Making General-Purpose Adaptive Hypermedia Work

Paul De Bra*+, Ad Aerts, Geert-Jan Houben+, Hongjing Wu
Department of Computing Science
Eindhoven University of Technology (TUE)
PO Box 513, Eindhoven
The Netherlands
{debra,wsinatma,houben,hongjing}@win.tue.nl

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

Abstract: Adaptive hypermedia systems (AHS) have typically been geared towards one specific application or application area, in most cases related to education. The AHA (Adaptive Hypermedia Architecture) system is a Web-based adaptive hypermedia system specifically intended to serve many different purposes. As such AHA must be able to perform adaptation that is based on the user's browsing actions, regardless of the interpretation of browsing as learning. A general-purpose adaptive hypermedia system must be able to handle cycles in adaptation rules and non monotonic user model updates. (An educational system in which a user’s knowledge about a hierarchical concept structure can only increase is much simpler.) This paper describes how AHA handles these aspects and indicates how other adaptive hypermedia systems may be turned into more general-purpose tools as well.

Keywords: Web-based adaptive hypermedia, user modeling, adaptation rules.

1. Introduction and Background

Authoring usable hypermedia documents is difficult. On the one hand an author wants to offer a lot of navigational freedom, with short paths to every possible destination. 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 applications [B96] 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 which the system maintains.

Educational applications are typical in this respect: the system monitors the user’s progress through an electronic textbook and gradually starts to offer more and more complex and detailed information.

Many adaptive hypermedia systems have a simple user model structure. Some features of the user are stored, often called preferences (which are set by the user and not altered by the system) and knowledge about concepts is maintained and updated while the user is browsing. (Actually the knowledge is estimated by the system, based on a record of accesses to information pages and on the results of some multiple-choice tests). Relationships between pages or concepts are also simple: many systems only deal with prerequisite relationships, meaning that one (prerequisite) concept should be studied before another one. These types of systems are well suited for educational applications that share the following properties:

1. Once the conditions (prerequisites) for a concept are fulfilled they remain fulfilled. (When the user is ready to study a concept, she will always remain ready to study that concept.)
2. A user’s knowledge about a concept can only increase. (The user never forgets what she has learnt.)

Educational adaptive hypermedia systems are based on the notion that the user model consists of
concepts with an associated knowledge value. Each time a page (about a concept) is accessed some server-side program (e.g. a Java Servlet) registers this access and updates the knowledge value for the associated concept. In order to turn such a system into a general-purpose one the notion of knowledge about a concept must be generalized to value for a user model attribute. As such a concept need not represent domain knowledge, and the attribute may represent any kind of property, not necessarily knowledge. As such, the applications an adaptive system must deal with may not satisfy the two properties above.

In [DC98] we presented a first version of the AHA system, which allows applications to "violate" the first property: the condition for including or providing access to a piece of information may include negations. Thus, still in educational terminology, learning about a concept may make another concept superfluous and therefore hidden or inaccessible. We call such relationship an inhibitor. (It is the opposite of a prerequisite.) In the first version of AHA applications must satisfy the second property, except that the user can manually alter her user model (through a form). This paper presents the next version of AHA, which supports applications that do not satisfy the second property. In addition, knowledge is no longer represented by a Boolean value (known or not known) but by an integer value between 0 and 100. We give some interesting examples of "general purpose" use of adaptation rules that show adaptive behavior that is very different from that of applications for learning.

The design and development of the general purpose AHA system posed some interesting challenges. In order to accommodate a rich model of an application domain we allow the definition of concept relationships. Changes to the (user model) value for a concept may induce changes to (the value of) other concepts. This has its origin in education as well, where concepts may form a hierarchy. The user may be required to study several small concepts (possibly pages) in order to acquire the knowledge about a larger concept. Updates to the (knowledge) value for the small concepts contribute towards the (knowledge) value for the larger concept. In AHA values can be updated arbitrarily (increased and decreased) and the structure of concept relationships is not required to satisfy constraints such as being acyclic. In general, allowing recursive updates to the user model may:

1. generate unpredictable or even undesired results;
2. cause infinite loops of updates.

