Building Adaptive Presentations with AHA! 2.0

Natalia Stash, Paul De Bra
{nstach, debra}@win.tue.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. An increasing number of Website developers try to provide some kind of "personalization". In this paper we show how the designers (called "authors" in this paper) can build their adaptive presentations using the AHA! system (for **A**daptive **H**ypermedia **A**rchitecture). Most existing Adaptive Hypermedia Systems are *special-purpose*, aimed at one specific application or application area (mostly educational applications). AHA! also started out as an educational system but now it is being turned into a *general-purpose* tool through a grant from the NLnet Foundation. AHA! aims at bringing adaptivity to a wide variety of applications such as on-line information systems, on-line help systems, museum and shopping websites, in addition to the area of on-line textbooks. In this paper we first present the overall architecture of the system. Then, we describe the main steps of the authoring process. In this part we present two new authoring interfaces which support the author in defining the domain/adaptation model. Finally, we show how the adaptive application is presented to the end-user. Information about AHA!, including an adaptive tutorial, can be found on the AHA! website http://aha.win.tue.nl/.

Keywords: adaptive engine, user model/profile, concept structure, authoring tools, graphical interface, web-based application

1 Introduction and Background

Adaptive hypermedia is a fairly new research domain (Brusilovsky, 1996). At the Eindhoven University of Technology the development of the AHA! system dates back to 1996, when an on-line course text on the subject of hypermedia was augmented with adaptive content and linking. Since then the software has been changed and extended, which lead to AHA! version 1.0 (De Bra et al, 2000) and (De Bra & Ruiter, 2001). This version has been studied and experimented with by several research groups in different countries (Cini & Valdeni de Lima, 2002), (Calvi & Cristea, 2002), (Romero et al., 2002). In this paper we are going to refer to the current version of the system - AHA! 2.0.

With AHA! 2.0 we have provided more versatile structures for the user model, the concepts and the adaptation rules. It is now possible to take into account such user characteristics as interests, preferences, goals, learning style, etc. This may considerably increase the efficiency of the user interaction with the system. Each user aspect can be represented through different concepts or through different attributes of concepts. We can distinguish adaptation aspects that are not related to each other but also "combine" related adaptation aspects such as *knowledge about* and *interest in* the same concept. AHA! 2.0 uses a combined domain/adaptation model to indicate how concepts relate to each other and how the system should update the user model. AHA! 2.0 applications can present and adapt webpages from a local server as well as pages from other webservers, as long as these external pages are "known" in the concept structure of the application.

A lot of effort has been put into the development of authoring support, to ensure the usability of AHA! for adaptive Web-based applications. Two authoring tools have been developed for defining the domain/adaptation model – a low-level rule editor and a high-level graphical tool.

AHA! provides two kinds of adaptation: fragments of a page can be conditionally included or excluded, and links can be conditionally hidden or presented in a different color. In the following sections we will describe the process of building an adaptive application and we also briefly explain how the adaptation can be seen in the presentation to the end-user. To describe this process it is necessary to first show how the whole system works and how the different system components are related to each other. Therefore in the next section we present the general AHA! architecture.

2 AHA! Architecture

The complete AHA! Architecture is shown in figure 1.

