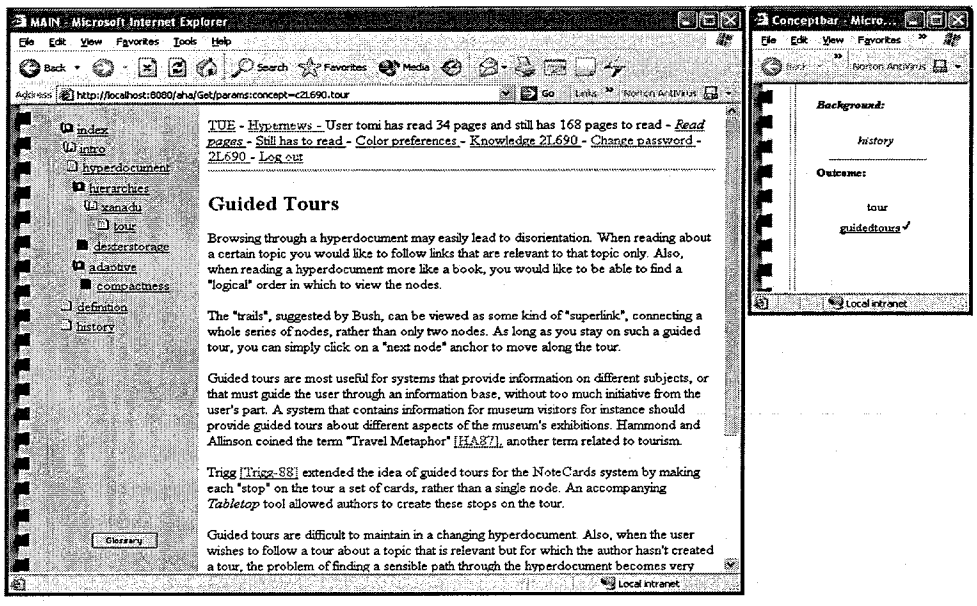


Figure 7: An example of different layout

In this layout there are two primary viewgroups (MAIN and Conceptbar) and one secondary viewgroup (Glossary). Viewgroup MAIN consists of three views (MainView, Table of Content and Toolbox) and the Conceptbar viewgroup contains one view (ConceptbarView). Button 'Glossary' in the Toolbox view triggers the display of the secondary viewgroup *Glossary*.

Concept types←→ Layout

Some adaptive hypermedia systems may have more than one *type* of concepts (pages). For example, InterBook has a *textbook page* and a *glossary concept* that are both concepts in terms of the AHA! architecture. We also assume that an author of an adaptive course may want different types of concepts to be presented differently (this is what happens in InterBook). To support this possibility, the Layout model allows an author to define several layouts. Each concept type has to be associated with one of the layouts. Presenting concepts of the same type always in the same way (using the same layout) contributes to the user confidence in the system and avoids confusion. AHA! doesn't have any predefined concept types. The author of the course can define any number of layouts and associate each of these layouts with a different layout. Figure 8 shows what happens when the user follows a link to some concept. The concept type of the desired concept is being checked any dependant on this check the associated layout is being used to present the concept to the user. If the concept type of the desired concept is the same as

the concept type of the previous concept than the layout views are being updated and no new layout is being used. Following a concept link also updates the 'active concept' field which is used by the AHA! engine to keep up the concept the user is interested in.

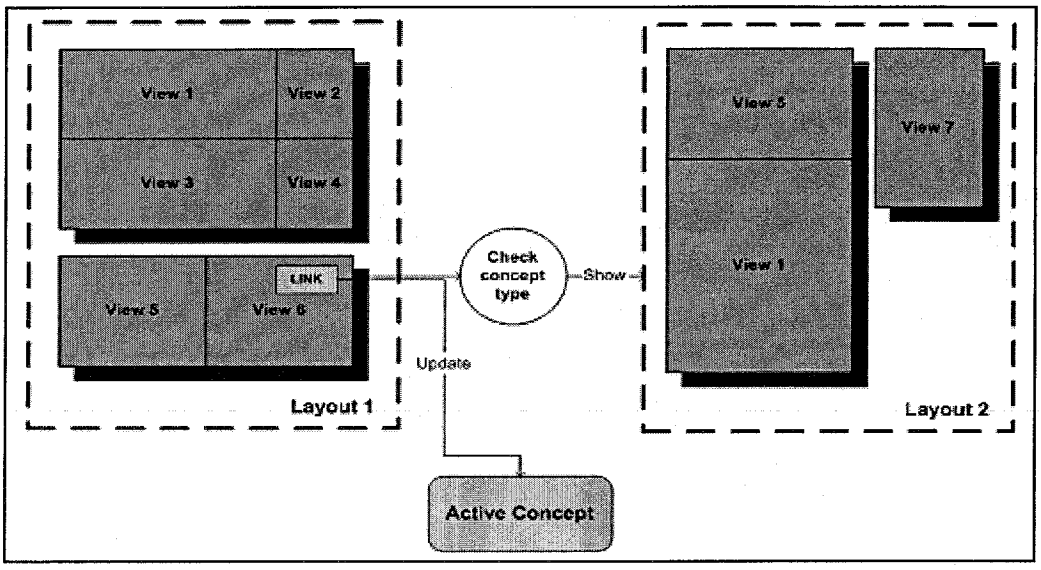


Figure 8: Following a concept link

Concept types ←→ Link annotation

As already said concepts of the same type are always represented in the same way. In order not to get the user too confused every concept type is also associated with link annotation structure. This means that concepts of the same type are always represented in the same way and also that links to concepts of this type are always annotated in the same way. This makes it easer for the user to predict what is going to happen and therefore contributes to user's confidence into the system.

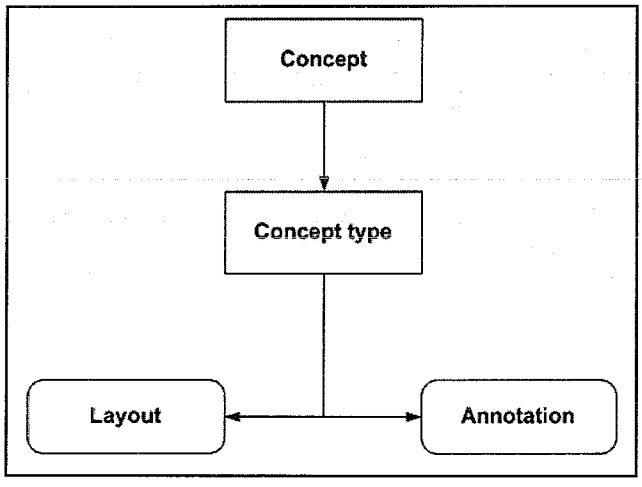


Figure 9: Concept type-Layout/Annotation

It is possible for the author of the course to define the annotation of the links for each concept type of the course. A concept link in AHA! is annotated in three parts: front icon, link text and list of icons that follow the link text.

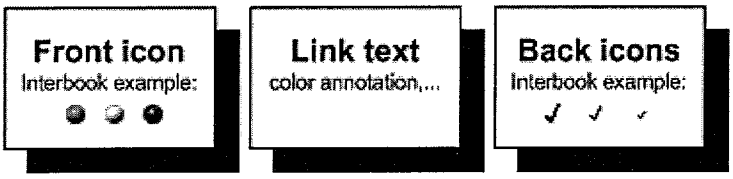


Figure 10: Link annotation structure

The Front icon mechanism uses the standard AHA! engine annotation function to determine which icon will be shown in front of the link text. The AHA! engine link annotation function results in one of four possible values: good (recommended to follow), bad (not recommended to follow), neutral (no advice) and unconditional (external link). For every resulting value of the AHA! engine link annotation function the author of the course can define an icon that will be used for the annotation. This is done by using XML structures like the following:

```
<fronticon>
      <good>GreenBall.gif</good>
      <bad>RedBall.gif</bad>
      <neutral>WhiteBall.gif</neutral>
      <unconditional></unconditional>
</fronticon>
```

The link text is also annotated by the standard AHA! engine annotation function. The color of the link text changes dependant on user's the knowledge about the concept. The author of the course can turn this feature on and off.

The third part of the link annotation is a list of icons behind the link text. This part is added to AHA! to simulate Interbook's link annotation mechanism. Interbook uses checkmarks to annotate links to domain knowledge concepts. Dependant on the user's knowledge about a certain concept a big, small, medium or no checkmark is shown behind the link to that concept. We have used this idea, adapted to AHA!'s internal concept structure, to develop a generic and configurable mechanism for this kind of link annotation. Following XML structure which can be used for configuration of 'Back icons' part of link annotation:

```
<iconanno>
      <attribute>
            <name>knowledge</name>
            <distribution>
                  <boundary>0</boundary>
                  <boundary>33</boundary>
```

```
                    <boundary>55</boundary>
                    <boundary>76</boundary>
                    <boundary>101</boundary>
                </distribution>
                <results>
                    <result>NoCheckM.gif</result>
                    <result>SmallCheckM.gif</result>
                    <result>MedCheckM.gif</result>
                    <result>BigCheckM.gif</result>
                </results>
        </attribute>
</iconanno>
```

AHA! courses can use more than one concept attribute to describe concept behavior. One can be interested in users knowledge about concepts, interests, goals, etc. Besides the *'knowledge'* attribute which is the only concept attribute used by Interbook courses, AHA! courses can use any number of concept attributes and for each of these attributes different annotation can be used. The author can associate a range of values for each concept attribute with a different icon. In the example above the following associations are being made: values between 0-33 (No checkmark), values between 33-55 (Small checkmark), values between 55-76 (Medium checkmark), values between 76-101 (Big checkmark). For each used concept attribute different ranges and icons can be used and all these icons can be used to annotate concept links and give the user more information about the concept.

Usually the author will not choose all three parts of the annotation mechanism for link annotation of concept types. For our Interbook example we use *Front icon* and *Link text* annotation for annotation of *Text page* links and for the annotation of *Item* links we use only *Back icons*.

For some views the use of some parts of annotation can be undesirable. Therefore it is possible for the views to 'overrule' the configured annotation and turn parts of annotation off. In our Interbook example the *Front icon* annotation is turned off in the text view and Link text annotation is turned off in all other views.

Layout model internally

The internal structure of the Layout model resembles the Layout structure presented at the beginning of this chapter. There is an internal list of *Layout* objects where each of these object contains a list of *Viewgroup* objects. Every *Viewgroup* object consists of a number of *View* objects that correspond to views represented to the user. Data read from XML configuration files is used to fill the in-memory representation of the Layout model. This in-memory Layout representation is used every time the user makes a request for some concept. Dependent on the requested concept, the right Layout object is used for showing the right windows and frames which represent the concept to the user.

