AHA! Adaptive Hypermedia for All

Paul De Bra*, Jan-Peter Ruiter
Department of Computing Science
Eindhoven University of Technology (TUE)
PO Box 513, Eindhoven, The Netherlands
debra@win.tue.nl

Abstract: There are roughly two types of (Web-based or other) Adaptive hypermedia systems (AHS): *special-purpose* systems, geared towards one specific application or application area, and *general-purpose* systems, designed with different applications in mind. Most existing systems are *special-purpose*, with a majority aiming at educational applications. Previous developments on the AHA system (De Bra & Calvi, 1998) and the AHAM model (De Bra et al, 1999) have shown that *general-purpose* AHS can be designed and implemented, but also that such systems tend to be too complicated for non-technical authors. This paper describes the "third generation AHA", called Adaptive Hypermedia for All, which is being developed as an open source project sponsored by the NLnet Foundation. AHA aims at bringing adaptivity to all kinds of Web-based applications, through a simple but powerful adaptive engine. In this paper we focus on the authoring interface for creating the conceptual structure of an adaptive application. But we also briefly describe how the AHA system can be used for some typical constructs found in (adaptive) Web-based applications.

Introduction and Background

In (Brusilovsky 1996) an overview is given of adaptive hypermedia systems (AHS) and of the methods and techniques these systems employ. In 1996 many AHS were not yet Web-based. Most were stand-alone (or locally networked) software systems aimed at a single *specific* application or application area, such as learning or information retrieval. Virtually all newer AHS are Web-based but still serve a single application area. This has important advantages: 1) A *special-purpose* AHS can be made easy to use for an author, through an authoring interface that is tailor made for the designated application, and 2) the interaction with the end-user can be optimized through a well-designed browser interface. A prime example of such a system is Interbook (Brusilovsky et al, 1998), a tool for creating adaptive textbooks. It offers users a frames-based presentation that includes a partial (and adaptive) table of contents, an overview of required prerequisite ("background") knowledge for the current textbook page, and an overview of "outcome" knowledge (concepts to which this page contributes). It offers some special features for learning as well, including a glossary and a "teach me" guided tour generator to learn about a specific concept. The special features of Interbook for educational applications may not be desirable in other application areas such as an adaptive search engine, on-line information systems (or information kiosks), on-line help systems, corporate websites, shopping sites (mail order catalogs), etc. The AHA project aims at creating a Web-based environment for creating very different types of adaptive applications.

Authoring usable hypermedia documents (Web-based or not) is always difficult. On the one hand an author wants to offer a lot of navigational freedom, with short paths to every page. But on the other hand an author wants to avoid overloading the user with too many links, which would make the selection of appropriate destinations difficult. Many adaptive hypermedia systems (AHS) tackle this problem by automatically selecting or emphasizing those links that are considered "appropriate", based on a model of the user's state of mind. Similarly, an author also wants to provide the "appropriate" information on each page, thereby including prerequisite explanations for users who need them, by providing additional information for the truly interested user, etc. See (Bruvilovsky 1996) for a long or (De Bra et al, 1999) for a short overview of AHS techniques. Such *personalization* is beginning to appear on websites, usually through a *My* suffix (like in "My Yahoo!", "My Excite", "myCNN.com", etc.). However, in most cases the personalization is based on a user profile that is created through an elaborate registration process, and not adapted to the changing interests and knowledge of the user that can be seen by observing the user's browsing behavior. The goal of the AHA project is to create a simple environment for making websites that adapt themselves to the user. It builds upon the adaptive engine in the AHA system (De Bra & Calvi, 1998, De Bra et al, 2000).

* Also at the "Centrum voor Wiskunde en Informatica" in Amsterdam, and the University of Antwerp, Belgium.

Architecture of the AHA system

The overall architecture of AHA is inspired by the AHAM reference model (De Bra et al, 1999), an extension of the Dexter model (Halasz & Schwartz, 1994). This model was constructed to capture the structures and functionality of most existing (and future) AHS. In this model an AHS consists of four parts that work closely together:

