AHA! Adding Adaptive Behavior to Websites

Paul De Bra, Natalia Stash, Barend de Lange
{debra, nstach}@win.tue.nl, barenddelange@hetnet.nl

Eindhoven University of Technology (TU/e)
Department of Computing Science
P.O. Box 513, Eindhoven, The Netherlands
Tel: +31 40 2472733, Fax: +31 40 2463992

Abstract. Most websites offer information that is potentially interesting to a wide audience. However, a single presentation may not be suitable for the wide range of people wishing to visit the website. AHA! is a server extension that makes it possible to adapt the content shown on webpages, and the links that appear on these pages, to each individual user. An AHA! application consists of a concept-structure and a set of XHTML pages. The concept-structure (which can be defined using a high-level graphical tool or a low-level editor) consists of concepts, attributes and condition-action rules that determine user model updates and conditions for object inclusion. The XHTML pages use the "object" tag to indicate where conditionally included objects should be placed. The condition-action rules determine which information is included, if any. The rules also determine the "suitability" of pages for a given user. Links to pages are shown in a different way depending on this suitability. Many adaptive systems exist today. They target a specific application area, like on-line textbooks, information kiosks, corporate information systems, etc. AHA! provides a general-purpose adaptive system. Current research and development efforts concentrate on improving the authoring support, e.g. by automatically detecting potential loops in the condition-action rules. In this paper we refer to upcoming version of the system AHA! 3.0.

Keywords: adaptive hypermedia, user model/profile, Website personalization

1 Introduction

Website personalization has been a key research issue over the last few years. This is caused by the need to adapt the content and presentation style for a given user or set of users. The first step in being able to provide an individualized approach is to identify users' characteristics. One way is for users to identify their characteristics themselves. They can do it manually, selecting desired categories of information, stating their preferences by answering questions and filling out registration forms. Another approach is observing the user's browsing behavior. In this paper we concentrate on *automatic personalization* based on user actions. The resulting websites are often called *adaptive*.

With the AHA! system we provide an environment that can be used by website developers to add adaptive behavior to their pages. A website may have multiple variants of the same resource. Based on the information about the user the system presents the "most appropriate" variant of the resource. In the area of adaptive web-based systems different techniques can be combined to increase the range of possible presentations: a system can produce different versions by including (different) embedded objects, but the same content can also be delivered in a different way by selecting a different presentation style (using stylesheets) and by conditionally hiding or annotating link anchors. Such adaptive response might result in the user being more satisfied with the quality of the resource. The aim of the system is to guide the user's navigation and support him with adaptive annotation but at the same time let him move freely through the pages of an application. Adaptation is not intended to restrict users but rather to guide them. However, the guidance versus restriction issue depends on the author(s) of a website, they are not an inherent property of the techniques that are used. In AHA! adaptation is based on *concept structures* and *adaptation rules* that are defined by an author of an application.

Many adaptive Web-based (and non-Web-based) applications exist to date. For more information see (Brusilovsky, 1996, 2001), and conference and workshop proceedings on adaptive hypermedia and adaptive Web-based systems. Most of these systems are special-purpose – aimed at a single application or application area. Using them for different kinds of applications, like information kiosks, corporate websites, museum websites and learning applications is not possible. Nevertheless, the basic types of user modeling and adaptation required in all these applications is not all that different. With AHA! we try to provide a *general-purpose* Web-based adaptive system allowing many kinds of adaptation rules, more types of relationships between concepts and more presentation freedom.

This paper is organized as follows. In the next section we show the overall AHA! architecture and describe how an adaptive presentation is generated for the end user. Section 3 describes the authoring

process for such a presentation. We conclude the paper with a discussion of related work and plans for future work.

2 Adaptive behavior in AHA!

The complete AHA! architecture is shown in figure 1.