The internal implementation of the Layout model follows the Model View Controller (MVC) architecture. MVC is a well known architecture used in a number of software packages specialized in different aspects of data presentation. The basis of MVC is the separation of data and presentation. The code is divided into data structures representing the user's data ("the model"), the objects that display the data and interact with the user ("the view"), and some logic to connect data and its representation ("the controller").

The Layout model is implemented as a separate Java package within AHA!. It can be seen as an additional layer on top of the AHA! engine which is responsible for data representation (see Figure 11). Objects in the Layout model extract data from underlying high level data structures which contain structured data gathered from the AHA! database during the initialization process. These high level data structures, which were not part of old versions of AHA!, contain easy accessible data. AHA! 2.0, the version of AHA! that was used as a basis for this project, used the XML concept database to store the concept data. Some of the views use data from different parts of the database and collecting this data on the fly is a rather expensive task. Therefore we have introduced a data layer between the database and the Layout model. This new layer consists of high level data structures that contain preprocessed data used to speed up the process of gathering data and representing it to the user. The layered architecture enables view designers to keep views simple 'one-task' objects without any complicated higher logic. A clear separation between the layers has number of advantages:

- It keeps view objects simple
- It enables a separate layer implementation
- It simplifies future improvements
- It simplifies version integration
- It speeds up the process of data gathering

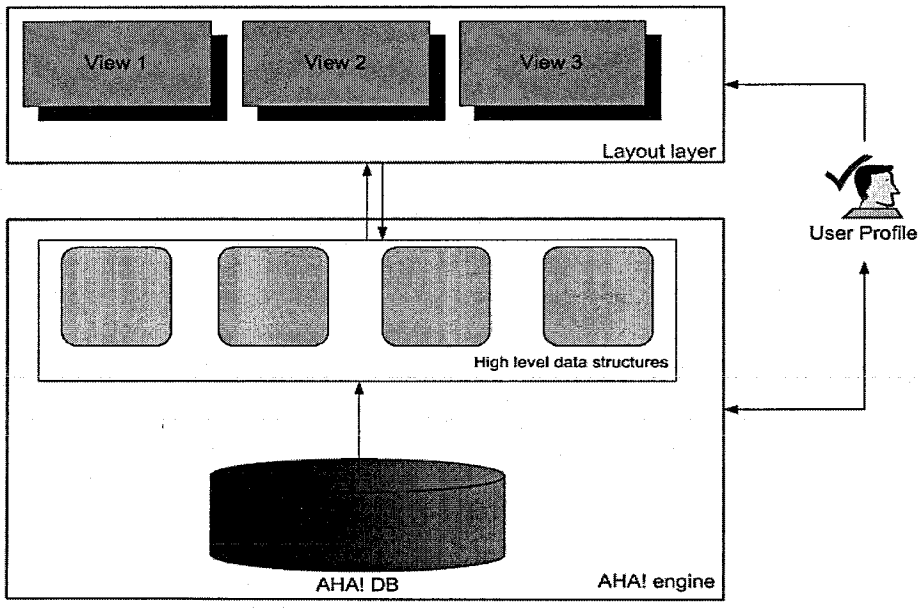


Figure 11: Layered implementation structure

A layout consists of a number of Viewgroups (windows) which are used to represent a concept to the user. However one of these windows is special because it is used as a trigger window. This window is responsible for showing of all other windows that are part of the same layout. When the user follows a link to a concept the 'Trigger window' of the concepts layout is opened. This window contains JavaScript code that triggers opening the other Layout windows (see Figure 12). The Frameset structures and JavaScript trigger code are generated from the Viewgroup and Layout objects and the actual content of the frames is produced by the view objects.

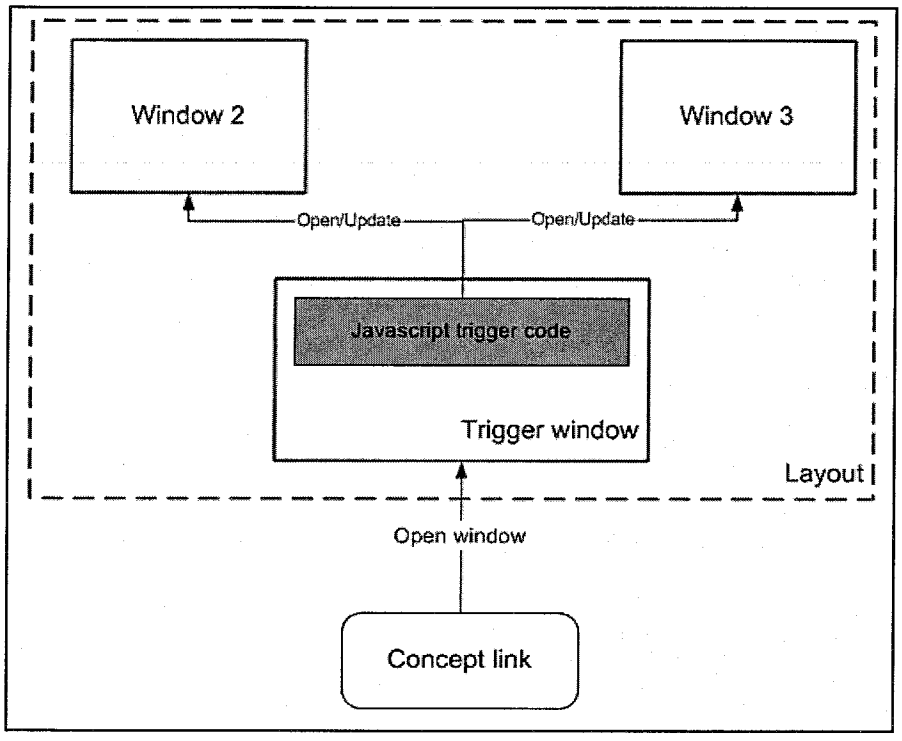


Figure 12: Layout windows construction

Two basic objects for request handling are the *ViewGet* servlet and the *Get* servlet (see Figure 13). *Get* servlet handles all the requests for window code either made by the user or by the system. Every time the user follows a concept link the *Get* servlet receives the request with a parameter *concept ID*. Dependent on the concept type the trigger window of the right layout is selected which in turn produces the response code (the frameset structure and possibly JavaScript trigger code) for the requested window. The frameset code produced by the windows of the selected layout contains calls to the *ViewGet* servlet which is responsible for filling the frames within the generated frameset structure. For every frame within the frameset structure the corresponding view is used to generate the browser code for that frame. Based on the view name the *ViewGet* servlet selects the right view from the viewlist which generates the browser code for that view.

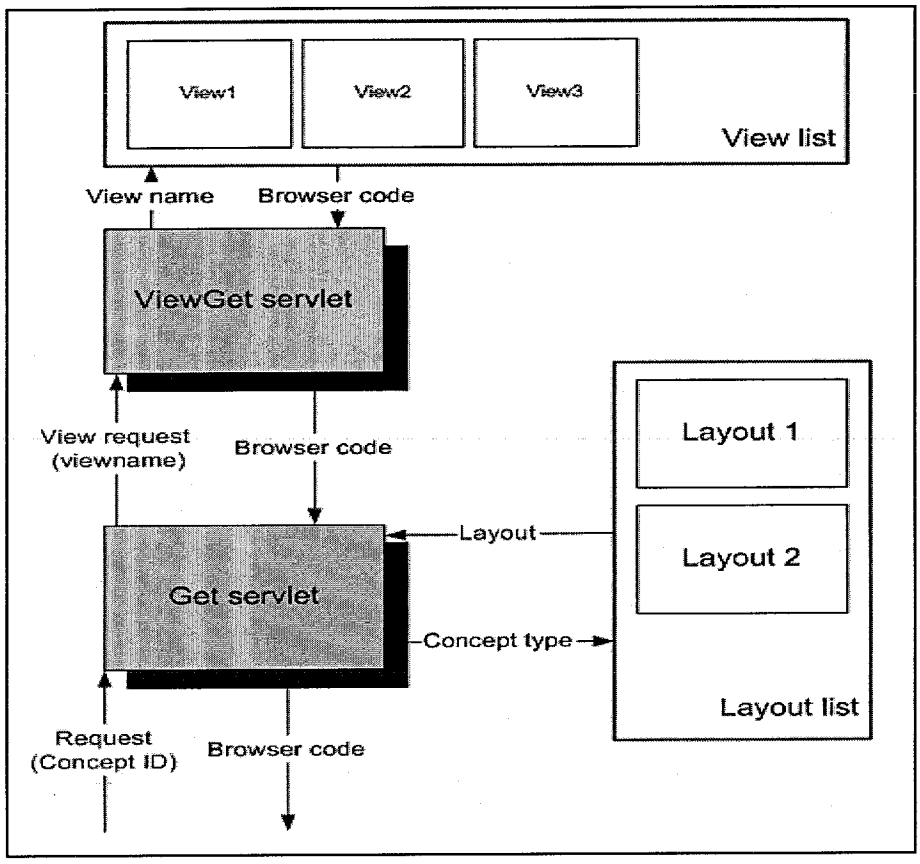


Figure 13: Layout request handling

Linking

Concept links embedded in browser code produced by the views are actually generated by *HTMLanchor* objects that are part of the AHA! Layout model. When a view 'wants' to represent a concept link it uses the *HTMLanchor* object to construct a HTML *Anchor* tag with attached JavaScript code. The *HTMLanchor* object receives a concept ID as parameter which is used to determine the trigger window of the right layout and the right annotation style. This information is further used to produce a HTML anchor element which contains all the information needed for link annotation and concept representation when the user follows the link. Separating link generation from views has number of advantages:

- It keeps views simple java objects where the programmer does not have to think about the details.