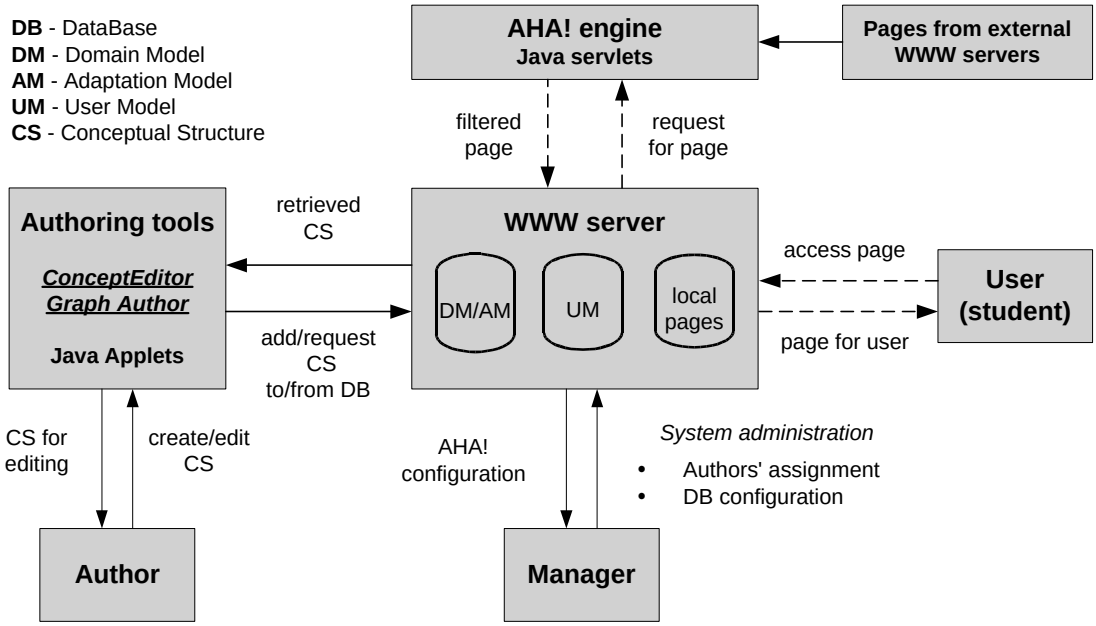

Figure 1. General AHA! Architecture.

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.

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". Besides this structure the author also has to create (write) the application content, usually consisting of a set of xhtml pages.

User (student) – the 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 is adapted on the basis of the information in the user model.
4. The adapted page 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 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 as a mySQL database. The manager chooses between file and database storage. AHA! can convert the files into the database and vice versa.

3 The Authoring Process

In this section we show how authors to build their adaptive presentations using AHA!. The authoring process consists of the following steps:

1. Defining Domain/Adaptation model.
2. Creating pages for an application.

To make this process of defining the domain/adaptation model easier we provide 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 result in 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. (The color scheme can be chosen by the user.) Fragments (within a page) can also have requirements to determine their desirability. The desirability of fragments within the accessed page is used to decide whether or not to include the fragment in the presentation of the page.

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 (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 these problems during the authoring process.

In AHA! 2.0 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.

Once the domain model is created and all the concepts and relations have been defined, the author has to associate pages to concepts. In the simplest case each page 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". 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).

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 "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.

We explain the Concept Editor using an example from (De Bra et al, 2002b) representing a few concepts in an imaginary course on WWW terminology. Assume that the page associated with the "javascript" concept becomes desirable when the user has knowledge about the "html tag" and "html special" concepts (see figure 2).

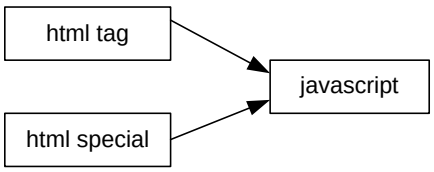

Figure 2. Prerequisite concept relationships in a course.

In figure 3 we show a screen dump of the "Concept Editor" for entering the conceptual structure of a course, when entering an adaptation rule for the "javascript" concept.