- The *domain model* describes the application domain in terms of *fragments*, *pages* and (abstract) *concepts*. In the AHA system pages are simple XML files containing fragments of HTML text (possibly with embedded images, JavaScript, etc.). XML tags are mainly used to delimit the boundaries of the fragments and to provide *conditions* on the inclusion of fragments. (We shall briefly describe these conditions later.) Pages are connected through hypertext **links**, using the HTML *anchor* tag (<A>). In AHA pages are at the bottom of a *concept hierarchy*. (In the full AHAM model the fragments are at the bottom.) In the next section we describe how the concept hierarchy is visualized in the authoring interface. Pages and higher-level concepts are also connected through **generate** and **requirement** relationships that play an important role in the processes of *updating the user model* and *performing the adaptation*. (See below.)
- The *user model* in AHA mainly consists of a table with for each page or concept an attribute value that is updated when the user browses through the Website. The updates are partially determined by the **generate** relationships. Currently for each concept AHA maintains an integer attribute with a value between 0 and 100. (Different attribute types are planned for the future, but for the most part having 101 possible different values for each concept is more than enough.) When a user accesses a page there are two possibilities:
    - If the page was *desired* (according to the requirement relationships we describe later) the attribute value for the page is increased to 100. In a learning context this means that the system thinks the user has full knowledge of this page.
    - If the page was *not desired* the attribute value is increased to 35 or left unchanged if it was already 35 or higher. This value 35 was chosen arbitrarily. In a learning context it represents that when a user reads a page for which the system thinks she is not (yet) ready, she only gains partial knowledge of that page.

  Updates to (the attributes of user-model) concepts are propagated to other concepts through the **generate** relationships. For each (page or) concept there is a list of relationships with other concepts. The relationships also have attributes associated with them that indicate how an update to the "source" concept's value defines the desired update to the "destination" concept's value. The update 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 value is propagated to the destination concept's value. In AHA all generate relationships for one concept are grouped into a structure like the following:

```
<genitem>
        <name>de_koninck</name>
        <item concept="belgian_beer" absolute="no" perc="10"/>
        <item concept="alcohol" absolute="yes" perc="5"/>
        <item concept="chocolate" absolute="no" perc="-3"/>
</genitem>
```

  A possible interpretation would be that the system's confidence that the user is interested in Belgian beers increases by 10 when the user visits the "de_koninck" page (assuming that in the user model the value for de_koninck goes from 0 to 100). The system's confidence that the user is interested in chocolate decreases by 3. The system also registers that the user is accessing a page on beer with 5% alcohol. This information can be used to conditionally include information in other pages, specific for beers with 5% alcohol content.

  The way updates are implemented in AHA allows the further propagation of *positive relative* updates, such as the one for "belgian_beer". This is useful to allow an interpretation of the user-model values as *knowledge*. Reading pages contributes to the knowledge of a section, which contributes to the knowledge of a chapter, etc. By only propagating positive relative updates any danger of infinite loops is avoided (De Bra et al, 2000).
- The *adaptation model* is a part in the AHAM model that is defined as a collection of rules that define how the adaptation must be performed. (Strictly speaking in AHAM the rules we described above for updating the user model through the generate relationships also belongs to the adaptation model.) In AHA the adaptation is defined through **requirement** relationships. For each fragment, page or concept we can define a relationship that indicates under which circumstances this fragment, page or concept is *desired*. Such a requirement relationship links the "source" fragment, page or concept to one or more "destination"

pages or concepts. Whether or not a fragment, page or concept is *desired* may not only depend on which information the user has read before, but also on which information the user has *not* read. (All user-model values are initialized to 0.) In the current AHA system for each (fragment, page of) concept there can be one requirement relationship, written as a Boolean expression in (user-model) pages or concepts. The requirement relationship for a fragment appears in the XML/HTML page that contains the fragment. Requirement relationships for pages and concepts are stored together in one XML file (typically named "xmlgenlist"). In the previous (non-existing) Website on Belgian products we can imagine a paragraph (fragment) describing special characteristics of beer with more than 10% of alcohol, and which is aimed at people who have shown great interest in the topic of beer. The Boolean expression for the requirement relationship for this paragraph could then be something like:

belgian_beer > 70 and alcohol > 10

As this example already shows that the simple user-model structure with just one (bounded) integer attribute per page or concept already enables the creation of adaptive applications based on knowledge (0 means *unknown* and 100 means *well known*), interest (by reading different pages on beer the user expresses her interest in the topic of beer) or anything else that can be expressed through a number (or an enumerated type simulated through numbers).