- It simplifies the process of adding new views
- It makes the linking process a mechanism instead of a hack where every view would have it's own, in many cases probably slightly different, link implementation
- It reduces mistakes
- Keeping link implementation logic centralized simplifies future improvements in the linking mechanism

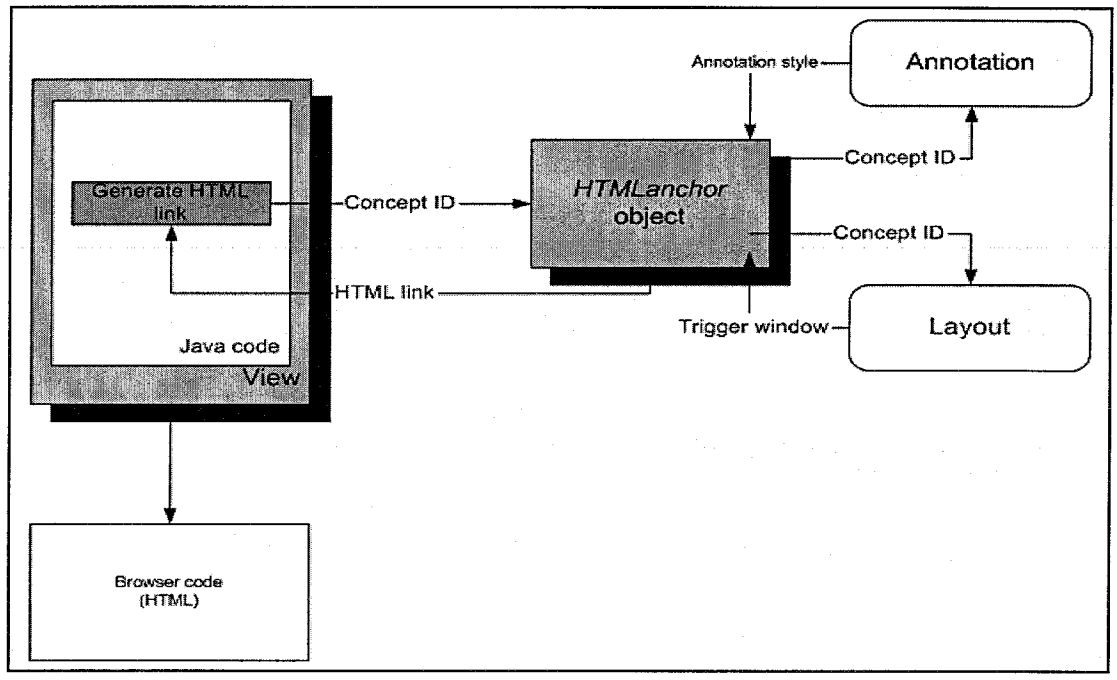


Figure 14: Link generation

The Interbook to AHA! Compiler

The final step in our attempt of bringing AHA! and Interbook together is the implementation of Interbook to AHA! compiler. There are several reasons that make this step extremely important:

- Testing the flexibility of our layout model by simulating real courses already offered by other Adaptive Hypermedia systems (in this case Interbook);
- Testing the correctness of data extracted by views. We can compare the course data served by Interbook with data served by AHA!;
- Achieving of total simulation of Interbook by AHA!. AHA! can serve real Interbook courses that also look like Interbook;
- Authoring of Interbook courses is much easer than authoring of AHA! courses. Implementation of a bridge between these two systems can simplify the AHA! authoring mechanism. The author can use the Interbook authoring mechanism (using Microsoft Word and tools to generate HTML from that) to implement a course and then use the Interbook to AHA! compiler to convert the course to AHA! format. We must note that this authoring process does not use the full power of AHA!. In particular it only uses link adaptation, not content adaptation in the form of the conditional inclusion of fragments. (But that can be added later.)

Paradigm translation

The most important part of the Interbook to AHA! compiler is the translation of the Interbook concept paradigm to the AHA! concept paradigm. These two paradigms differ in some basic aspects which make it difficult to serve Interbook courses using the AHA! engine. The Interbook paradigm consists of text pages, also called sections, and glossary concepts. Text pages are presented in a Text Window and glossary concepts are presented in a Glossary Window. AHA! on the other hand sees everything as a "concept". We introduced *concept types* to the AHA! concept paradigm. This turned to be a very simple and versatile solution for our problem. Each concept is of some type and each concept type is associated with different layout. This means that each concept type can be represented in a different way, depending on the associated layout, which is exactly what we need to simulate Interbook courses. AHA! does not have a predefined set of concept types. The author of a course can define any desirable number of concept types and represent every concept type using different layout. The connection between concept types and the layout model, which can be established using small XML configuration files, offers great flexibility and possibility to simulate the user interface of almost every existing Adaptive Hypermedia system.
To get back to the Interbook concept paradigm, our Interbook to AHA! compiler generates three kinds of concepts to simulate Interbook courses:

1. Items (simulation of Interbook concepts presented in a Glossary Window)
2. Sections (simulation of Interbook text pages presented in a Text Window having child nodes)
3. Leafs (simulation of Interbook text pages presented in a Text Window without child nodes)

All these concept types are associated with different layouts. The difference between Sections and Leafs is very small. They use layouts that are almost the same with the difference that the Sections layout shows the child nodes of the active concept and the Leafs layout does not.

Compiler Input/Output

The Interbook to AHA! compiler uses special Interbook files as input and produces AHA! formatted XML files as output. Interbook course files are valid HTML documents that contain Interbook specific codes. These codes are used to connect sections (text pages) and glossary concepts. Every section has a set of prerequisite concepts (concepts that are required to be known before reading the section) and a set of outcome concepts (concepts that are introduces by the section).
AHA! courses are saved in a different way. AHA! uses XML structures to save concept data and separate XHTML files are used for the resources. AHA! XML concept structures are much more complex than Interbook concept relations. Interbook uses two kinds of relations between Sections and Glossary concepts: 'is prerequisite' and 'is outcome'. AHA! on the other hand uses expressions to implement different kinds of relationships. These expressions can be of any form as long as they are syntactically valid. We use the following expressions to simulate Interbook concept relationships and Interbook behavior:

- For each prerequisite concept of each section:
  '**Conceptname.*knowledge*>=(1/3*100)**' for AHA requirement relationship which simulates Interbook 'is prerequisite' relationship
- For each outcome concept of each section:
  '**if (required) Conceptname.*knowledge*+1/3*(100-Conceptname.*knowledge*)**
  else Conceptname.*knowledge*+1/6*(100-Conceptname.*knowledge*) '
  for AHA Condition-Action rules which simulate Interbook 'is outcome' relationship

The following figure shows the course data transformation from Interbook format to AHA! format.

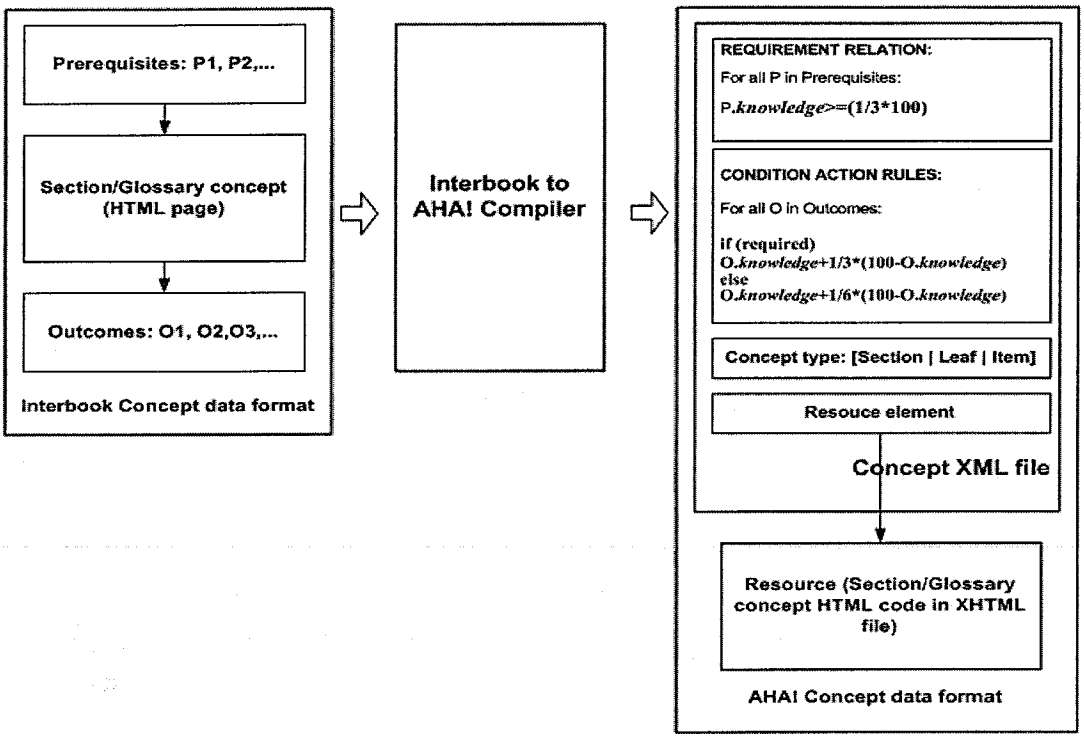


Figure 15: Interbook to AHA! concept data

At this moment we have used 'Interbook to AHA!' to compile two Interbook courses to AHA! format. Figure 16. shows a screen from Interbook's *'Korfhage: Information Storage and Retrieval'* course served by AHA!.

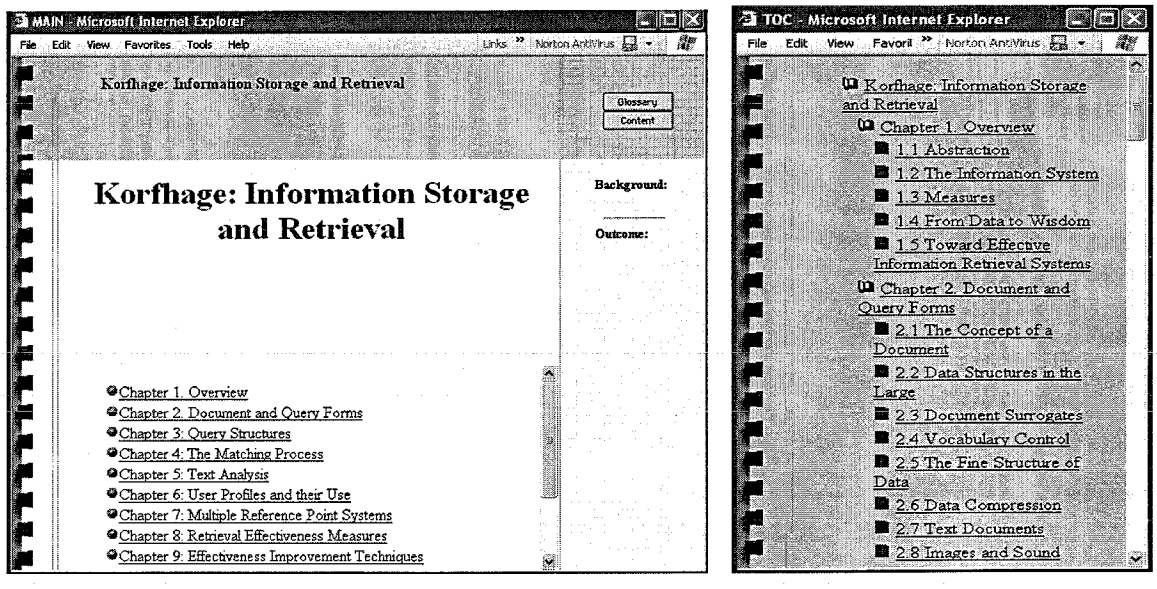


Figure 16: NisaKorfhage course screen example

Conclusions

The results of this project exceed by far our original expectations and goals. After the slow and problematic beginning where some importation decisions had to be made about the further continuation of the project, we have found a way to improve AHA! in terms of methodology instead of using narrow minded solutions. We have introduced Interbook features to AHA! engine, but we did not stop there. We went one step further and gave AHA! a generic user interface mechanism.