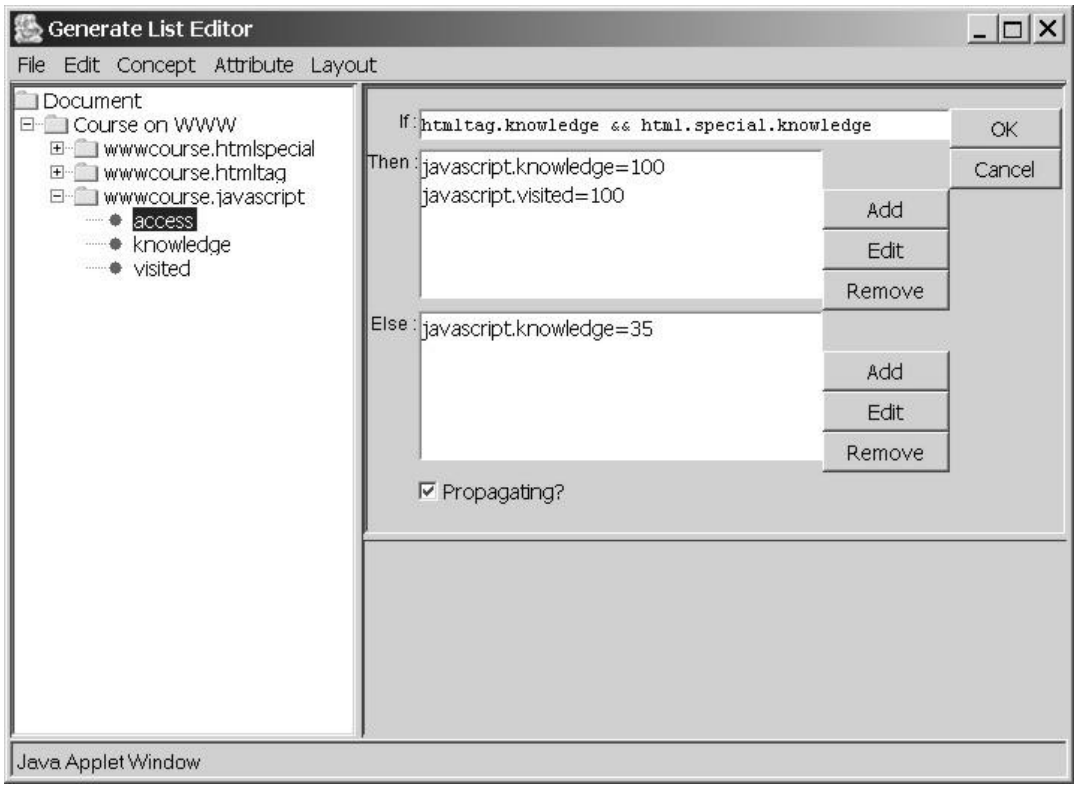

Figure 3. Entering conceptual structure with "Concept Editor".

The “javascript” concept has 3 attributes: “access”, “knowledge” and “visited”. For the “access” attribute we define a requirement (in the “if ” field) and actions which are associated with this attribute.

When a user accesses the page about “javascript” there are 2 possibilities:

1. If the page is desired (when the condition is true) the values of the “knowledge” and “visited” attributes are set to 100 (see the “Then” field). In the learning environment this means that the system thinks the user has full knowledge of this concept, and that the user has visited the page.
2. If the page is not desired the attribute is set to 35 (see the “Else” field). This value 35 was chosen arbitrarily. In a learning context it means 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.

For each action we have a “Propagating” attribute that indicates whether this action is allowed to trigger other actions or not. Propagation is used for instance to have knowledge of pages influence knowledge of higher-level concepts.

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 “javascript” page is read and fully known, but becomes undesirable later (when somehow the knowledge of htmltag or htmlspecial goes down) and the user revisits the “javascript” page, the knowledge of “javascript” 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 also 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 the concepts httprequest and httpreply both contribute knowledge to the concept http. The Graph Author automatically decides that both concepts contribute equally (50% each) to the knowledge of http. A knowledge propagation relationship can also have a value associated with it. Through these values we see that the knowledge contribution of htmltag, htmlspecial and javascript to the concept html is not equally distributed. 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.

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 (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 lead to these rules.