- The final element in the AHAM model is the *adaptive engine* that performs the actual adaptation. While the adaptation rules indicate the *desirability* of fragments, pages and concepts, the engine generates the pages in such a way that the user can distinguish desired from undesired information. Many techniques for adapting page content and link presentation exist (Brusilovsky, 1996). The AHA system currently uses the following techniques:
    - *Conditional inclusion of fragments*: desired fragments, i.e. fragments for which the *requirement* Boolean expression is true (or for which there is no condition) are included in the page; undesired fragments are omitted.
    - *Hiding or annotation of links*: AHA presents link anchors as colored text (not underlined). AHA uses three link colors: "good" for links to previously unread desired pages, "neutral" for links to desired pages that were already read, and "bad" for links to undesired pages. The default colors for "good", "neutral" and "bad" are blue, purple and black. Assuming that normal text is black this color scheme corresponds to the technique of *link hiding*: links to undesired pages are present (and fully functional) but not clearly visible. In AHA the user can change the color scheme. Another well-known color scheme, used e.g. in Interbook (Brusilovsky et al, 1996, Brusilovsky et al, 1998) uses the traffic light metaphor: green, yellow (or orange) and red. Such a scheme results in the technique of *link annotation*. (Interbook uses other annotations in addition to this.)

AHA supports "pure" Web-based applications. On the server side the adaptive engine consists of Java Servlets that are activated when the Web-server receives HTTP requests from the browser. Whenever the user clicks on a link the requested page will be "filtered" by the adaptive engine and the user-model will be updated by taking into account the page visit (and the associated generate relationships). Requests may also be generated through JavaScript code that is embedded in the HTML pages, e.g. for updating Web-pages that consist of several frames. For the AHA engine all these requests are treated equally and handled separately. It is thus possible to create frames-based as well as non-frames-based applications with AHA. AHA is also a "pure" Web-based system in the sense that it *only* reacts to HTTP requests. There are no special tools or protocols to interact with the adaptive engine, no user-model updates based on timeouts, etc. AHA can be installed on any Web-server supporting Java Servlets. (Currently we have a "production" version running on W3C's Jigsaw server and on Sun's JSWDK and are experimenting with the emerging Servlet support on the popular Apache server.)

The architecture, and even the current implementation of AHA is sufficiently powerful to support a wide variety of adaptive Web-based applications. We show parts of a few possible examples later. The biggest shortcoming until now has been the lack of authoring support. In the next section we discuss the automatic generation (and maintenance) of the XML file(s) that represent the conceptual structure of an application, from a user-friendly and (partially) graphical authoring interface.

The authoring interface

In order to develop any non-trivial Web-based application one needs to first design (or "draw") the overall conceptual structure. Traditionally, in hypertext applications, a "conceptual" diagram shows how pages are connected through links. The link structure normally contains a *hierarchy*, which corresponds to a book structure (with chapters, sections, subsections and paragraphs), some special structures such as table of contents, index and

glossary, and a set of carefully chosen cross-reference links between pages. See e.g. (Botafogo et al, 1992) for an article on metrics for link structures. Many Websites even offer the hierarchical part of the link structure as a navigation aid for end-users, through a so-called *site map*.

For an adaptive application (at least when developed for AHA) one needs to consider not only the **link** structure but also the **generate**- and the **requirement** structures. In this paper we concentrate on the creation of the **generate** relationships. (The creation of link structures is well known, and the requirement structures can be pretty arbitrary. We plan to study authoring for special types of requirement relationships such as *prerequisites* later.)

In most cases the application domain will have some kind of hierarchical structure. Figure 1 shows a small part of such a structure, from our imaginary Website on Belgian products. Such a hierarchy normally corresponds to generate relationships that have a positive, relative value. Reading about "de_koninck" shows interest in beer and also means you learn something about beer. Learning something about beer means you also learn something about Belgian products. And learning about Belgian products means you're learning something about Belgium. To keep the overview simple and readable the graph does not show the *percentage* of each generate relationship.

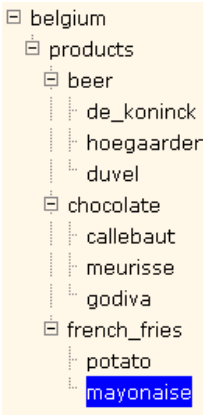

Figure 1: concept hierarchy shown through generate relationships

When building an application the other, more arbitrary generate relationships must also be entered, and can be shown in the same graphical way. Figure 2 shows the generate relationships between different beers and the concept "alcohol". To indicate that the concepts "de_koninck", "hoegaarden" and "duvel" have a hierarchical and therefore prime place somewhere else in the graph their names are preceded by an icon like that for a windows "shortcut".