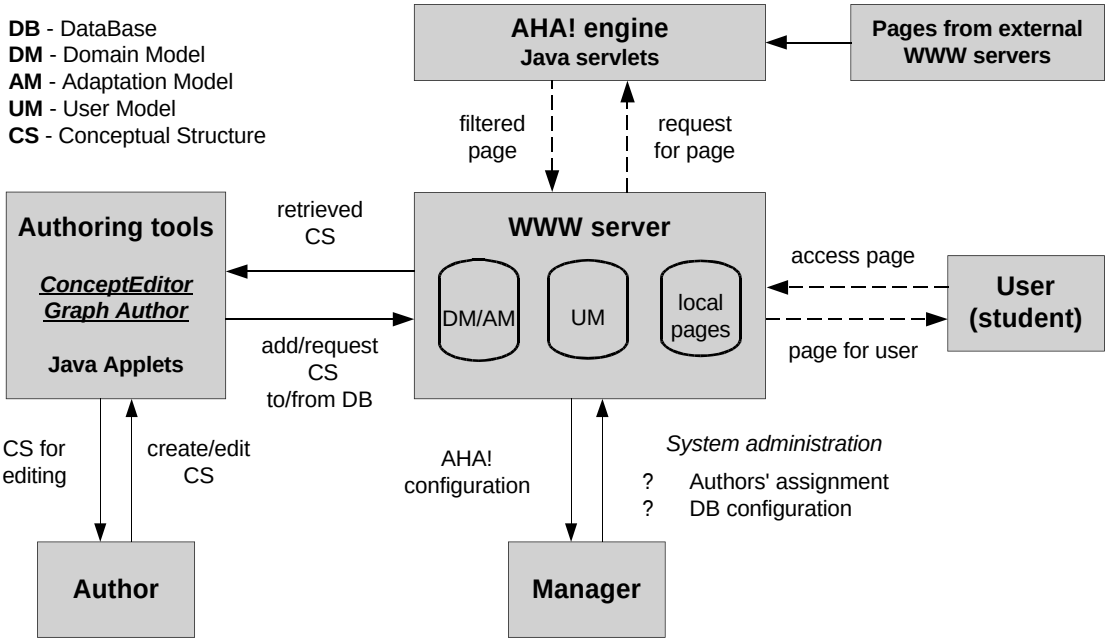

Figure 1. General AHA! Architecture.

AHA! is a server-side extension that provides adaptation functionality. It allows Website developers to add automatic personalization to their pages. The adaptation is based on information that is stored in a so-called "user model". The adaptive engine consists of Java servlets that are activated when the Web-server receives HTTP requests from the browser. This server-side extension is *generic*, i.e. application independent, and *open*, meaning that the source of the information may itself be external to the server running the adaptive engine. (AHA! then works more or less as a proxy, retrieving the external resource and applying its adaptation to the page before forwarding it to the end-user's browser.)

The following users interact with the system:

Manager – system administrator – registers "authors" who want to create their adaptive applications using AHA!. He also performs the initial AHA! configuration.

Author – AHA! author – creates a structure of the (application) domain model consisting of concepts and adaptation rules. An author can design his application using one of the authoring tools – the low-level "Concept Editor" or the high-level "Graph Author" (described in sections 3.1.1 and 3.1.2). Besides this structure the author also has to create (write) the application content, usually consisting of a set of xhtml pages.

User – the end-user of an application. Interaction between him and the system is performed in the following way:

1. The user accesses a page.
2. The user model is updated, based on that page access
.
3. The page may conditionally include objects that are accessed automatically.
4. The user model is updated for each access to an included object.
4. The object's (re)presentation is determined.
5. Go to step 3 if the object includes other objects, and repeat until there are no more objects to process.
6. The page is adapted on the basis of the information in the user model and the result of each object.
7. The adapted page with its objects (and conditionally included fragments) is sent to the browser.

AHA! works through the use of Java servlets and can adapt local or remote pages to a user. In the domain model some (most) concepts

are associated with a *resource*. The adaptation rules for the concept are activated/executed when that resource is accessed. Both local and remote webpages can be associated with a concept.

The domain/adaptation model and the user models (for all the users) can be stored as xml files on the server, or in a mySQL database. The manager chooses between file and database storage. AHA! can convert the files from xml files to mySQL and vice versa.

When a user logs in for the first time a user model is created. For each concept in the domain model the user model contains attribute-value pairs. It also contains some user-related information that is not about the domain model of an application. This information is represented through the attributes of a concept which is called "personal". Just like for each normal concept the author can assign multiple attributes the "personal" concept, to take into account various domain-independent user characteristics and to provide adaptation on their basis. For example, it becomes possible to design different presentations for people from different age groups, with a different knowledge level in the subject domain, to design more terse (like viewgraphs) or more verbose (like full course text) presentations depending on the user's preferences, etc..

When a user requests a page the AHA! servlet processes the page and sends a filtered page to the browser – with fragments or objects conditionally included or excluded and with modified links (actually link colors). The conditional inclusion of fragments is most suitable for including short explanations that some users may need (because they do not know a certain technical term for instance) and when other users would be bothered by these explanations (because of too much repetition). For fragments that may appear on different pages the author can opt to create a separate object that is conditionally included on each of these pages. Since version 3.0 AHA! can conditionally include fragments that are part of the page as well as fragments stored as separate objects. For link anchors AHA! uses three different colors to indicate the "desirability" of the links: *good*, meaning desirable and not previously visited, *neutral*, desirable but previously read, and *bad*, meaning not desirable. Depending on the choice of colors AHA! can be configured to use the *link hiding* technique (with black undesirable links) or the *link annotation* technique (with all links shown in visible, different colors). The default color scheme is *blue*, *purple* and *black*.