The final results of the project are the following:

- On the conceptual level we have introduced new elements: hierarchy, concept types and titles. This extension was required in order to simulate Interbook's user interface paradigm at first, but finally concept types served as an important part of our Layout model.

- We have added different Interbook features to AHA!: Navigation bar, Concept bar, Glossary, Child-nodes bar, Toolbox and Table of Contents.

- We have developed a flexible and highly configurable layout model that can simulate user interface of almost any existing Web based adaptive hypermedia system. The new AHA! layout model offers versatile user interface possibilities and brings the AHA! one step closer to its main goal of being a generic AH environment for all kinds of AH applications. View based concept presentation is extremely flexible and gives a course author the power to adapt the user interface to the needs of the course.

- We have developed number mechanisms to support the layout model: link annotation mechanism, concepttype-layout mechanism and linking mechanism.

- The last step of our project was developing a compiler for translating Interbook courses to AHA! format. This final step enabled us to serve Interbook courses in Interbook style, using Interbook user interface, with AHA! engine.

- We published two papers about the project results. The first paper was presented during the AH2003 workshop of the 12th WWW conference in Budapest, Hungary and the second one will be presented during the ELearn conference in Phoenix, Arizona.

Recommendations

More standard views

The course user interface is dependent on the set of already existing views. Greater variety of views means more possibilities for the author to construct a good user interface. The views that we have developed so far are copies of Interbook's views. One can think however of more useful views that can be included into this set of standard views. One of the views used by many (adaptive) hypermedia systems is *History view.* This view shows the history of the users movements in hyperspace. It would be very useful to develop a view that shows a list of links to the last 10 or 15 (or any number the author wishes) concepts the user has visited.

Graphic tools

Development of graphic tools could make the authoring of course user interface much easier and much more efficient. In current situation the author specifies the course user interface by implementing user interface configuration files which can be done using any existing XML editor and any existing text editing tool. This gives the author a lot of power and flexibility to build up the course user interface however during the configuration process the author is 'blind' and in order to get the desired effect the author had to do a lot of switching between the XML editor and the AHA! application has to be done. Starting the AHA! application is the only way to see what the layout specified in the configuration files actually looks like. It is obvious that this layout authoring process can be greatly improved by giving the author 'eyes' during the layout configuration process. Graphic tools are the solution. One can think of introducing a graphic tool which would enable the author to graphically build up the layout and the tool would generate the XML configuration file needed by the AHA! engine.

Adaptive user interface

An improvement would be making the user interface adaptive. Dependant on the user model different layout could be used to represent the same concept type. This sounds in contrast with the basic idea of the *Layout model* introduced by this report where the concepts of the same concept type are always represented in the same way by using the same layout. However defining more layouts per concept type and making this dependant on the user model could make the *Layout model* an adaptive user interface model.
For example one could make concept representation dependant on the device the user is using. Somewhere in the user model name/value pair 'device' could be stored. For smaller devices simple layouts could be defined containing only really important views.

For bigger screens more complex layouts could be defined with more views which also need more space. So the concept representation could be adapted to the size of users screen.
Another example of usefulness of defining more layouts per concept type would be the implementation of some kind of 'incremental interface'. Dependant on the user's level of knowledge or some other criteria which would define user's familiarity with adaptive hypermedia systems and their user interfaces simple or more complex concept representations could be used to represent concepts. One can think of using simple concept representations in the beginning when the user doesn't have much knowledge. After some time when the user gets more familiar with the user interface more complex concept representations could be used.
We must stress that using more layouts per concept type can also have certain disadvantages. The user can loose self confidence and trust into the system if the user interface is too unpredictable. Therefore differences between the layouts of the same concept type should not be too big (this doesn't count for the example of adaptation of user interface to user's device). System must give user the feeling that he/she has some degree of control over the system.

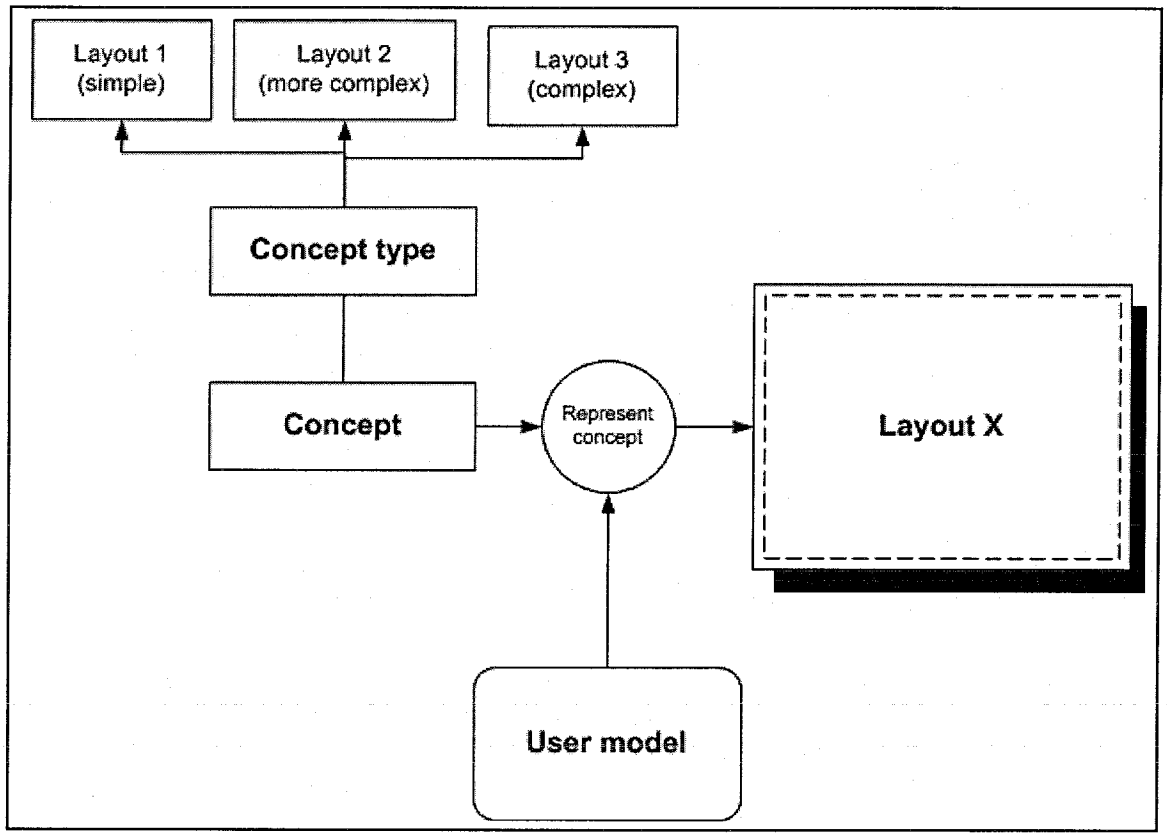


Figure 17: Adaptive user interface

Link annotation mechanism improvements

The motto of this project was 'mechanism instead of hack'. We have developed a mechanism for the user interface construction which enables the author of the course to build up the course user interface using XML configuration files. We have developed a linking mechanism for concept links generation which simplifies view development. And we made our first attempt to develop a link annotation mechanism. The author can use a XML configuration file to specify a link annotation style for each concept type used by a course. This mechanism is a combination of the AHA!'s link annotation and Interbook's link annotation mechanism. However it is not quite ready yet to serve as a generic link annotation mechanism. It needs to give more power to the author to adapt link annotation of different concept types to his/hers wishes and course characteristics.

More power for the author

Another very important improvement would be extending the user interface adaptation possibilities by introducing the total data-presentation separation. One can think of giving the author the opportunity of implementing his/hers own views, in addition to using a set of predefined views. If the internal static AHA! data structures would be saved as XML files the author could use any standard XSLT editor to implement views as XSLT files which could extract data from XML formatted data structures. This model would give the author the possibility to represent the data in any desirable way without being dependent on already implemented views.

Adding content adaptation to Interbook's authoring mechanism

The development of the '*Interbook to AHA!*' compiler has introduced a new direction in AHA!'s authoring process. It is possible to develop a course using Interbook's authoring process and then use the 'Interbook to AHA!' compiler to convert the course to the AHA! format. The advantage of this two-step process is that Interbook's authoring mechanism is much easier and faster than AHA!'s authoring mechanism. However this new authoring process does not use the full power of AHA!. Interbook's authoring process has been developed for Interbook courses which only make use of link annotation technique. AHA! courses however use besides link adaptation also content adaptation in the form of the conditional inclusion of fragments. In order use the full power of AHA! content adaptation should be added to the new authoring mechanism.

Existing views improvement

The views we have implemented so far are copies of Interbook's views and therefore totally adapted to Interbook's internal structure. These views reflect Interbook's internal concept relations. The internal structure of AHA! differs however from Interbook in many ways. AHA! supports different kinds of concept relations. The Concept bar view and the Glossary view are used for showing concept relations in Interbook style. These views could be improved to reflect different kinds of AHA! concept relations.

References

Brusilovsky, P. (1996) Methods and techniques of adaptive hypermedia. In P. Brusilovsky and J. Vassileva (eds.), *User Modeling and User-Adapted Interaction* **6** (2-3), Special Issue on Adaptive Hypertext and Hypermedia, 87-129.

Brusilovsky, P. (2001) Adaptive hypermedia. *User Modeling and User Adapted Interaction* **11** (1/2), 87-110, also available at http://www.wkap.nl/oasis.htm/270983.