This paper describes how AHA deals with recursive adaptation rules in order to guarantee that the outcome of each user model update is always predictable and desirable. We illustrate this with some typical examples of update rules.

This paper is structured as follows. Section 2 briefly describes the architecture of AHA. Section 3 illustrates why AHA has a recursive update algorithm. Section 4 describes that user model update algorithm, and shows the deterministic (and finite) behavior of that algorithm. Section 5 gives a few non-educational examples expressed in AHA. Section 6 concludes with remarks on possible future extensions of AHA.

2. Overview of AHA

The general structure of AHA is somewhat comparable to many other adaptive hypermedia systems, as described in [B96]. In [DHV99] we defined a reference architecture for adaptive hypermedia systems, called AHAM. According to AHAM an adaptive hypermedia application consists of three parts:

- A domain model describes the information content of the application, both at the conceptual level and at the level of pages and information fragments. It also contains concept
relationships, for instance to indicate that some concepts are prerequisites for other concepts.

- A user model describes how the system models the user’s state of mind. In most cases (including AHA) an overlay model is used, meaning that for each concept of the domain model one or more attribute values are maintained that represent how the user "relates" to this concept.
- An adaptation model (called teaching model in [DHW99]) contains rules that describe how a user’s actions (page accesses) result in user model updates and how the user model determines how a requested page will be presented.

In the (new) AHA system these three parts have the following structure:

- The domain model consists of fragments, pages and (abstract) concepts. Each page is an XML file with (almost opaque) fragments that are conditionally included, and with hypertext links. In our current applications the "opaque" content is actually HTML, but AHA simply ignores this content except for the links. In the sequel we use the term page concept to indicate a concept represented by a page. When we just say abstract concept we mean a concept that is not a page. The term concept is used to indicate a concept that can be a page concept or abstract concept. AHA distinguishes three types of concept relationships:
  - link relationships: AHA recognizes HTML anchor (A) tags that represent hypertext links between page concepts. (AHA only uses links between pages. The general AHAM model also considers links between concepts in general.)
  - generate relationships: In AHA the access to a page may (recursively) cause an update to several elements in the user model. The generate relationships specify these updates. They are stored (together) in a special XML file. For each concept (page concept or abstract concept) there is at most one generate relationship (between that concept and possibly several other concepts). The generate relationships are described in detail in the adaptation model below.
  - requirement relationships: In AHA fragments, pages and concepts may be considered desirable or undesirable. The requirement relationships specify the conditions for this "desirability". The requirement relationships for fragments are stored inside the pages; the requirements for pages and abstract concepts are stored in a special XML file. The requirement relationships are described in detail in the adaptation model below.

- The user model consists of some user identification and preferences, and then for each (page or abstract) concept in the domain model a value. These values are called knowledge for historical reasons, but that does not imply that they must actually represent knowledge. This information is stored in a special XML file in the following format:

```xml
<knowledgeitem>
  <knowledgename>concept</knowledgename>
  <knowledgeperc>80</knowledgeperc>
</knowledgeitem>
```

This entry means that the value for "concept" is 80. All values in AHA are integers between 0 and 100, just like in Da Silva’s educational system [PDS98]. When a user first registers for an AHA application the user model is generated automatically with all knowledge values set to 0. When an update to a value is specified in a way that would produce a negative value, the result is 0, and when the update would produce a value over 100 the result is 100. Apart from these preferences and values AHA also keeps a logfile (per user) with all page accesses.

Note that fragments are not represented in the user model.

- The adaptation model consists of author-defined generate rules (corresponding to generate relationships) and requirement rules (corresponding to requirement relationships), and some system-defined adaptation rules that define the adaptive behavior of AHA. The basic
adaptation rule of AHA deals with the "desirability" of fragments, pages and abstract concepts. The requirement rules define this desirability. A requirement for a page or concept looks like:

```xml
<concept>
  <conceptname>
    theconcept
  </conceptname>
  <relationexpression>
    requirement1 > 30 and requirement2 < 80
  </relationexpression>
</concept>
```

This entry (in the XML file containing the requirements) means that the desirability of "theconcept" depends on the value of concept "requirement1" being higher than 30 and the value of concept "requirement2" being lower than 80.