The following section describes the steps of the authoring process showing how the authors can add adaptive behavior to their Websites.

3 The Authoring Process

The authoring process consists of the following steps:

1. Defining Domain/Adaptation model.
2. Creating fragments/objects and their concepts.
3. Creating pages for an application.

To make the process of defining the domain/adaptation model easier AHA! provides two Java Applet based authoring interfaces. These tools manipulate a (server-side) XML file that can also be manually edited by a technically skilled author. However, the XML file describing the whole conceptual structure of an application is quite verbose and may be hard to read. Authoring tools hide the XML syntax from the author. The adaptation rule language of AHA! is described in detail in (De Bra et al, 2002a). The XML notations describing concept relationship types are presented in (De Bra et al, 2002b). As the author of an application does not need to know the XML syntax of the resulting file we will only show some examples of entering these conceptual structures using the "Concept Editor" and "Graph Author" tools (and not the corresponding XML files).

There is no AHA! authoring tool for the (xhtml) pages. Every author can use his own favourite web page authoring tool.

3.1 Defining the Domain/Adaptation Model

In order to create an adaptive application an author needs to design the overall conceptual structure. For most applications the domain model is a hierarchy of concepts. The relationships between these concepts give an idea of the order in which concepts should be presented to the users. The AHA! authoring tools allow an author to define the domain model for an application along with the adaptation model associated with it. This means that for each *concept* the author defines *requirements* as well as a set of *generate rules*. These rules determine how page accesses generate user model updates. Through *requirement rules* the author defines how AHA! performs the adaptation. These rules indicate under which circumstances a concept is "desired" (in other words, under which conditions the user is ready to "access" the concept). When a resource is associated with the concept the "desirability" is used to determine the color of (anchors of) links to the resource and to determine whether the resource should be included in the presentation when it is used as an embedded object. Fragments (within a page) can also have requirements

to determine their desirability. The desirability of fragments within the accessed page is used to decide which base-fragment must be included in the presentation of the page. (This is similar to the inclusion of embedded objects, but for objects the requirements are part of the concept structure and for a fragment the requirements are written on the page that contains that fragment.

Generate rules define a set of actions to be performed when the user "accesses" the (page/resource associated with the) concept. Each rule has a condition that is checked to see whether the associated action should be performed. It is also possible to specify an alternate action to be performed when the condition is not satisfied. Each rule is associated with an attribute of the concept, and the rule is "triggered" by an update to that attribute. For a page there are also rules that are triggered by an access to that page. The access is treated as an update to an access (pseudo-)attribute. The action of a rule can be the assignment of an absolute value to the destination concept's value, but it can also be a relative update, meaning that a (fixed) percentage of the update to the source concept's attribute value is propagated to the destination concept's attribute value. The adaptation engine keeps track of which attributes of which pages or concepts are updated by rule actions, and "triggers" the execution of their associated rules. This process continues until there are no more rules to execute. Potential problems which can be caused by rule execution include (non-)termination and (non-)confluence (the latter meaning that different results may be generated depending on the order in which triggered rules are executed). In (Wu et. al, 2001) we have shown how to predict some of these problems during the authoring process.

Since AHA! 2.0 (De Bra et al, 2002a) we provide a versatile concept structure. An author can use multiple attributes for each concept, such as "knowledge about", "interest in", etc. Apart from the system-defined "access" attribute for pages an author is free to choose attribute names at will. Attributes can have Boolean, numeric or string values. (Relative use model updates as described above are only possible for numeric values.)

Once the domain model is created and all the concepts and relations have been defined, the author has to associate resources to concepts. In the simplest case each resource is a concept and vice versa. But quite often "higher level" concepts should be taken into consideration. For designing adaptive applications containing these higher level concepts the author can use the graphical interface "Graph Author" (see Section 3.1.2). Once the author has created the conceptual structure the manager of the system converts the resulting XML file produced by one of the authoring tools to the internal format used by the system (XML or mySQL).

We will take as an example a domain model for "Beverages". The set of "unrelated" concepts representing the domain model is shown in figure 2. In the following subsections we show how to define relationships between these concepts using the "Concept Editor" and the "Graph Author".

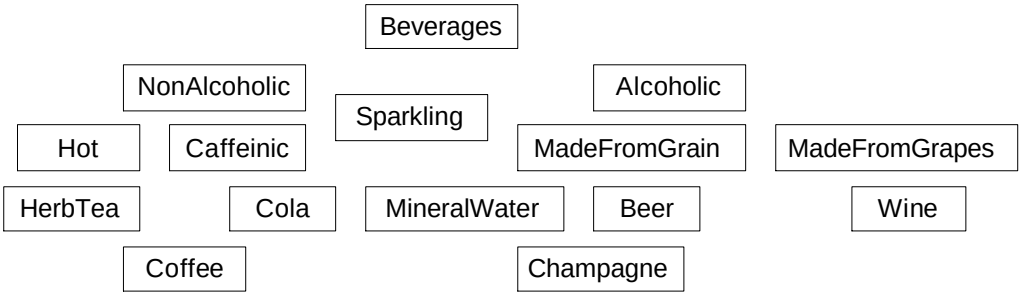

Figure 2. Concepts representing the domain model of "Beverages".