Brusilovsky, P., Eklund, J., and Schwarz, E. (1998) Web-based education for all: A tool for developing adaptive courseware. *Computer Networks and ISDN Systems* (Proceedings of Seventh International World Wide Web Conference, 14-18 April 1998) **30** (1-7), 291-300.

De Bra, P. (1998) *Definition of hypertext and hyupermedia*, http://wwwis.win.tue.nl/ah/

De Bra, P., Aerts, A., Smits, D., and Stash, N. (2002) AHA! Version 2.0: More Adaptation Flexibility for Authors. In: M. Driscoll and T. C. Reeves (eds.) Proceedings of World Conference on E-Learning, E-Learn 2002, Montreal, Canada, October 15-19, 2002, AACE, pp. 240-246.

De Bra, P., Brusilovsky, P., and Houben, G.-J. (1999a) Adaptive Hypermedia: From Systems to Framework. *ACM Computing Surveys* **31** (4es), also available at http://www.cs.brown.edu/memex/ACM_HypertextTestbed/papers/25.html.

De Bra, P. and Calvi, L. (1998) AHA! An open Adaptive Hypermedia Architecture. In P. Brusilovsky and M. Milosavljevic (eds.), *The New Review of Hypermedia and Multimedia* **4**, Special Issue on Adaptivity and user modeling in hypermedia systems, 115-139.

De Bra, P., Houben, G. J., and Wu, H. (1999b) AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. In: Proceedings of 10th ACM Conference on Hypertext and hypermedia (Hypertext'99), Darmstadt, Germany, February 21 - 25, 1999, ACM Press, pp. 147-156.

Grigoriadou, M., Papanikolaou, K., Kornilakis, H., and Magoulas, G. (2001) INSPIRE: An Intelligent System for Personalized Instruction in a Remote Environment. In: P. D. Bra, P. Brusilovsky and A. Kobsa (eds.) Proceedings of Third workshop on Adaptive Hypertext and Hypermedia, Sonthofen, Germany, July 14, 2001, Technical University Eindhoven, pp. 13-24.

Henze, N. and Nejdl, W. (2001) Adaptation in open corpus hypermedia. In P. Brusilovsky and C. Peylo (eds.), *International Journal of Artificial Intelligence in Education* **12** (4), Special Issue on Special Issue on Adaptive and Intelligent Web-based Educational Systems, 325-350, also available at http://cbl.leeds.ac.uk/ijaied/abstracts/Vol_12/henze.html.

Melis, E., Andrès, E., Büdenbender, J., Frishauf, A., Goguadse, G., Libbrecht, P., Pollet, M., and Ullrich, C. (2001) ActiveMath: A web-based learning environment. In P. Brusilovsky and C. Peylo (eds.), *International Journal of Artificial Intelligence in Education* **12** (4), Special Issue on Special Issue on Adaptive and Intelligent Web-based Educational Systems, 385-407.

Shneiderman, B., Kearsley G., *Hypertext Hands-On!: An Introduction to a New Way of Organizing and Accessing Information*, Addison Wesley, 1989.

Weber, G. and Brusilovsky, P. (2001) ELM-ART: An adaptive versatile system for Web-based instruction. In P. Brusilovsky and C. Peylo (eds.), *International Journal of Artificial Intelligence in Education* **12** (4), Special Issue on Adaptive and Intelligent Web-based Educational Systems, 351-384, also available at http://cbl.leeds.ac.uk/ijaied/abstracts/Vol_12/weber.html.

Wu, H., De Kort, E., and De Bra, P. (2001) Design Issues for General Purpose Adaptive Hypermedia Systems. In: Proceedings of Twelfth ACM Conference on Hypertext and Hypermedia (Hypertext 2001), Aarhus, Denmark, August 14-18, 2001, ACM Press, pp. 141-150.

Table of Figures

Abbreviations

AHA	Adaptive Hypermedia Architecture
AHA!	Adaptive Hypermedia for All!
AHS	Adaptive Hypermedia System
CL-HTTP	Common LISP Hypermedia Server
EB	Electronic Textbook
HS	Hypermedia System
HTML	HyperText Markup Language
MVC	Model View Controller
UI	User Interface
UM	User Model
WWW	World Wide Web
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language

Appendixes

Adaptive Textbooks on the World Wide Web
(paper about Interbook)

AHA! The Next Generation
(paper about AHA!)

Adaptive Textbooks on the World Wide Web

John Eklund, Faculty of Education, The University of Technology, Sydney, PO Box 222 Lindfield NSW 2070, Australia. Phone +61 2 95145613 Fax: +61 2 9514 5666 j.eklund@uts.edu.au

Peter Brusilovsky, Human-Computer Interaction Institute, School of Computer Science, Carnegie Mellon University Pittsburgh, PA 15213, USA plb@cs.cmu.edu

Elmar Schwarz, Department of Psychology, Carnegie Mellon University Pittsburgh, PA 15213, USA eschwarz@andrew.cmu.edu

Keywords

adaptive, authoring, hypermedia, navigation, textbook, user-model, WWW

Abstract

This paper examines some recent research in the area of adaptive navigation support, a class of adaptation in user-model based interfaces, and specifically discusses the authoring and delivery tool for adaptive electronic textbooks (ETs) called Interbook [HREF2], implemented on the WWW. Interbook uses history-based, knowledge-based and prerequisite-based adaptive annotation of links to suggest to the individual user an appropriate path through a learning space. We describe the authoring environment and the principles upon which it is based, and discuss our efforts to experiment with, and to evaluate, this tool.

Introduction

In the domain of educational hypermedia on the WWW, instructional material may be linked in an organised fashion, the aim being to sequence the learning materials to achieve a specific educational objective. The author has sequenced the curriculum to provide an optimal learning path for the average learner (Weber & Specht, 1997). In this case the knowledge implicit in the hypermedia is well defined and carefully structured, similar to that found in a textbook.

For a moment, consider a textbook such as one of the many "visual quickstart" guides that may help a novice or intermediate user of software such as *Clarisworks*, or *Word for Windows*. These textbooks are typically structured through the use of chapters, sections,

headings and subheadings, and the material is sequenced from "begin here" to "more advanced". It has an implied set of prerequisites so that it is intended to be read, overall, from beginning to end. There are also navigational devices in this textbook: A Table of contents at the start, an index at the back, and throughout the book there are pointers to other places in the text that may be relevant to what is being discussed. This textbook may also have a "test yourself" section at each chapter, so that readers may find out how much of the material they have understood before proceeding to the next chapter. In all these features of the textbook, the expert has placed their understanding of the content, and their understanding of how novices might best learn from it, into the book, and thus onto the knowledge contained within it. The pages provide a sequence to the knowledge in the book, but there is also the opportunity to move around it. Similarly if this book were put in the form of a hypertext document, it could read in a linear manner, or traversed very easily using the hyperlinks.

Two problems with this common form of textbook present themselves: The textbook, whether in print or on the WWW is the same for all users - it has no understanding of who is using it, and cannot change it's behaviour in terms of what it presents on that basis. This is a problem because readers have different knowledge, goals, computer experience and will learn the material at different rates. Secondly, providing the readers (or users) with navigation aides assumes that they can make sensible decisions about when to use them, and about where to proceed in the body of knowledge. However, there is some evidence (eg Hammond, 1989) which tends to suggest that they do not always make well informed and careful decisions. Once again some readers will be able to handle certain navigation features - others not: So perhaps the book should be structured and indexed differently for different classes of readers/learners/users?

We contend that the often cited problem of "becoming lost in hyperspace" is really one of loosing sight of objectives, of being unable to make sense of the new material, and in terms of learning, there is no distinction between this problem in paper-based text or in hypertext, or from a book and a hypertext document.

Given that the hypertext version of the textbook will allow greater possibilities to implement individualised support, the question posed is: How can we design a Web-delivered learning environment in the form of a textbook which goes some way toward addressing these problems?

Adaptive Hypermedia Systems and adaptive navigation support

Adaptive hypermedia systems are capable of altering the content or appearance of the hypermedia on the basis of a dynamic understanding of the individual user. Information about a particular user can be represented in a *user model* to alter the information presented. We define these systems as"...all hypertext and hypermedia systems which reflect some features of the user in the user model and apply this model to adapt various visible and functional aspects of the system to the user." [HREF7]. More specifically, Adaptive Navigation Support (ANS) is a generic name for a group of techniques used in adaptive hypermedia systems (Brusilovsky, 1996) which use this model to provide

directional assistance to the user. For example, suggesting where the user should proceed, or annotating what is learned and what is ready to be learned.

This paper is specifically concerned with adaptive navigation support on the Web, and we describe some ANS methods for the WWW (see also Eklund & Zeiliger, 1996). Using a Web-based adaptive educational system which uses ANS, namely InterBook ([HREF2], Brusilovsky, Schwarz, and Weber, 1996), we offer one possible solution to the problems of the passive nature of the media (its inability to alter itself to meet the needs of the individual user), without limiting the free-browsing, learner-controlled nature of hypermedia and more generally that of the WWW.

Adaptive navigation support techniques can be classified in several groups according to the way they adapt presentation of links (Brusilovsky, 1996): direct guidance, sorting, hiding, and annotation. These will be briefly described.

Direct guidance can be applied in any system which can decide what is the next "best" node for the user to visit according user's goal and other parameters represented in the user model. ELM-ART (Schwarz, Brusilovsky, and Weber, 1996) is an example of an adaptive system implemented on the WWW that uses this technique. ELM-ART [HREF6] generates an additional dynamic link (called "next") connected to the next most relevant node to visit. Direct guidance has been criticised for being "too directive" as it provides almost no support for users who would like make their own choice rather than follow the system's suggestion.

In *adaptive ordering* technology all the links of a particular page are sorted according to the user-model using some easily recognisable means of conveying this to the user, such as having the more relevant links closer to the top (Hohl, Bocker, and Gunzenhauser, 1996). This technology exists within ELM-ART and Interbook. Adaptive ordering has a limited applicability: it can be used with non-contextual links, but it cannot be used for indexes and content pages (which usually have a stable order of links), and can never be used with contextual links and maps.

Hiding is an annotation technology which restricts the navigation space by hiding links to irrelevant pages. A page can be considered as irrelevant for several reasons: for example, if it is not related to the user's current goal (Brusilovsky, and Pesin, 1994; Vassileva, 1996) or if it presents materials which the user is not yet prepared to understand (Brusilovsky and Pesin, 1994; Perez, Gutierrez, and Lopisteguy, 1995).