For a fragment the following piece of XML is included in a page:

```xml
<if expr="requirement1 > 30 and requirement2 < 80">
  <block>
    here is the conditionally included fragment
  </block>
</if>
```

The "visual" result in AHA of requirements being satisfied (or not) is as follows:

- AHA has three link colors, called GOOD, NEUTRAL and BAD. (They are user-defined, and blue, purple and black by default.)
  - When a page is desirable (i.e. its requirement is fulfilled) and has not been accessed by this user then links to this page are shown in the GOOD color. Links to a previously accessed desirable page are shown in the NEUTRAL color.
  - Links to an undesirable page are shown in the BAD color.
- When a fragment (of the page that is being accessed) is desirable (i.e. its requirement is fulfilled) the fragment is included in the page.

These link colors are enforced through the use of link classes combined with a style sheet. It may not appear that way to a Web-savvy user because links change from blue to purple when visited. But it is AHA that enforces this change and not the browser that generates it.

When a page is accessed its value in the user model is updated. Initially the value is 0. When a desirable page is accessed its value is increased to 100. When an undesirable page is accessed its value is increased to 35 or left at its previous value if that was already 35 or higher. Changes to the (user model) value of a page may induce updates to the value of other concepts, based on the generate rules. The generate rules are stored in an XML file. Entries look like:

```xml
<genitem>
  <name>concept1</name>
  <genlist>concept2:+40 concept3:-30 concept4:50</genlist>
</genitem>
```

This relationship means that when the value of concept1 is augmented by X (for instance 35, 65 or 100), the value of concept2 is augmented by 40% of X, the value of concept3 is decremented by 30% of X and the value of concept4 is set to 50, independent of X. (The resulting values are "clipped" so they remain between 0 and 100.)

Typically the generate relationships (and rules) are used to model a hierarchical (containment) structure of concepts and subconcepts. When a section of a textbook contains 10 topics (or pages), each page may generate 10% of the knowledge for that section, so that
the value of that section concept becomes 100 after reading all 10 pages. However, many other uses for generate clauses exist. The example above already shows that a page access may cause the value for a concept to decrease as well as increase. This clearly does not fit the model of knowledge in a hierarchy of concepts. We will give more examples later.

The above constructs to change the appearance of links, and the conditional inclusion of fragments represent adaptive techniques that are traditionally called [B96]:

- **Adaptive presentation:** in AHA the presentation of pages is influenced by conditionally including fragments of content. This is a low level technique. Brusilovsky [B96] distinguishes changes to the text from selecting different presentation media. But since AHA targets a Web environment all media objects are included from HTML pages, so media selection can be done through conditional inclusion of content fragments that contain the HTML tags to include objects of the appropriate media type.

- **Adaptive navigation support:** in AHA the user is aided in making a selection among the links that are available on every page. This is done by presenting desirable links in a different color than undesirable links. The user can choose which colors AHA will use. The default color scheme is blue (for GOOD links), purple (for NEUTRAL links) and black (for BAD links). This implements what Brusilovsky [B96] calls link hiding, because the undesirable links are presented in the same color as normal text and are thus effectively hidden. (Links in AHA are typically not underlined.) When the user makes the undesirable links visible by altering the colors she gets what Brusilovsky [B96] calls link annotation.

Because general-purpose authoring tools for XML data files are currently lacking AHA comes with some scripts to convert HTML pages to the required XML syntax and with a forms-based interface for creating the list of generate rules and requirement rules.

