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

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.

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 systems (AHS) (Brusilovsky 1996) 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.

AHS store some features of the user, often called *preferences* (which are set by the user and not altered by the system) and *knowledge about concepts*, in a *user model*. That model is maintained and updated while the user is browsing. Relationships between pages or *concepts* are 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. A possible exception is when a student performs badly on a test.)

Educational AHS 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. 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. As such, the applications an AHS must deal with may not satisfy the two properties above.

In (De Bra & Calvi 1998) 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*. In the first version of AHA applications must still satisfy the second property. 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.

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

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 one concept may induce changes to (the value of) other concepts. Updates to the (knowledge) value for (small) concepts can contribute towards the (knowledge) value for another "higher level" concept. In AHA values can be updated arbitrarily (increased and decreased). Decreasing values is probably most useful in applications where the values have a meaning other than "knowledge". Also, the structure of concept relationships is not required to satisfy constraints such as being acyclic. (A concept knowledge hierarchy would normally be 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 the 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. Brief Overview of AHA

The general structure of AHA is somewhat comparable to many other AHS, as described in (Brusilovsky 1996). In (De Bra, Houben & Wu 1999) we defined a reference architecture for AHS, called AHAM. According to AHAM an adaptive hypermedia application consists of three parts: a *domain model*, *user model* and *adaptation model* (previously called *teaching model*). In the (new) AHA system these three parts have the following structure:

The *domain model* describes the application domain in terms 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, not between abstract concepts.)
- **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. (See 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. (See 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 (knowledge) value. This information is stored in a special XML file. All values in AHA are integers between 0 and 100, just like in Da Silva's educational system (Pilar da Silva 1998). AHA also keeps a logfile (per user) with all page accesses. Note that knowledge about fragments is not represented in the user model.

The *adaptation model* consists of author-defined *generate rules* (each corresponding to a set of *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:

```
<concept>
   <conceptname>theconcept</conceptname>
   <relationexpression>req1 > 30 and req2 < 80</relationexpression>
</concept>
```

This entry (in the XML file containing the requirements) means that the desirability of "theconcept" depends on the value of concept "req1" being higher than 30 and the value of concept "req2" being lower than 80.

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

```
<if expr="req1 > 30 and req2 < 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) it is shown in the GOOD color if the page was not visited before, and in the NEUTRAL color otherwise.
    - Links to an *undesirable* page are shown in the BAD color.
- When a fragment is *desirable* (i.e. its requirement is fulfilled) the fragment is included in the page.

The link colors are enforced through the use of link classes combined with a style sheet. So although these colors are blue and purple by default, it is AHA that enforces the use of these colors, not the user's browser.

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 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:

```
<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 5 topics (or pages), each page may *generate* 20% of the knowledge for that section, so that the *value* of that section concept becomes 100 after reading all 5 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.

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. Consequently it was not possible to express that 5 pages contributed towards one "composite" concept. As a result the definition of some requirements like prerequisites was difficult. If some information in a document depended on a chapter of 5 pages having been read completely, the first version of AHA required clauses like:

```
<relationexpression>p1 and p2 and p3 and p4 and p5</relationexpression>
```

In the new AHA one can define each page to generate 20% of a "chapter" (apart from generating 100% of itself). A page for which the entire chapter is prerequisite knowledge then has a requirement:

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

Apart from this, it becomes possible to require at least 4 out of the 5 pages to have been read by specifying:

```
<relationexpression>chapter>=80</relationexpression>
```

(This would require a very long expression in the old AHA system.)

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. In the new AHA such "cascading" knowledge contribution can be *expressed* using recursive rules:

```
<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 also be *performed* recursively by AHA. This is easy to implement, and safe for concept hierarchies (i.e. 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. (See Section 5 for an example.)
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 be easily 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. Potential loops can be easily detected. Unfortunately, many “potential” loops may turn out to not actually be infinite loops, so the authoring system should not forbid their definition.
2. Allow each concept to be updated only once (per run of the algorithm). Unfortunately issue 3 above requires concept D to be updated twice.
3. Allow each *generate rule* to be used only once (per run). Issue 3 above can be easily extended with an update to a concept E that can only be done by applying a *generate rule* from D to E twice.

(Actually, rules 2 and 3 can be modified to allow some other number of iterations instead of 1.)
In AHA we have chosen a perhaps 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 (with a relative value) 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 (with a relative value) 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:

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

(From the 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 (so that repeat visits have an effect).

Note that in this example X may be a positive or a negative value, and independent of that the update to concept2 may recursively trigger other updates while the update to concept3 never triggers other updates.

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. In practice the time needed to perform the adaptation is negligible compared to that for sending the page to the browser.

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) would depend 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 an “explorer”-like menu structures:

- first menu item
    - first submenu *conditionally* appears
    - ...
- second menu item
    - second 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.

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

Each submenu is then conditionally included as follows:

```
  <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:

```
<genitem>
   <name>studentpage1</name>
   <genlist>studentpage1:0 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. So the small updates to “student” and “faculty” are repeated each time “studentpage1” is accessed. By augmenting (the value of) “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. Other educational adaptive systems have a user model architecture that is similar to that of AHA: with knowledge values for concepts. By making it possible that accessing a page decreases the “knowledge” for some concept (as well as supporting increases to knowledge) other AHS can also be made more versatile.

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 (De Bra, Houben & Wu 1999), 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

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.)

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)

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

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).

Pilar da Silva, D. (1998). *Concepts and documents for adaptive educational hypermedia: a model and a prototype.* 2nd Workshop on Adaptive Hypertext and Hypermedia, (pp. 33-40).
(URL: http://wwwis.win.tue.nl/ah98/Pilar/Pilar.html)