Adaptive annotation technology augments the links with a comment which informs the user about the current state of the nodes behind the annotated links (Brusilovsky, Pesin, and Zyryanov, 1993; de La Passardiere, and Dufresne, 1992; Hohl, Becker, and Gunzenhauser, 1996; Schwarz, Brusilovsky, and Weber, 1996). Link annotations can be provided in textual form or in the form of visual cues, for example, using different icons, or colours, font sizes, or font types. Typically the annotation in traditional hypermedia is static, that is independent of the individual user. Adaptive navigation support can be

provided by dynamic user model-driven annotation. Adaptive annotation in its simplest history-based form (outlining the links to previously visited nodes) has been applied in some hypermedia systems (for example, TopClass [HREF8], which shows a folder as unread with a "U" until all of the items within that folder have been visited), including several World-Wide Web browsers. Even the form adaptive annotation which distinguishes two states of links is quite useful.

Adaptive annotation for WWW

History-based adaptive annotation is familiar to WWW users because any WWW browser allows them to distinguish visited and unvisited nodes, showing these nodes in different colours. We offer more advanced methods of adaptive annotation which could be also very helpful for WWW users. All adaptive navigation support methods are based on three main decisions about representing the knowledge about the domain, the course, and the student.

The knowledge about the domain is represented in the form of a concept-based domain model. The simplest form of domain model is just a set of domain concepts. These concepts can be named differently in different systems - topics, attributes, knowledge elements, objects, learning outcomes, but in all the cases they are just elementary pieces of knowledge for the given domain. Depending on the domain and the application area, the concepts can represent larger or smaller pieces of domain knowledge. A more advanced form of the domain model is a network with nodes corresponding to domain concepts (and with links reflecting several kinds of relationships between concepts). This network represents the structure of the domain in a hypermedia system.

The domain model provides a structure for the representation of the user's knowledge of the subject. For each domain model concept, an individual user knowledge model stores some value which is an estimation of the user knowledge level of this concept. This can be a binary value (known Ð not known), a qualitative value (good-average-poor), or a quantitative value, such as the probability that the user knows the concept. The individual user-knowledge model, which is called an overlay model, is most often used in adaptive hypermedia systems. An overlay model of user knowledge can be represented as a set of pairs "concept - value", one pair for each domain concept. The overlay model is powerful and flexible, it can measure independently the user's knowledge of different topics.

The knowledge about the course is represented by indexing hypermedia nodes containing various units of learning material (presentations, tests, examples, problems) with domain model concepts which are related to the content of the unit. This is a relatively popular direction for the development of education-oriented hypermedia systems. There are two major types of indexing: content-based indexing and prerequisite-based indexing. With content-based indexing, a concept is included in a page index if some part of this page presents the piece of knowledge designated by the concept (Brusilovsky and Pesin, 1994; Schwarz, Brusilovsky, and Weber, 1996; Zeiliger, 1993). With prerequisite-based indexing, a concept is included in a page index if a student has to know this concept to understand the content of the page (Schwarz, Brusilovsky, and Weber, 1996).

We identify two further methods for ANS on WWW: knowledge-based annotation and prerequisite-based annotation. The idea of the "knowledge-based" method is to distinguish different levels of the user's knowledge of the node. We suggest the use of three graduations: not-known, in-work (partially known) and well-learned, and annotate differently the links to the nodes of these three classes as in (Brusilovsky, Pesin, and Zyryanov, 1993; de La Passardiere and Dufresne, 1992). Here by "not-known" we mean that the user has never heard about some of the concepts from the node's outcome. "In-work" means that the user has acquired some information about all the concepts presented in this node (it does not necessarily imply that the user has just visited this node!). "Well-known" means that the user confirmed his or her knowledge of all the concepts presented in this node by answering tests or solving problems. This method requires a user model which can distinguish three levels of user knowledge of the concept: the user has never heard about a concept, the users has read some information about a concept, and finally that the user has correctly answered a test or solved a problem which requires this concept. It also requires the embedding of tests into the courseware, which has not yet been implemented in the version of Interbook described in this paper.

In the "prerequisite-based" method we distinguish nodes which are ready and not-ready to be learned as in (Brusilovsky and Pesin, 1994). The node is considered as not-ready-to-be-learned in two cases: first if any of the concepts in the prerequisite section of node index is not-known and second if any concept from the outcome section of node index has a not-known prerequisite concept. This method could be implemented with the simplest form of overlay model which only distinguishes known from not-known. Pre-requisite based adaptive annotation is a feature of Interbook.

Adaptive Navigation Support in InterBook

InterBook (Brusilovsky, Schwarz, and Weber, 1996) is a system for authoring and delivering adaptive electronic textbooks on WWW. All InterBook-served electronic textbooks have generated table of content, a glossary, and a search interface. In InterBook, the structure of the glossary resembles the pedagogic structure of the domain knowledge. Each node of the domain network is represented by a glossary entry. Likewise each glossary entry corresponds to one of the domain concepts.

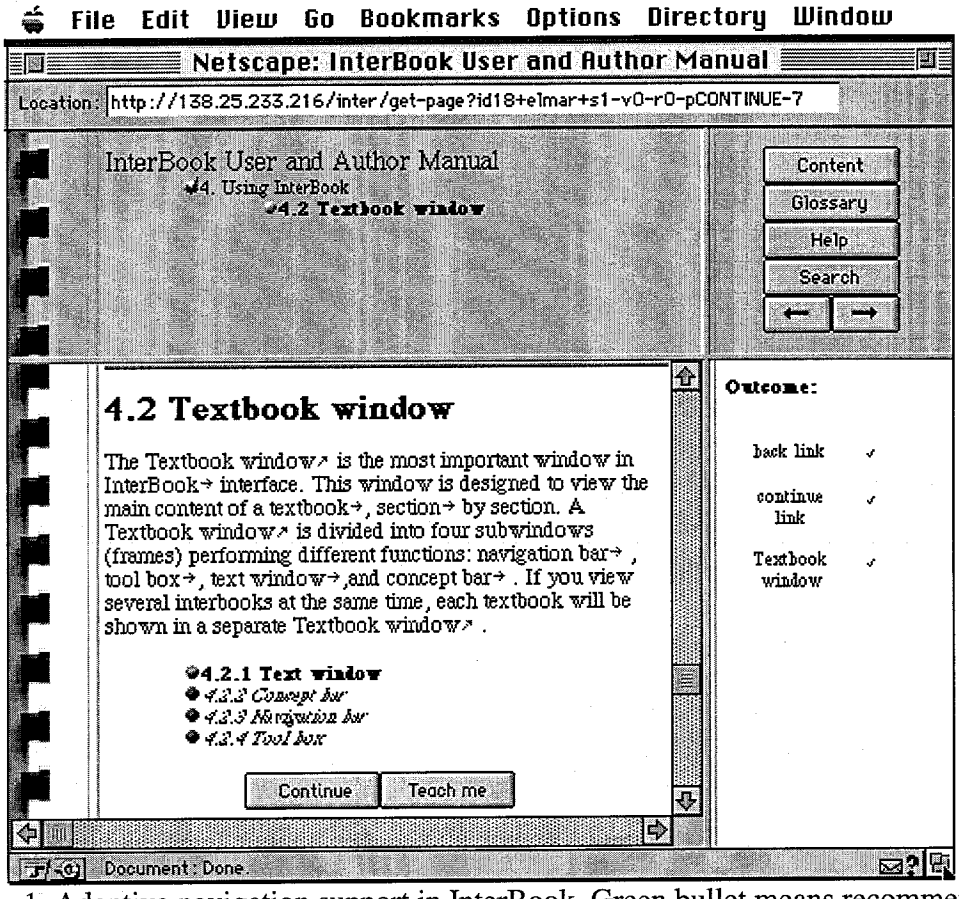


Figure 1. Adaptive navigation support in InterBook. Green bullet means recommended, red bullet means "not ready to be learned", white bullet means "nothing new".

All sections of an electronic textbook are indexed with domain model concepts. For each section, a list of concepts related with this section is provided (we call this list the spectrum of the section). The spectrum of the section can represent also the role of a concept in the section (each concept can be either an outcome concept or a background concept).

The knowledge about the domain and about the textbook content is used by InterBook to serve a well-structured hyperspace. In particular, InterBook generates links between the glossary and the textbook. Links are provided from each textbook section to corresponding glossary entries for each involved background or outcome concept. Similarly from each glossary entry describing a concept InterBook provides links to all textbook units that can be used to learn this concept. It means that an InterBook glossary integrates features of an index and a glossary. These links are not stored in an external format but generated on the fly by a special module that takes into account the student's current state of knowledge represented by the user model.

InterBook uses coloured bullets and different fonts to provide adaptive navigation support (Figure 1). Wherever a link appears on InterBook pages: in the table of content,

in the glossary or on a regular page, its font and colour of its bullet will inform the user about the status of the node behind the link. InterBook integrates all three methods of annotation: history-based, knowledge-based and prerequisite-based. Currently four colours and three fonts are used. Green bullet and bold font means 'ready and recommended', ie., the node is ready-to-be-learned but still not learned and contains some new material. A red bullet and an italic font warns about a not-ready-to-be-learned node, while white means 'clear, nothing new', ie., all concepts presented on a node are known to the user. Violet is used to mark nodes which have not been annotated by an author. A check mark is added for already visited nodes. Currently, InterBook does not support tests and can not provide "well-learned" annotation. This is currently under development.

The user model in Interbook is initialised from the registration page via a stereotype model, and is modified as the user moves through the information space. New work on Interbook includes the provision of an "interview" to further specify the user model, and embedded testing for knowledge-based navigation support. The user model for each user is stored in a file on the server in a Lisp format.

Authoring

Authoring an adaptive electronic textbook can be divided into 5 steps which are described in detail below (see Figure 2). In brief, an Electronic Textbook is prepared as a specially structured *Word* file and the task is to convert this file into InterBook format. The result of this process is a file with the Textbook in InterBook format which can be served on WWW by the InterBook system.