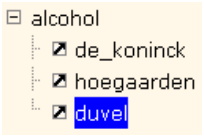

Figure 2: structure of positive and negative generate relationships

Figure 1 can be extended with negative relationships from the different beers to the concept "chocolate" and negative relationships from the chocolates to the concept "beer". Reading about "de_koninck" not only increases the interest (and/or knowledge) in beer but also decreases the interest in chocolate. If both updates would propagate then "products" would receive a positive update through beer and a negative update through chocolate. This undesirable situation explains why the AHA system does not propagate negative updates (De Bra et al, 2000).

The graphical overview in the authoring interface shows a *top-down* view on the generate relationships. Using the book metaphor the top-level element will be the book, with the chapters at the next level, then the sections, etc. This metaphor is so common (especially since everyone knows the Windows Explorer) that we expect authors to easily understand these overviews. For defining the generate relationships however we take a *bottom-up* approach. When an author enters a concept (or page) into the structure he must specify the influence of this concept on other concepts. Using the book metaphor again, when entering a subsection the author identifies to which section it belongs. When entering a section the author says to which chapter this section belongs, etc. Therefore we have created an authoring interface for entering the generate relationships for which the given concept is the source. Figure 3 contains (part of) the interface that shows the generate relationships for "de_koninck" in our imaginary example. Each time the author adds or updates generate relationships the graphical (explorer-like) presentation of the overall structure is updated accordingly.

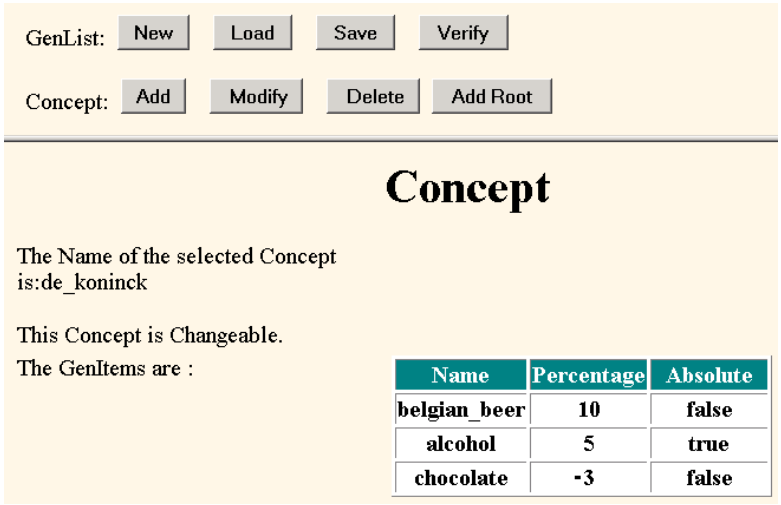

Figure 3: interface for entering and updating generate relationships

The first entry in the table in Figure 3 is interpreted as the hierarchical position of the concept. So it is the order of the items that determines where "de_koninck" is shown with and where without the "shortcut" icon. And in case there is a cycle in the generate relationships the order in which the concepts were created determines which concept gets the highest position in the hierarchy.

Figure 3 also shows another feature of AHA: for each concept the author can indicate whether or not it is "changeable". The user of an AHA application can manually alter the user-model by means of a form. For each concept the author marked as "changeable" the user can view and alter the value in her user-model. We realize that adaptive Web-based applications can only work with unreliable input (the page requests made by the user), and therefore it is best to allow end-users to manually update (or "correct") their user-model.

Examples of AHA applications

In this section we briefly indicate how to realize some common constructs using the AHA system. We first look at the most popular application area for AHS: learning by reading, and possibly verifying knowledge through multiple-choice quizzes. The author must decide for each page how much knowledge it contributes to each course topic. This determines the *percentage* to use in the generate relationship that links pages to topics. In order to allow users to skip parts of the material the author can choose the numbers such that reading most (but not all) pages about a topic already results in 100% knowledge. Topics can be grouped into larger units, and knowledge of topics is accumulated in the larger units because AHA propagates positive relative updates.

The AHA system contains a module for multiple-choice tests. That module allows for randomized tests that make cheating somewhat difficult and that are thus suitable for student evaluation. Simple fixed multiple-choice quizzes can be created in plain HTML by using links to pages that explain whether an answer is right or wrong. When the user chooses an incorrect answer, the answer-page can explicitly decrease the knowledge value for some topics and reset the value of pages that must be reread to 0. When the user reads the material again knowledge about the pages and topics is augmented again.