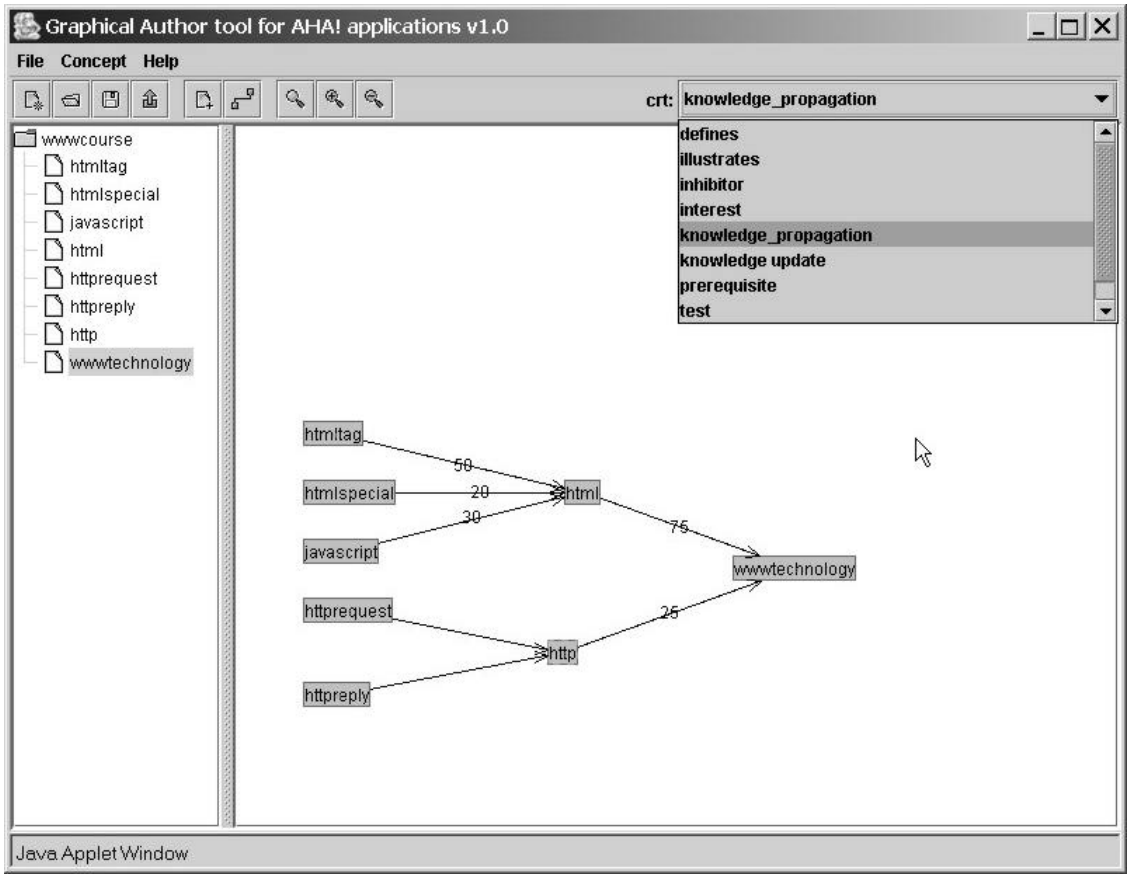

Figure 4. Entering conceptual structure with “Graph Author”.

3.2 Creating 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 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). AHA! 2.0 can still work with this old format, but now the local pages can also be written in XHTML, 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.

We have not provided any authoring tool for creating pages as the author can use any text or XML (XHTML) editor. He only has to know how to use special AHA!-related tags. The most important tags for adaptation are the “<if>” 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="htmltag.knowledge && htmlspecial.knowledge">
  <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>
```

With the “a” tag we use the “class” attribute to distinguish between different types of links. For example: <a href=”javascript.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. AHA!’s standard color scheme uses blue, black and purple, but the user can change this color scheme.

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

4 Presentation of Adaptive Applications

When a user logs in for the first time a user model is created. For each concept in the domain model the user profile 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 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 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 it is being processed, and a filtered page is shown in the browser – with fragments 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 link anchors AHA! uses three different colors to indicate the “desirability” of the links, as explained in the previous section. 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 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 used to guide the users, not to restrict them.

5 Conclusions

We continue to improve and to extend the functionality and usability of AHA! for both authors and end-users. 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! 2.0 at the same time. 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 future research issues 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 bythe 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*, Vol. 4, pp. 1-19, Kluwer academic publishers.

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

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.

Calvi, L., De Bra, P. (1997). "Proficiency-Adapted Information Browsing and Filtering in Hypermedia Educational Systems", *User Modeling and User-Adapted Interaction*, vol. 7, pp. 257-277, 1997.

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). 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 DiscoveringInteresting 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.