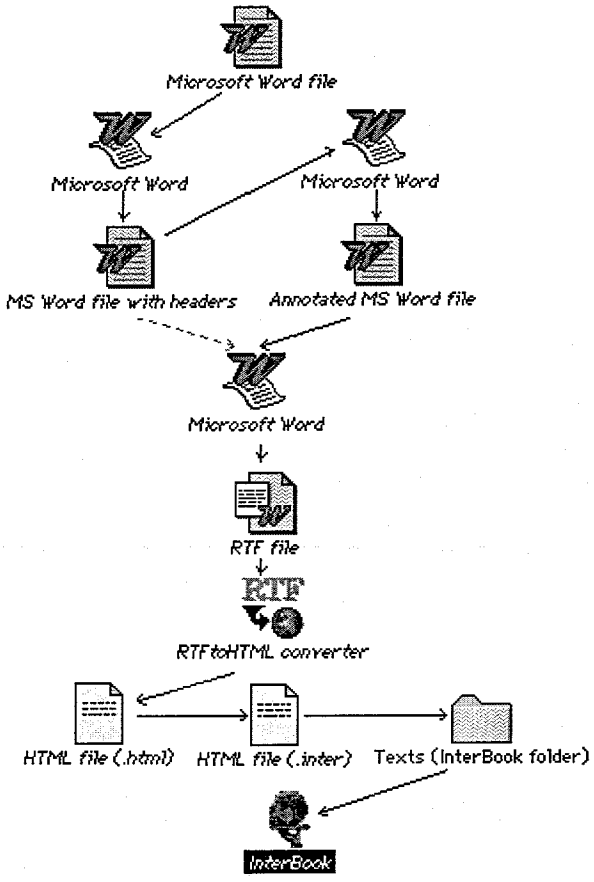


Figure 2. Adaptive Electronic Textbook on the WWW in 5 steps.

InterBook recognizes the structure of the document through the use of headers. It means that the titles of the highest level sections should have a pre-defined text style "Header 1", the titles of its subsections should have a pre-defined paragraph style "Header 2", and so forth. The title of the textbook should have paragraph style "Title". The result of this step will be a properly structured MS Word file.

The second step in the authoring process then involves concept-based annotation of the Electronic Textbook (ET) to let InterBook know which concepts stand behind each section. This knowledge allows InterBook to help the reader of the ET in several ways, and the result of this step is an annotated (and structured) MS Word file.

An annotation is a piece of text of special style and format inserted at the beginning of each section (between the section header and the first paragraph). Annotations have special character style (hidden + shadowed) which are not visible in the text window to the reader of the ET. For each unit the author provides a set of outcome and background concepts. In this way, each section is annotated with a set of prerequisite concepts (or terms which exist in other sections which should be read before the current section), and a set of outcome concepts (terms which will be assumed known once the reader has

visited the section). The format for the outcome annotation is: (out: concept-name1, concept-name2, etc.) and the format for the background annotation is: (pre: concept-name1, concept-name2, etc.).

Once the annotations are complete the file is saved in RTF format. The RTFtoHTML program [HREF5] with some special settings is used to convert the ET into HTML format. Then the .html extension on the file is manually altered to .inter so that it can be recognised by the Interbook system.

Lastly, when the InterBook server starts, it parses all interbook files in its "Texts" folder (i.e. all files with extension .inter) and translates it into the list of section frames. Each unit frame contains the name and type of the unit, its spectrum, and its position in the original HTML file. The obtained LISP structure is used by InterBook to serve all the available textbooks on WWW providing the advanced navigation and adaptation features. The content which is presented to the user is generated on-the-fly using the knowledge about the textbook, the user model, and HTML fragments extracted from the original HTML file. These features of InterBook are based on the functionality of the Common Lisp Hypermedia Server.

Evaluating Interbook

The results of some evaluations studies of adaptive hypermedia systems have already been reported. Brusilovsky and Pesin (1995, [HREF4]) conducted an experiment with the ISIS adaptive tutor with twenty-six subjects and used the overall number of navigation steps, the number of repetitions of previously studied concepts, the number of transitions from concept to concept and transitions from index to concept. They found that the number of movements were significantly less for students with the adaptive tutor, and concluded that adaptive annotation made learners more purposeful, completing the work with less navigation steps.

While Interbook undergoes development at Carnegie Mellon University, we are currently experimenting with the Interbook tool at the University of Technology, Sydney [HREF1]. An Interbook server has been established and we have been granted permission by Peachpit Press to use sections of Charles Rubin's book "MacBible Guide to Clarisworks4" (Rubin, 1996) as content for an adaptive ET. The database and spreadsheet sections of the textbook are being set up as Interbooks on the server through the authoring process described above. Second year students in an educational computing elective, which has a focus on the use of Clarisworks Database and Spreadsheet modules for information handling, will be using the ETs: One group *with* and a group *without* the adaptive annotation. We are attempting to measure both comprehension (via a multi-choice test administered after the sessions) and their navigation traits (via the use of audit trails).

Our hypothesis is that students using the ET with adaptive link annotation (ie using the adaptive version of the ET) will show paths which are "more purposeful" (less nodes visited, greater average time spent on each node, less movements to unrelated nodes),

with increased comprehension (better results on tests), than those using the same ET without the adaptive link annotation.

Hook [HREF3] draws attention to the fact that the interface design is often inextricably linked to the adaptive component in an adaptive system. Removing adaptivity may remove a natural part of the system and its intended use. Other studies (Boyle and Encarnacion, 1994; Kaplan et al., 1993) have also centred on "with and without adaptivity". These experiments have offered some straightforward, if oversimplified, results in favour of adaptivity, but have not convincingly separated a range of confounding variables, such as those for which Hook argues. These can be deceptively simple. For instance, if we ask a student why they followed the annotated links, we may find that it is because they thought it would be the easiest way to get through the session! There is a clear need to integrate the results of a questionnaire, along with the hard numerical data from the audit trails, to make sense of experimental outcomes. Just as the evaluation of the tool cannot be separated from its development, measuring the effectiveness of its components cannot be successfully achieved in isolation from broader human factors.

In terms of these considerations for Interbook, the interface is reasonably complex although it does provide an easy entry level, as learners may use the hyperlinks like a book index, of which they will be reasonably familiar, or just click the "continue link" to read in a linear way. Fortunately also, the visible adaptive component of Interbook, namely the simple coloured bullets next to the links, is a straightforward addition to the interface and should not confuse the fact that we are measuring the addition or exclusion of adaptive navigation support with a measurement of the effects of changes in the interface. We also intend a number of sessions before the trial where students may familiarise themselves with the interface to minimise this affect.

Some early results will hopefully be available at the time of the presentation of this conference paper.

Conclusion

In this paper we have specifically discussed adaptive navigation support (ANS) on the Web, describing some reasonably simple yet effective ANS methods. InterBook provides an authoring and delivery mechanism into which the content of a book may be structured through the identification of key domain concepts to create an adaptive electronic textbook (Schwarz, Brusilovsky, and Weber, 1996). Based on the user's path through the structured hyperspace using a domain model and a user model, InterBook annotates links as visited, learned, ready-to-be learned and not ready-to-be learned, integrating the history-based, knowledge-based and prerequisite-based methods of ANS. We have outlined the authoring process with Interbook and our plans for an evaluation study. Interbook is based on the principle of adaptive navigation support, that is, individual navigation advice provided to the student, without removing or limiting their freedom to browse, on the basis of where they have been, where they are now, and what is to be learned.

References

Boyle, C. & Encarnacion, A. O. (1994). MetaDoc: an adaptive hypertextreading system. *User Models and User Adapted Interaction,* 4(1),1-19.

Brusilovsky, P. (1996) Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction* 6 (2-3), 87-129.

Brusilovsky, P. and Pesin, L. (1994) ISIS-Tutor: An adaptive hypertext learning environment. *Proceedings of JCKBSE'94, Japanese-CIS Symposium on knowledge-based software engineering.* Edited by H. Ueno and V. Stefanuk. Pereslavl-Zalesski, Russia, May 10-13, 1994, pp. 83-87.

Brusilovsky, P. and Pesin, L. (1995) Visual annotation of links in adaptive hypermedia. *Proceedings of CHI'95 (Conference Companion).* Edited by I. Katz, R. Mack and L. Marks. Denver, May 7-11, 1995, pp. 222-223.

Brusilovsky, P., Pesin, L., and Zyryanov, M. (1993) Towards an adaptive hypermedia component for an intelligent learning environment. In *Human-Computer Interaction,* Lecture Notes in Computer Science, Vol. 753, L. J. Bass, J. Gornostaev and C. Unger (eds), Springer-Verlag, Berlin. pp. 348-358.

Brusilovsky, P., Schwarz, E., and Weber, G. (1996) A tool for developing adaptive electronic textbooks on WWW. *Proceedings of WebNet'96, World Conference of the Web Society.* San Francisco, CA, October 15-19, 1996, pp. 64-69, http://www.contrib.andrew.cmu.edu/~plb/WebNet96.html.

Kaplan, C., Fenwick, J., & Chen, J. (1993). Adaptive hypertext navigationbased on user goals and context. *User Models and User AdaptedInteraction*, 3(3), 193-220.

de La Passardiere, B. and Dufresne, A. (1992) Adaptive navigational tools for educational hypermedia. In *Computer Assisted Learning,* I. Tomek (ed) Springer-Verlag, Berlin. pp. 555-567.

Eklund, J. and Zeiliger, R. (1996) Navigating the Web: Possibilities and practicalities for adaptive navigational support. *Proceedings of Ausweb96: The Second Australian World-Wide Web Conference.* , pp. 73-80.

Hammond, N. (1989). Hypermedia and learning: Who guides whom? In H. Maurer(Ed.), *2-nd International Conference on Computer Assisted Learning,ICCAL'89* (Vol. 360, pp. 167-181). Berlin: Springer-Verlag.

Hohl, H., Bšcker, H.-D., and GunzenhŠuser, R. (1996) Hypadapter: An adaptive hypertext system for exploratory learning and programming. *User Modeling and User-Adapted Interaction* **6** (2-3), 131-156.

Ibrahim, B. and Franklin, S. D. (1995) Advanced educational uses of the World-Wide Web. *Computer Networks and ISDN Systems* **27** (6), 871-877.

Perez, T., Gutierrez, J., and Lopisteguy, P. (1995) An adaptive hypermedia system. *Proceedings of AI-ED'95, 7th World Conference on Artificial Intelligence in Education.* Edited by J. Greer. Washington, DC, 16-19 August 1995, pp. 351-358.

Rubin, C (1996) *The Macbible Guide to Clarisworks 4.* Peachpit Press. California.

Schwarz, E., Brusilovsky, P., and Weber, G. (1996) World-wide intelligent textbooks. *Proceedings of ED-TELECOM'96 - World Conference on Educational Telecommunications.* Boston, MA, June 1-22, 1996, pp. 302-307, http://www.contrib.andrew.cmu.edu/~plb/ED-MEDIA-96.html.