Another popular construct is that of a "table of content" frame, showing a menu and possibly also one or more submenus. Basically, this *navigation frame* acts like the left frame in the Windows Explorer. Submenus can be opened in the navigation frame, or when a page that belongs to a different submenu is opened by clicking on a link (anchor) in the *information frame*. The (sub)menus in the navigation frame can be implemented as conditional fragments. Each menu is assigned a unique number, and a global "concept" *menu* must be defined. Each page is connected to the appropriate menu by adding a generate relationship between the page and the *menu* concept. This generate relationship must define an *absolute* update to the number of the (sub)menu that must be opened when this page is displayed. Whenever the page is loaded in the information frame the *menu* concept is set to the correct value. A one line JavaScript program in the HTML page can instruct the browser to reload the *navigation frame*. Because the request for the page precedes that for the menu the menu is retrieved with the appropriate submenu opened up. Such Explorer-like menus are used in our on-line course text about graphical user-interfaces, at

http://wwwis.win.tue.nl/2R350/. (The whole system is so fast that you hardly notice that the menu is refreshed *after* loading a page and not *simultaneous* with loading the page.)

A final example is that of verbosity levels. In the same course on graphical user-interface, there is a one-to-one correspondence between the Webpages that are available for the students and the viewgraphs that are used during the lectures. The same topics are treated and the overall style of the presentation is the same. But for each item the on-line text is much longer than the one-liners on the viewgraphs. The pages and viewgraphs are always in sync during updates to the course because they are actually the same pages. A global "concept" *verbose* is used to guide the conditional inclusion of fragments. In the course only the levels 0 and 100 are used, but by using more values one can create a Web presentation that exists at different levels of verbosity.

Conclusions and Future Work

Web-based adaptive systems need not be geared towards a single application or application area. AHA (the Adaptive Hypermedia Architecture) was originally built to make an on-line (hypermedia) course adaptive. For this purpose a user-model was introduced with for each page (or concept) a knowledge value. We soon realized that there was nothing in the way AHA treated this knowledge value that actually required the value to represent knowledge. We have developed applications (and shown examples in this paper) in which the AHA engine performs arbitrary types of adaptation based on the user's navigation (De Bra & Calvi, 1998). We have presented (most of) the interface for creating and manipulating the *generate relationships* that define how user actions result in user-model updates (and thus indirectly also in adaptation that is based on the user-model). We continue to improve and extend the functionality and usability of AHA, especially for authors, through a research grant of the NLnet Foundation (see www.nlnet.nl). In this AHA project (or Adaptive Hypermedia for All) we are developing open source software that will be made available from our Website http://wwwis.win.tue.nl/. (Preliminary software is already available at this time and we welcome feedback.) Among the planned developments are analysis tools for detecting potential problems with the *link*, *generate* and *requirement relationships*. These tools will warn an author about pages that never become *desired*, or that can never even be reached, fragments that are never included, etc. Apart from improving the usability for authors 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.

References

Botafogo, R.A., Rivlin, E., Shneiderman, B. *Structural Analysis of Hypertexts: Identifying Hierarchies and Useful Metrics.* ACM Transactions on Information Systems, 10:2, pp. 142-180.

Brusilovsky, P. (1996). *Methods and Techniques of Adaptive Hypermedia*. User Modeling and User-Adapted Interaction, 6, pp. 87-129. (Reprinted in Adaptive Hypertext and Hypermedia, Kluwer Academic Publishers, 1998, pp. 1-43.)

Brusilovsky, P., Schwarz, E. & Weber, T. (1996). *A tool for developing adaptive electronic textbooks on WWW*. Proceedings of the AACE WebNet'96 Conference, pp. 64-69.

Brusilovsky, P., Eklund, J., Schwarz, E. (1998). *Web-based Education for All: A Tool for Developing Adaptive Courseware*. Computer Networks and ISDN Systems (Seventh International World Wide Web Conference), 30, 1-7, 291-300.

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

De Bra, P. & Calvi, L. (1998). *AHA: a Generic Adaptive Hypermedia System*. 2nd Workshop on Adaptive Hypertext and Hypermedia, pp. 1-10. (URL: http://wwwis.win.tue.nl/ah98/DeBra.html)

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., Houben, G.J. & Wu, H. (1999). *AHAM, A Dexter-based Reference Model for Adaptive Hypermedia*, Proceedings of the ACM Conference on Hypertext and Hypermedia, pp. 147-156.

Halasz, F., Schwartz, M. (1994). *The Dexter Hypertext Reference Model*. Communications of the ACM, 37:2, pp. 30-39.