3.1.1 The Concept Editor

The "Concept Editor" or "Generate List Editor" is a low-level authoring tool. It is suitable for applications that require many different kinds of adaptation rules. If the application uses only a few types of rules, like having the access to a page raise knowledge about the concept associated with that page and also knowledge about higher level concepts, then the more user-friendly "Graph Author" described in the following section may be more appropriate. Using the "Concept Editor" the author has to write all the individual adaptation rules himself.

Let us take a concept "Cola" from the domain model and show how the author can define the adaptation rules for this concept using the "Concept Editor". Assume the concept is associated with a page "Cola.xhtml". The page becomes desirable when interest in concept "Caffeinic" is greater than or equals 50 and interest in concept "Sparkling" is greater than or equals 70. For adaptation this means that links to "Cola.xhtml" will be shown either in blue (before the first visit to the page) or purple (afterwards) for users with high interest in caffeinic and sparkling beverages. Links will be hidden by making them black as normal text and not underlined for uninterested users. In figure 3 we show a screen dump of the

"Concept Editor" for entering the conceptual structure of the domain, when entering an adaptation rule for the "Cola" concept.

The "Cola" concept has 3 attributes: "access", "knowledge" and "visited". When a user accesses "Cola.xhtml" the attribute *access* temporarily becomes true. This event causes the execution of actions associated with this attribute. The author can specify several actions, to be performed when the user accesses the page. In figure 3 we show 2 adaptation rules. For each rule we define a requirement (in the "If" field), a set of actions to be performed if the requirement is satisfied ("Then" field), and a set of alternate actions to be performed if the requirement is not satisfied ("Else") field. For actions we have a "Propagating" attribute that indicates whether this action is allowed to trigger other actions or not. Propagation is used for instance to transfer knowledge from pages to higher-level concepts.