### 3. The Need for Recursive Updates

All adaptive features depend heavily on the updates to the user model being performed correctly. In the first version of AHA every access to a page could only change the Boolean value for some concepts. As a result it was not possible to express that 10 pages contributed towards a "composite" concept. As a result the definition of some requirements like prerequisites difficult. If some information in a document depended on the a chapter of 10 pages having been read completely, the first version of AHA required clauses like:

```xml
<relationexpression>
  page1 and page2 and page3 and page4 and page5 and page6 and page7 and page8 and page9 and page10
</relationexpression>
```

(pageX is allowed shorthand for pageX=100)

In the new AHA one can define each page to generate 10% of "chapter" (apart from generating 100% of itself, which every page concept does). A page for which the entire chapter is prerequisite knowledge then has a requirement:

```xml
<relationexpression>
  chapter=100
</relationexpression>
```

Apart from this, it becomes possible to require at least 8 out of the 10 pages to have been read by specifying:

```xml
<relationexpression>
  chapter>=80
</relationexpression>
```
Pages and concepts can also form a deeper hierarchy, like when pages contribute knowledge to a section and sections to a chapter and chapters to a whole book. There are two ways to implement such a hierarchy: recursively or non-recursively. In a non-recursive definition there will be *generate rules* like

```xml
<genitem>
  <name>somepage</name>
  <genlist>sectionXY:+20 chapterX:+5 book:+1</genlist>
</genitem>
```

The use of integer values may make it impossible to accurately indicate the small contribution a single page makes towards the knowledge of a chapter or whole book. Another problem is that for each page its contribution to the section, chapter and book all have to be indicated. It is easier to only indicate how a page contributes to its section, then indicate (once) how much that section contributes to its chapter, and indicate (once) how much that chapter contributes to the whole book:

```xml
<genitem>
  <name>somepage</name>
  <genlist>sectionXY:+20</genlist>
</genitem>
<genitem>
  <name>sectionXYZ</name>
  <genlist>chapterX: +25</genlist>
</genitem>
<genitem>
  <name>chapterXY</name>
  <genlist>book:+20</genlist>
</genitem>
```

Now, in order for page accesses to contribute towards the (knowledge) value of the whole book, user model updates must be performed recursively by AHA. This is easy to implement, and safe for concept hierarchies (acyclic structures with only positive contributions to knowledge values). However, when designing the user model update algorithm so that it works with arbitrary *generate rules* the following issues have to be dealt with:

1. There may be loops in the recursion.
   Example: (An update to) A generates +100% of (that update to) B and B generates -100% of A. When page A is read (for the first time and A is desired), the value of A becomes 100 and the value of B becomes 100. But because B is updated, it generates an update to A: A is decreased by 100. This update to A generates an update to B, etc.

2. It may be difficult to decide how to handle repeat visits to pages.
   Example: A generates +50% of B. When page A is visited for the first time the value for A becomes 100 and that of B becomes 50. We do not want the value of B to become 100 when A is revisited. On the other hand, if some other request decreases the value of A (and maybe recursively also of B) then a repeat visit to A should update B again.
   Another example is when one set of pages is typically accessed by visitors of some type (say students) and another set of pages is typically accessed by other visitors (say faculty). If repeat visits are registered one can let them augment a "concept" that identifies the visitor type. We will see below how this can be done in AHA.

3. One page access may induce two updates to a concept. If A generates +50% of B and +50% of C, B generates +60% of D and C generates +40% of D, then a (desirable) first access of A generates an update of +30 to D through B and another update of +20 to D through C. Both updates must be applied (correctly) in order to get a predictable (deterministic) result.
4 The Update Algorithm in AHA

The basic user model update algorithm is a simple recursive application of generate rules. In AHA a (single) generate rule is associated with each concept. So if a concept value is updated (and actually changed by the update), AHA will simply look at the generate rule for that concept to determine which other concept values to update. For each of these concepts the update continues recursively. As shown above this basic algorithm can easily be tricked into producing non-deterministic results or infinite loops. There are three easy ways to avoid loops, without giving up on having a recursive algorithm:

1. Disallow the definition of sets of generate rules with loops. Loop detection is (conceptually) easy and can be done while the author is entering generate rules.
2. Allow each concept to be updated only once (per run of the algorithm).
3. Allow each generate rule to be used only once (per run).

(Actually, each of these rules can be modified to allow some other number of iterations instead of 1.)

Each of these measures has its merits and drawbacks. We want to point these out to illustrate that coming up with an update algorithm that is general purpose is non-trivial.