Vassileva, J. (1996) A task-centered approach for user modeling in a hypermedia office documentation system. *User Modeling and User-Adapted Interaction* **6** (2-3), 185-224.

Weber, G. & Specht, M. (1997, in press). User modeling and Adaptive Navigation Support in WWW-based Tutoring Systems. *Paper to be presented at UM97, The Sixth International Conference on User Modeling*, Chia Laguna, Sardinia , Italy, June 2-5 1997.

Zeiliger, R. (1993) Adaptive testing: contribution of the SHIVA model. In *Item banking: Interactive testing and self-assessment,* NATO ASI Serie F, Vol. 112, D. Leclercq and J. Bruno (eds), Springer-Verlag, Berlin. pp. 54-65.

Hypertext References

HREF1
The project home page for the evaluation of Interbook
http://www.education.uts.edu.au/projects/interbook

HREF2
Interbook Home Page
http://www.contrib.andrew.cmu.edu/~plb/InterBook.html

HREF3
Hook K. Evaluating Adaptive Systems: Some Problems
http://www.sics.se/~kia/evaluating_adaptive_systems.html

HREF4
Visual annotation of links in adaptive hypermedia
http://www.acm.org/sigchi/chi95/Electronic/documnts/shortppr/plb_bdy.htm

HREF5
RTFtoHTML
http://www.sunpack.com/RTF/alpha3.htm

HREF6
ELM-ART
http://www.psychologie.uni-trier.de:8000/projects/ELM/elmart.html

HREF7
Adaptive Hypertext and Hypermedia Publications
http://www.education.uts.edu.au/projects/ah/publications.html
HREF8
Wbt Systems Home Page
http://www.wbtsystems.com/

Copyright

AHA! The Next Generation

Paul De Bra, Ad Aerts, David Smits, Natalia Stash
Department of Computer Science
Eindhoven University of Technology
Eindhoven, The Netherlands
+31 40 2472733
debra@win.tue.nl

ABSTRACT
AHA! is a simple Web-based adaptive engine, that was originally developed to support an on-line course. This paper describes AHA! version 2.0, a new major release that aims to significantly increase the adaptive versatility of AHA! without sacrificing AHA!'s simplicity that makes it easy to use. The new features in AHA! are inspired by AHAM [4], a Dexter [6] based reference model for adaptive hypermedia systems.

ACM Categories
H5.2 (user interfaces), H5.4 (hypertext/hypermedia)

General Terms: Design, Experimentation, Human Factors

Keywords: Adaptive hypermedia, condition-action rules,
adaptation engine.

1. INTRODUCTION

In 1994 we started a course on Hypertext, consisting of lectures and a Web-based course text. From 1996 on the lectures were discontinued and the course text was augmented with *adaptive content and linking*. Our adaptive software later became the AHA! system [3, 5], for **A**daptive **H**ypermedia **A**rchitecture. AHA! was made available as Open Source. Its simple architecture (based on Java Servlets, and XML and HTML for the pages) made it possible to add adaptation to very different applications. AHA! has been studied and used independently by several researchers/
educators from different parts of the world.In 1999 we developed a reference model for adaptive hypermedia,
called AHAM (Adaptive Hypermedia Application Model) [4,7]. Based on the formal Dexter model for hypermedia [6] AHAM provides a framework to express the functionality of most adaptive hypermedia systems (or ahs). See [3] for a recent overview of existing ahs and adaptive features. AHAM provided most of the inspiration for the additions to AHA! that have resulted in version 2.0, described in this paper.

In this paper we shall only describe AHA! 2.0 features and properties. Information on older versions can be found in previous papers about AHA!, including [5]. Different versions of AHA! and documentation about them can be found at the Website http://aha.win.tue.nl/.

2. BRIEF OVERVIEW OF AHA!

AHA! is a typical adaptive Web-based system, in the sense that each time the user requests a page (by following a link) some server-side software is activated that performs the following steps:

1. The requested page is retrieved from the local file system or from a remote source. (AHA! can load pages from other web servers through HTTP and adapt them just like local pages.)
2. The system loads a *domain model* of the application (or has it in memory already). This model reveals how the requested page relates to other pages or to higher level *concepts*.
3. The system loads a *user model* (or has it in memory). This model contains some of the user's aspects such as preferences, and an *overlay model* that shows how the user relates to the pages and concepts of the domain model. In AHA! the user model consists of a set of attribute/value pairs for each concept or page. The author can define and name arbitrarily many attributes (of type string, integer or Boolean).
4. The system loads an *adaptation model* (or has it in memory). It contains rules that are used to update the user model based on the access to the requested page, and rules to adapt the presentation of the page to the individual user. The rules are explained in Section 3.
5. The system executes the rules, and thus updates the user model and presents the adapted page. The rule execution is explained in Section 4.

The AHA! system deviates from this general scheme (dictated by the AHAM model) in one detail: the *domain model* and *adaptation model* are stored together. According to Brusilovsky [3] there are two types of adaptation in ahs: *adaptive presentation* and *adaptive navigation support*. In AHA! these are implemented as follows:

Fragments in a page can be *conditionally included*. This is most useful to add (short) extra explanations for users who need them, or to remove unwanted elements from lists.

Links (actually, link anchors) can be shown in different colors. AHA! uses three colors, called *good*, *neutral* and *bad*. The color of a link anchor depends on the *desirability* of the link, which in turn represents the desirability of the destination (page) of the link. This desirability is expressed using the same kind of *condition* as used for the conditional inclusion of fragments. The *good* links, leading to desired pages, are shown in *blue* by default. The *neutral* color, *purple* by default, is used for links to desired pages that were visited before. The *bad* color, *black* by default, is used for links to *undesired* pages. AHA! disables the underlining of links, so link anchors with the same color as normal text are effectively *hidden*. The color scheme can be changed by the end-user, to make the *bad* links visible if the user wants this.

3. THE DOMAIN/ADAPTATION MODEL

Concepts and pages and their associated adaptation rules are represented using an XML file (stand-alone version) or a mySQL database table (server version). We give a part of a tiny imaginary example, in which a system has information about chocolate and beer. We use an alternative syntax (more compact than AHA!):

```
<concept>
<name>de-koninck</name>
<desc>Beer from Antwerp</desc>
<requirement>beer.interest > 20</requirement>
<attribute name="access" type="bool" isPersistent="false">
<desc>standard attr: true when page accessed</desc>
<generate>
<requirement>beer.interest < 100</requirement>
<trueAction>beer.interest += 10</trueAction>
</generate>
<generate>
<requirement>chocolate.interest >= 5 and
chocolate.interest < 50</requirement>
<trueAction>chocolate.interest -= 5</trueAction>
</generate>
</attribute>
<attribute name="interest" type="int" isPersistent="true">
...
</concept>
```

The concept "de-koninck" corresponds to a page about the first author's favorite beer. This page is only *desirable* if the user's interest in beer is already greater than 20. (Links to the page will be *bad* if the interest is too low and *good* or *neutral* otherwise.) Each page has a volatile attribute, called "access" that is false by default and becomes true temporarily when the page is accessed. Within certain limits ready about de-konink raises the interest in beer and decreases the interest in chocolate. The "interest" of dekoninck is a persistent attribute, meaning that it is permanently stored in the user model. It too can have *conditional actions* to update attributes of other concepts. Updates can be based on constants (as in the example) or on the update to the rule's attribute. (Instead of the constant amount 10, a part of a change ininterest in de-koninck can be *propagated* to beer.interest.)

4. THE AHA! ADAPTATION ENGINE

The adaptation rules in AHA! are *condition-action* rules, like rules studied in the field of active databases [1]. The adaptation engine maintains a queue of pending rule instantiations. Whenever an attribute value of some concept is modified the adaptation rules for which the requirement (condition) is true (at that time) are added to the queue. When a user accesses a page (by following a link) the "access" attribute for that page becomes true. The adaptation rules associated with this attribute are the first ones examined and put in the queue (if their condition is true). The process of rule executions (and adding more rules to the queue) continues until the queue is empty and thus there are no more rules to execute. From research in active databases [1] we know that the rule execution process is not guaranteed to *terminate*. In [7] we showed how such problems can be detected at authoring time. (AHA! currently does not yet warn authors about this.)

5. CONCLUSIONS / FUTURE RESEARCH

The "new" AHA! user model and adaptation engine greatly improve the versatility of AHA!. Especially arbitrary adaptation rules (unlike simple propagation of knowledge in educational applications) are easier to express, as the example in section 3 has shown. The extensions are based on the rule system presented in [7] for the AHAM reference model. In the future we want to extend AHA! by borrowing more ideasfrom the AHAM:

links to concepts (not just pages), and a method to "select" the best page to present when following such a link;

a way to express *generic* rules, so that rules don't need to be replicated for every page or concept they apply to.

6. ACKNOWLEDGEMENT

The development of the AHA! system is supported by a grant of the NLnet Foundation, through the "Adaptive Hypermedia for All!" project (conveniently abbreviated to AHA!).

7. REFERENCES

[1] E. Baralis, and J. Widom. An algebraic approach to static analysis of active database rules. ACM Transactions on Database Systems, Vol. 25, nr. 3, pp. 269–332, 2000.

[2] Brusilovsky, P. Adaptive Hypermedia, User Modeling and User-Adapted Interaction, Vol. 11, nr. 1–2, pp. 87–110, Kluwer academic publishers, 2001.

[3] De Bra, P., A. Aerts, G.J. Houben, and H. Wu. Making General-Purpose Adaptive Hypermedia Work. Proceedings of the AACE WebNet Conference, pp. 117–123, San Antonio, Texas, 2000.

[4] De Bra, P., G.J. Houben, and H. Wu. AHAM: A Dexterbased Reference Model for Adaptive Hypermedia. Proceedings of the ACM Conference on Hypertext and Hypermedia, pp. 147–156, Darmstadt, Germany, 1999.

[5] De Bra, P. and Calvi, L., AHA! An open Adaptive Hypermedia Architecture. The New Review of Hypermedia and Multimedia, vol. 4, pp. 115-139, Taylor Graham Publishers, 1998.

[6] Halasz, F., and M. Schwartz. The Dexter Hypertext Reference Model. Communications of the ACM, Vol. 37, nr. 2, pp. 30–39, 1994.

[7] Wu, H., E. De Kort, and P. De Bra. Design Issues for General-Purpose Adaptive Hypermedia Systems. Proceedings of the ACM Conference on Hypertext and Hypermedia, pp. 141–150, Århus, Denmark, 2001.