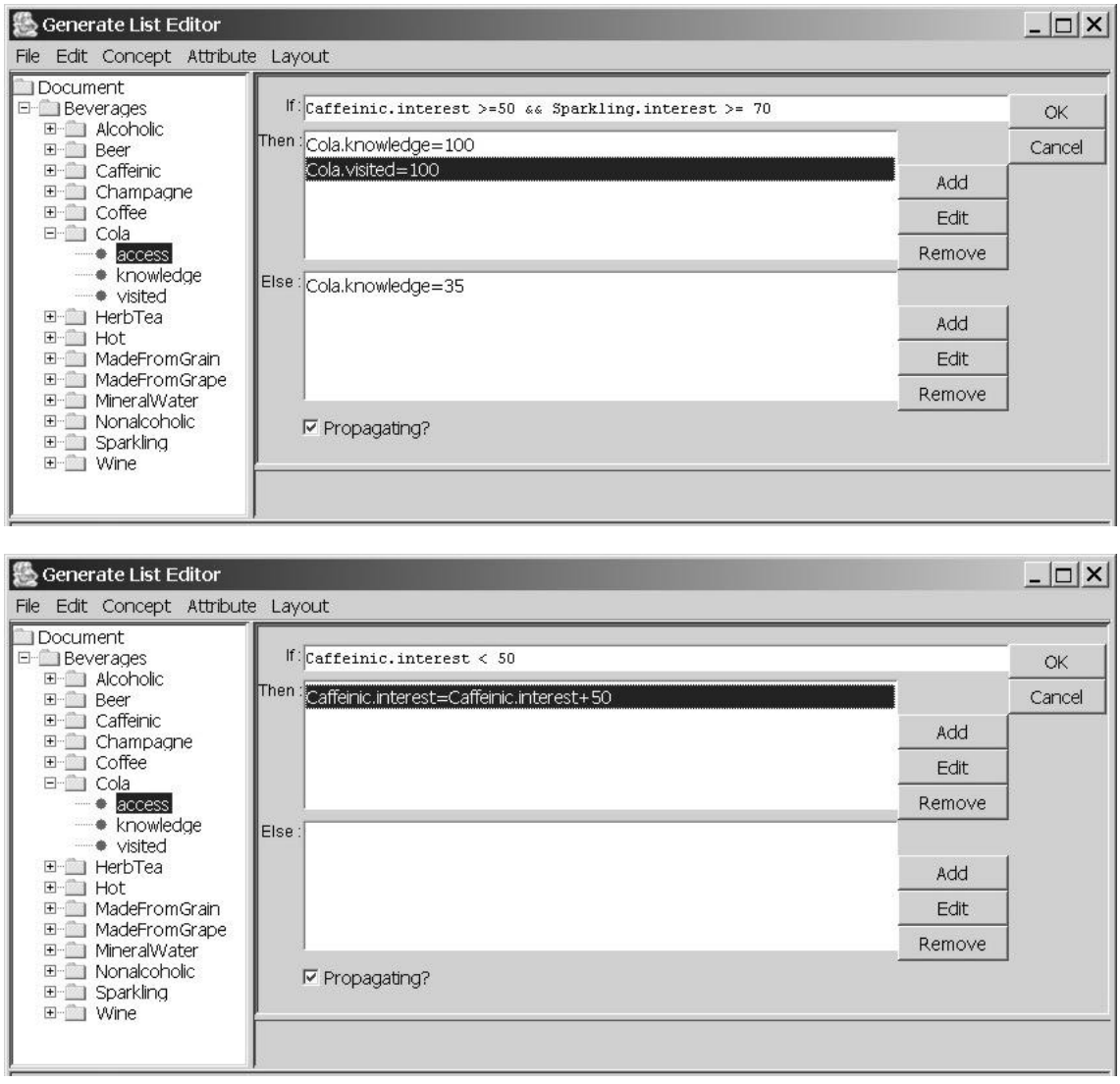

Figure 3. Entering conceptual structure with "Concept Editor".

The adaptation rule entered in the first screen dump specifies 2 possibilities for user accessing the page about "Cola":

If the page is desired (the requirement is true) the values of the "knowledge" and "visited" attributes are set to 100. In a learning environment this would mean that the system thinks the user has full knowledge of this concept, and that the user has visited the page completely.

If the page is not desired the "knowledge" attribute is set to 35. This value 35 was chosen arbitrarily. In a learning context it would mean that when the user reads a page for which (the system thinks) he is not ready yet, he gains only partial knowledge of the page, and the page is still not considered visited. (Actually, the visited status is left unchanged.)

The second screen dump shows that access to “Cola.xhtml” raises interest in “Caffeinic” beverages if this interest is not high enough. We can specify a similar adaptation rule for changing interest in “Sparkling” beverages.

A drawback of using the “Concept Editor” for defining rules is that frequently occurring concept relationships lead to a lot of repetitive work. Also, these relationships are described through their effect on the user model only, and are thus not visible as “relationships”. An author who is revisiting an application through the “Concept Editor” may not recognize which relationships are expressed by which adaptation rules and may thus not recognize the higher-level structure that is hidden in the large set of rules. Sometimes it can also be quite difficult to describe the intended user model and adaptation through the adaptation rules. The author’s task is made a bit easier by predefining a common structure for concepts. When the author adds a concept a default set of attributes and rules is created. This not only reduces the amount of work for the author, but it also helps to avoid common (beginners) mistakes. A side effect of the rule in figure 3 for instance is that when the “Cola” page is read and fully known, but becomes undesirable later (when somehow the knowledge of “Caffeinic” or “Sparkling” goes down) and the user revisits the “Cola” page, the knowledge of “Cola” goes down to 35. To avoid such problems the default *template* for a new concept contains a more appropriate set of adaptation rules.

3.1.2 The Graph Author

In (De Bra et al, 2002b) a new, higher level authoring tool is proposed. Typical concept relationships that occur in many applications are *hypertext links*, *prerequisite relationships* and *inhibitor relationships*. But for different applications other types of relationships may be used, for example *exemplifies*, *defines*, etc. In (De Bra et al, 2002b) we described how the notion of *concept relationship types* can be incorporated into the AHA! system (using an XML notation). In figure 4 we show a screen dump of the “Graph Author”, after entering a conceptual structure. The figure gives an example of *knowledge propagation relationships.*

With the “Graph Author” entering concept relationships becomes as simple as drawing a (labeled) graph. In figure 4 we see that, for example, the concepts “HerbTea” and “Coffee” both contribute knowledge to the concept “Hot”. The “Graph Author” automatically decides that both concepts contribute equally (50% each) to the knowledge of “Hot”. A knowledge propagation relationship can also have a value associated with it. Through these values we can show that knowledge contribution can be distributed unevenly. Each concept relationship is of a type that is selected from a list. A system designer (or skilled author) defines which types exist, and also defines the translation of concept relationships to AHA! adaptation rules. The “Graph Author” knows how to combine different relationships between the same concepts into correct AHA! adaptation rules that express the meaning of all the given rules. Concept relationship types can express *generate rules* as well as *requirement rules*. In (De Bra et al., 2002b) it is shown how a graph of concept relationships of different types is translated to the combined domain/adaptation model of AHA! 2.0. (For AHA! 3.0 this remains essentially the same.)