1. Circular definitions in generate rules are useful for applications like guided tours, where each step in the tour makes the next page in the tour desirable and all other pages in the tour undesirable. One may wish to have the end of the tour lead back to the beginning, thus creating a loop. Depending on how this loop is implemented (using generate rules) it might result in an infinite loop in a simple straightforward update algorithm.
2. When we allow only one update to a concept (in each run) then the third example of Section 3 would fail: The update of D through B and C would no longer be possible. If only one of these updates is selected the resulting update to D depends on the choice and is thus unpredictable. (We are not arguing here that incrementing D by 30+20=50 is "the" only and correct solution here. We only argue that non-deterministic behavior is not desirable.)
3. We can extend the third example of Section 3 so that if we allow each generate rule to be used at most once (or any other fixed number of times per run) it will fail. Assume that A generates +50% of B and +50% of C, B generates +60% of D and C generates +40% of D and D generates 50% of E. We already explained that we want D to be updated by B and C. So D is updated with +30 and +20, but this should result in E being updated with +15 and +10. This requires the generate rule from D to E to be executed twice.

The above examples show that these restrictions are all undesirable in some cases. Therefore, in AHA we have chosen a somewhat less intuitive but certain way to avoid infinite loops while ensuring deterministic behavior. We distinguish four cases:

- **monotonic update**: the update to the value of a concept induces an update to another concept, with a "+" in the genlist field. The update algorithm then continues recursively for that concept. A monotonic clause in a generate rule is only allowed to update (the value of) an abstract concept, not a page concept.
- **non-monotonic update**: the update to the value of a concept induces an update to another concept, with a "-" in the genlist field. The update algorithm performs the update but does not continue recursively for that concept. Again, non-monotonic updates are only allowed to update abstract concepts.
- **absolute update**: the update to the value of a concept induces an update to another concept, with a fixed value (no "+" or "-" in the genlist field). The algorithm performs the update...
but does not continue recursively for that concept. Only page concepts are allowed to have a
generate rule with absolute updates.

- **self update**: the update to the value of a concept induces an update to the concept itself, by
  including the concept in its own "genlist". This does not recursively cause the rule for that
  concept to be executed again. A typical use of a self update is to make repeat page visits
  have effect on other concepts. Only page concepts are allowed to have a generate rule with a
  self update.

Consider the following example:

```xml
<genitem>
  <name>concept1</name>
  <genlist>concept1:0 concept2:+40 concept3:-30 concept4:50</genlist>
</genitem>
```

(From the above restrictions we already see that concept1 must be a page concept, because it
generates an absolute update.)

If the value of concept1 is augmented by X, the value of concept2 is augmented by 40% of X and
the generate rule for concept2 will be (recursively) executed. The value of concept3 is
decremented by 30% of X, the value of concept4 is set to 50, but the generate rules for concept3
and concept4 are not executed, and finally the value of concept1 is reset to 0.

Note that in this example X may be a positive or a negative value, but AHA currently has no
events that cause the update algorithm to start with a negative X. (This feature may be added in the
future.)

By only (recursively) propagating monotonic updates, and by limiting (clipping) all updates to the
range 0..100, the update algorithm is guaranteed to terminate. In worst case all values are initially
zero and in each recursive step one abstract concept is augmented by 1. In this case the algorithm
can run for 100 times the number of abstract concepts steps and then it stops because all abstract
concepts’ values are 100. So the algorithm always terminates in O(number of abstract concepts)
time. Usually only a few recursive steps are necessary, so the time needed to perform user model
updates is negligible compared to that for performing the adaptation on the page content. (AHA
uses Java servlets; each page access requires very little computation time, typically less than 0.1
sec on a Pentium class system.)

The restrictions on the four types of updates are necessary to make the update algorithm
deterministic. If A updates B and C and B and C would each perform an absolute update to D,
with different values, then the result of the update algorithm (on D) depends on the order in which
the two absolute updates are performed. However, if the updates to B and C cause recursive steps
then they must be monotonic and the restriction on monotonic updates then says that B and C must
be abstract concepts. The restriction on absolute updates says that B and C are not allowed to
generate an absolute update on D because only page concepts are allowed to generate absolute
updates.