The “Graph Author” can show the graph for each concept relationship type separately or can overlay the graphs using different colors or line styles. Also, the example of figure 4 is somewhat atypical because knowledge propagation can also be (and is typically) expressed by turning the list of concepts, shown on the left, into a windows-explorer-like hierarchy. The “graph” part can then be used to created the concept relationships for all other types.

Note that the “Concept Editor” can read files generated by the “Graph Author”, but not vice versa. It is impossible to deduce from a set of adaptation rules the set of high level relationships that led to these rules.

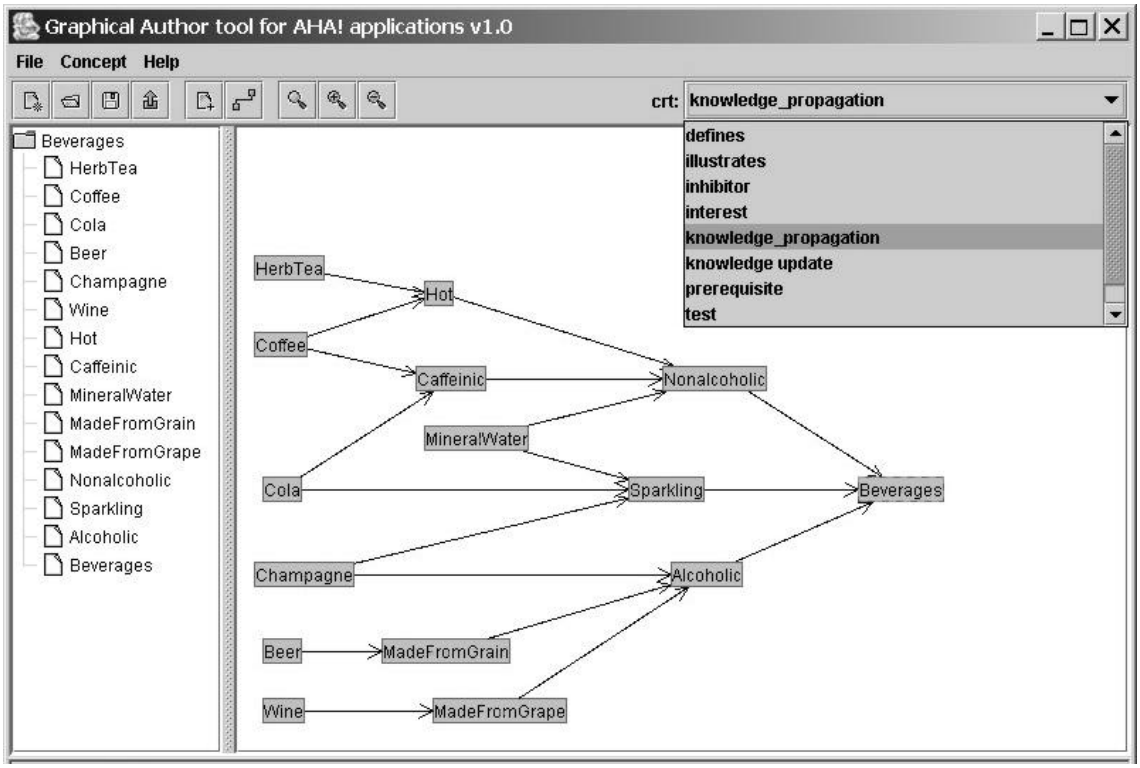

Figure 4. Entering a conceptual structure with the "Graph Author".

3.2 Creating fragments and pages for an adaptive application

The author of an application does not need to be the author of the pages that he associates with his application. One of the ideas of the AHA! system is to reuse existing fragments/pages that reside on local or remote servers and to add adaptive features to create an application with them. However, in many cases creating an application will consist of both defining the conceptual structure and writing the content.

For local documents several formats have been used in the past. All were based on HTML. The early generations of AHA! used HTML comments for items used by the adaptation engine, for instance an "if" statement to conditionally include fragments. Later the format was been changed to XML. CDATA constructs were used to distinguish between the XML adaptation constructs and the HTML content. (The only common element was the "<a>" tag used by AHA! and HTML). The current version of AHA! can still work with this old format, but now the local pages can also be written in (standard) XHTML, (possibly) augmented with a module for AHA!-specific tags. These tags can be used for the generation of optional headers and footers and for the conditional inclusion of fragments. The AHA! engine looks at AHA! tags to perform the adaptation.

3.2.1. Creating Fragments

In AHA! 3.0 it becomes possible for fragments to autonomously update the user model (e.g. increase the knowledge of an item) and determine its own presentation. For this we use the standard <object> tag to identify fragments in XHTML pages and to turn them into resources that are associated with a concept. The behavior of a fragment is described in the concept, linked to that fragment. This concept describes under which conditions which base-fragment is included into the page. A base-fragment is a well-formed (xhtml) document that is scanned for the inclusion of more fragments. For example, there may be some general information about "Sparkling", which is (conditionally) included in the pages about Cola, Champagne and MineralWater. This fragment "Sparking" describes how to make a beverage sparkling. Once a reader has read this part of information it is not necessary to read exactly the same information again. A short summary could be enough. This adaptation should occur on every page that includes the "Sparkling" information.

To use adaptive fragments, two steps have to be taken. First a concept must be created for this fragment. This can be done using the low level as well as high level authoring tool. Second, the fragment must be included (as an object) in the XHTML page.

In the low level authoring tool an attribute "showability" is added. This attribute describes under which condition which object must be inserted into the XHTML page. Let us revisit the beverage

example. We define a fragment “Sparkling” with the following behavior: the first time a reader sees this fragment the XHTML file “sparkling_firsttime.html” is inserted in the position of the fragment identifier. The next time “sparkling_summary.html” is inserted. This behavior can be described using adaptation rules. In the beginning a reader has no knowledge of sparkling. Once he has seen this fragment, the knowledge is increased to 100, using the access attribute of this concept. The AHA! editors provide the possibility to connect conditions to return-fragments. A rule for the above example could be:
if Sparking.knowledge==0 then include “sparkling_firsttime.html” and
if Sparkling.knowledge==100 then include “sparkling_summary.html”.

Figure 5 shows the above defined constructions:

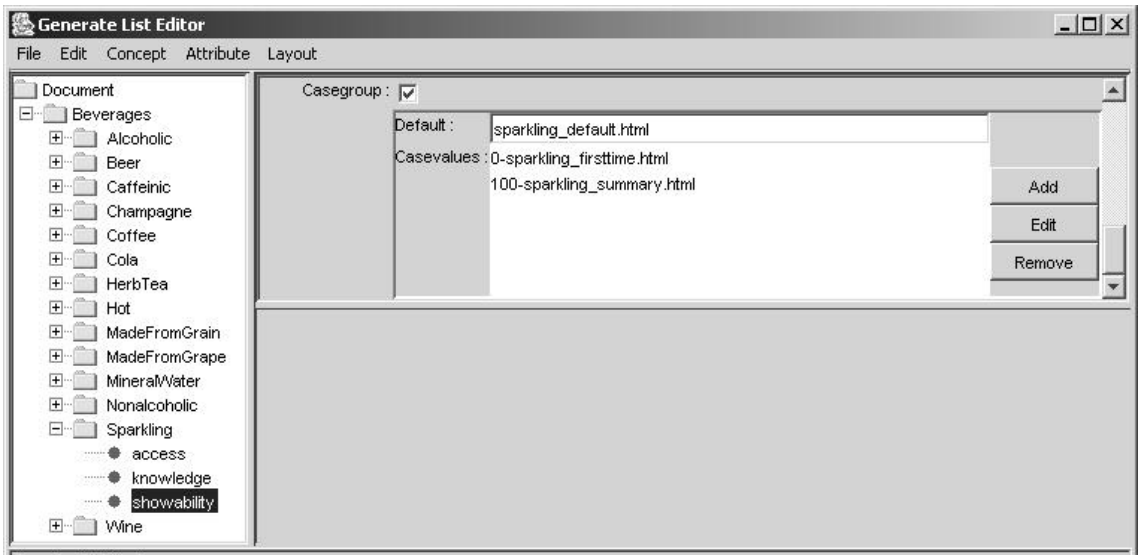

Figure 5. Entering case values connected to base fragments with the “Concept Editor”.

The second step is to insert this fragment into the pages about Cola, MineralWater and Champagne. This is done by inserting a simple “object” tag into the XHTML files:
<object name=”Sparkling” type=”aha/text”> </object>.

Most (modern) HTML editors have some support for the <object> tag because it is a standard (X)HTML tag. This new way of handling fragments has an important advantage over inserting the fragment text into the page: when a fragment is used many times in one adaptive application, it only needs to be defined/maintained in one place. Another big advantage is the use of a standard XHTML DTD. When no special AHA! tags are used, standard XHTML can be used. Standard (X)HTML editors do not understand the special “if” tags used in the AHA! 2.0. The final advantage is the possibility to use recursive fragments. Pages are created on the fly. Each time a fragment is found on a page, the AHA! engine determines its presentation and scans the resulting fragment for more (recursively) included fragments and so on till the end of the page (this is done in depth first order). After processing all the inclusions the page is presented to the user. This way of using fragments enables the creation of large and complex adaptive websites without the need to know the whole application up front.