### 5. Examples

A first general purpose example is that of a menu structure the ones that appear in our Web-based
course on Graphical User-Interfaces and in an on-line kiosk system with information for interns.
Each menu (or submenu) item corresponds to a page of the course. The menu is shown in a
separate frame, like:

- first menu item
  - first submenu conditionally appears
To implement a menu structure where clicking on a menu item opens up the submenu (and closes the submenus for other menu items) a set of *absolute updates* can be used. It is not actually the click that triggers AHA to update the user model, but the access to the page associated with the menu item. (The page is displayed in a separate frame and contains a (one-line) Java function to update the menu frame. Restarting the menu is necessary to allow the server to generate the updated menu.)

```xml
<genitem>
   <name>firstmenu</name>
   <genlist>firstmenu:100 secondmenu:0 thirdmenu:0 ...</genlist>
</genitem>
```

(The "firstmenu:100" clause is included for clarity but in fact unnecessary. In AHA "firstmenu:+100" is implied by the access to "firstmenu" and that is enough to guarantee that the value of "firstmenu" becomes 100.)

Each submenu is then conditionally included as follows:

```xml
<if expr="firstmenu=100">
   <block>
      <ul>
         <li> first submenu conditionally appears 
         <li> ... 
      </ul>
   </block>
</if>
```

A second general purpose example is that of grouping users. If certain pages are typically accessed by students, and other pages by faculty, then one can automatically detect whether the user is a student or faculty member. The Eindhoven University of Technology is currently building an adaptive central university website (with general information). Distinguishing groups of users automatically is one of the intended adaptive features. For a student page one writes:

```xml
<genitem>
   <name>studentpage1</name>
   <genlist>studentpage1:0 conceptstudentpage1:+100 student:+2 faculty:-1</genlist>
</genitem>
```

Each access to "studentpage1" resets the "studentpage1" value so that the update is repeated for each visit to that page. By updating "conceptstudentpage1" a copy of the "studentpage1" concept is obtained that can be used to represent the knowledge of "conceptstudentpage1". When "studentpage1" is accessed for the first time (and is desirable) the value of "studentpage1" first goes from 0 to 100. This causes "conceptstudentpage1" to go from 0 to 100 as well. The *self update* to "studentpage1" resets the "studentpage1" value to 0. So the small updates to "student" and "faculty" are repeated each time "studentpage1" is accessed, but the value for "conceptstudentpage1" remains at 100 and is thus not updated repeatedly. The update behavior for "conceptstudentpage1" is thus identical to what the update behavior for "studentpage1" would have been if there was no *self update* to "studentpage1". By augmenting "student" and decrementing "faculty" each time a student-related page is accessed one indicates that the confidence that the user is a student and not a faculty member increases. Other pages or fragments can be made dependent on expressions like "student > faculty".
6. Conclusions and Future Work

Web-based adaptive systems need not be geared towards a single application or application area. AHA was originally built to make an on-line (hypermedia) course adaptive. After its redesign it is now being used in different applications, and its adaptation features are used to recognize user groups, to support explorer-type menus, and other aspects that are not necessarily related to the user's knowledge. The examples in Section 3 show how "concepts" and "knowledge values for concepts" can be used (some would say abused) to perform different types of adaptation.

The possible values that can be associated with a concept (in AHA) are limited to a single integer attribute with values between 0 and 100. While this enables us to define a user model update algorithm that always terminates with a deterministic result it is still limited. In the AHAM model for adaptive hypermedia systems, an arbitrary number of attributes (with arbitrary value domains) can be associated with each concept. The recent additions to AHA allow many (educational and non-educational) applications to be implemented using AHA, but in order to become truly versatile AHA needs to be augmented with facilities to define a user model with arbitrary attributes and value domains for each concept. When this will be done the method in which AHA avoids infinite loops and ensures deterministic behavior of the application of adaptation rules can no longer easily be generalized to such rich user models. Further research is needed to determine how to handle (recursive) user model updates for complex user models.

7. References

[B96]

[DC98]

[BSW96]

[DC98]

[DHW99]

[PDS98]