3.2.2. Creating Pages

We have not provided any authoring tool for creating pages as the author can use any text or XML (XHTML) editor. For AHA! 2.0, with fragments embedded in the pages the author has to know how to use special AHA!-related tags. The most important tags for adaptation are the “<if>”, “<object>” and “<a>” tags. In an “if” tag the author defines an expression which is evaluated, and then the corresponding block of text will be included. The syntax for using “if” tags is the following:

```
<if expr="Caffeinic.interest &gt; 50 &amp;&amp; Sparkling.interest &gt; 70">
        <block>block of the text to be presented if the expression is fulfilled</block>
        <block>block of the text to be presented if the expression is not fulfilled</block>
</if>
```

This is useful for small fragments that are not used more than once in an adaptive application. For more flexibility the <object> tag, described above can be used. The syntax for using the “object” tags is the following:

<object name="coursename.fragmentname" type="aha/text"> </object>.
The "aha/text" type tells the AHA! engine that this is a conditionally included object.

With the "a" tag we use the "class" attribute to distinguish between different types of links. For example: <a href="Cola.xhtml" class="conditional">.

AHA! recognizes the link classes *conditional* and *unconditional*. When a link is of the class *conditional* the AHA! engine translates the class to *good*, *bad* or *neutral* depending on whether the destination of the link is a desired, undesired or uninteresting (desired but visited) page. This results in different link colors (defined in a stylesheet).

AHA! filters the pages (by including the appropriate fragments and also by generating the optional header and footer) and sends them in XHTML format to the browser.

4 Conclusions and Future Work

We continue to improve and to extend the functionality and usability of AHA! for both authors and end-users. This paper has presented the latest authoring tools for AHA! 3.0, and the development of a richer method for the conditional inclusion of fragments.

Many ideas for future improvements exist and will be worked on. Concerning the authoring process, we are going to complement authoring tools with the ability to analyse the correctness of information entered by the author. The tools will be extended with static analysis methods to avoid potential problems of (non-)termination and (non-)confluence. It will help authors in creating valid rule sets and warn them about pages that never become desired, fragments that are never included, etc. In the future we will also develop an interface for defining new concept relationship types and add the ability to author adaptation rules at the high (graph) level and the low level of AHA! at the same time. In the current version (2.0) we can use AHA!-specific tags for creating adaptive webpages but this is not necessary anymore using the standard "a" and "object" tags (in version 3.0). We believe that all these issues will make the authoring process easier and will help authors to develop better and more versatile presentations.

Concerning the end-user side, we are thinking about other kinds of adaptivity which can be added to existing ones. One of the research issues is creating "stable" presentations. This means that in some cases users want to return to previously visited pages and see them in the way they were before, not adapted to the current, modified state of the user model. Thus we are working on providing the possibility to keep pages unchanged in accordance with user requirements, for example during some period of time (like one session). Another issue is providing adaptation to users' cognitive styles. At this moment most of the work in this field is of an empirical nature. We will also create more applications using AHA! in order to be able to evaluate the user acceptance of adaptive Websites in general and adaptive AHA! applications in particular.

Acknowledgements

AHA! is an Open Source project funded by the NLnet foundation. Publications about AHA! and new releases can be found on the project website http://aha.win.tue.nl/.

References

Brusilovsky, P. (1996) Methods and Techniques of Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, 6, pp. 87-129, 1996.

Brusilovsky, P. (2001) Adaptive hypermedia. *User Modeling and User Adapted Interaction*, 11 (1/2) pp.87-110, 2001.

Calvi, L., Cristea, A. (2002). *Towards Generic Adaptive Systems: Analysis of a Case Study*. Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Springer Verlag, LNCS 2347, pp. 77-87.

Cini, A., Valdeni de Lima, J. (2002). *Adaptivity Conditions Evaluation for the User of Hypermedia Presentations Built with AHA!*. Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Springer Verlag, LNCS 2347, pp. 490-493.

De Bra, P., Aerts, A., Houben, G.J., Wu, H. (2000). Making General-Purpose Adaptive Hypermedia Work. *Proceedings of the AACE WebNet 2000 Conference*, pp. 117-123.

De Bra, P., Aerts, A., Smits, D., Stash, N. (2002a). AHA! Version 2.0, More Adaptation Flexibility for Authors. In *Proceedings of the AACE ELearn'2002 conference*, October 2002, pp. 240-246.

De Bra, P., Aerts, A., Rousseau, B. (2002b). Concept Relationship Types for AHA! 2.0. In *Proceedings of the AACE ELearn'2002 conference*, October 2002, pp. 1386-1389.

De Bra, P., Ruiter, J.P. (2001). AHA! Adaptive Hypermedia for All. *Proceedings of the AACE WebNet Conference*, pp. 262-268.

Romero, C., De Bra, P., Ventura, S., de Castro, C. (2002). Using Knowledge Levels with AHA! for Discovering Interesting Relationships. *Proceedings of the AACE ELearn'2002 Conference*.

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