Authoring Support for Adaptive Hypermedia Applications

Hongjing Wu, Geert-Jan Houben, Paul De Bra
Department of Mathematics and Computing Science
Eindhoven University of Technology
PO Box 513, 5600 MB Eindhoven
The Netherlands
Email: {hongjing,houben,debra}@win.tue.nl

Abstract: A hypermedia application offers its users a lot of freedom to navigate through a large hyperspace. The rich link structure of the hypermedia application can not only cause users to get lost in the hyperspace, but can also lead to comprehension problems because users read information in an order not foreseen by the author. Adaptive hypermedia systems (or AHS for short) aim at overcoming these problems by providing adaptive navigation support and by providing adaptive content.

We have developed a reference model for adaptive hypermedia applications: AHAM (for *A*daptive *H*ypermedia *A*pplication *M*odel), which is an extension of the Dexter hypermedia reference model (Halasz et al., 1990, 1994). The goal of AHAM is to describe adaptive hypermedia applications, especially from the point of view of authors designing such applications. AHAM divides an AHS into a *Domain Model* (DM), *User Model* (UM) and *Teaching Model* (TM). This paper describes support tools that help authors to create these three parts of an adaptive hypermedia application, and to assure that these parts together form a usable and consistent complete application.

1. Introduction

The rich link structure that is typical of hypermedia applications offers users a lot of freedom to navigate through a large hyperspace. However, it also causes users to lose their way in that hyperspace, and it leads to comprehension problems because users may read pages in an order which the author did not foresee and which may not even make sense. Adaptive hypermedia applications aim at overcoming these problems by providing adaptive navigation support and adaptive content presentation.

An *adaptive hypermedia system* (AHS) (Brusilovsky, 1996) is a special kind of hypermedia system that can adapt the contents and links of an application depending on *observed* user features. For building an AHS, the author has to provide two (related) parts: first of all the author has to write the actual hypertext; second, the author has to specify some dynamic adaptation control information. In practice it turns out to be necessary to provide authoring tools that facilitate the specification of both aspects. In existing AHS (Brusilovsky et al., 1996a, 1996b, De Bra et al., 1997, 1998, Calvi et al., 1997) authoring is often inconvenient, due to the mixture of (textual) content and diverse hints for adaptation. In Interbook (Brusilovsky et al., 1996b) the author must write structured, annotated MS-Word files. In AHA (De Bra et al., 1998) the author must write annotated HTML files. This mixture of content, links and adaptation makes it difficult for authors to maintain a clear picture of how *concepts* relate to each other. This implies that it is hard to define adaptation on an abstract conceptual level.

In (Wu et al., 1998, De Bra et al., 1999) we have presented a reference model for adaptive hypermedia applications, called AHAM (for Adaptive Hypermedia Application Model). In order to separate different authoring aspects, AHAM splits up an adaptive hypermedia application into a *domain model* (DM), *user model* (UM) and *teaching model* (TM). These three parts together form what the Dexter hypermedia reference model (Halasz et al., 1990, 1994) calls the *Storage layer*. Apart from this layer Dexter and AHAM also mention (but do not formally define) a Runtime layer which deals with sessions and the actual presentation and interaction on the screen, and a Within-Component layer which deals with implementation dependent representations of content and links.

In this paper we address operations of DM, UM and TM that are needed for creating (authoring) adaptive hypermedia applications. We describe these operations at the conceptual level, not the implementation level. The definition of these operations shows how the tasks of 1) creating content, 2) combining smaller pieces into composite (abstract) concepts, 3) defining relationships between these concepts, 4) verifying the consistency of these relationships, and 5) defining rules for the adaptation of content and links to each individual user, are separated. The separation of these tasks will make authoring of adaptive hypermedia applications easier than it is today. The separated tasks together form a description of the functionality of an authoring tool based on the AHAM model.

This paper is organized as follows: In Section 2 we describe the AHAM reference model, with its three parts (domain model, user model, teaching model). In Section 3 we show how authoring is facilitated by adding support functions to these three parts. Section 4 concludes and indicates future work.

2. AHAM, a Dexter-based Reference Model

In hypermedia applications the emphasis is always on the information nodes and on the link structure connecting these nodes. The Dexter model captures this in what it calls the Storage layer. It represents a *domain model* DM, i.e. the author's view on the application domain expressed in terms of concepts. In *adaptive* hypermedia applications the central role of DM is shared with a *user model* UM. UM represents the relationship between the user and the domain model by keeping track of how much the user knows about each of the concepts in the application domain.

In order to perform adaptation based on DM and UM an author (or AHS designer) needs to specify how the user's knowledge influences the presentation of the information from DM. In AHAM this is done by means of a *teaching model* TM consisting of so-called *pedagogical rules*. An adaptive engine uses those rules to manipulate link anchors from the Dexter model's *anchoring* and to generate what the Dexter model calls the *presentation specifications*. Like the Dexter model, AHAM focuses on the Storage layer, the anchoring and the presentation specifications. Figure 1 shows the structure of adaptive hypermedia applications in AHAM.

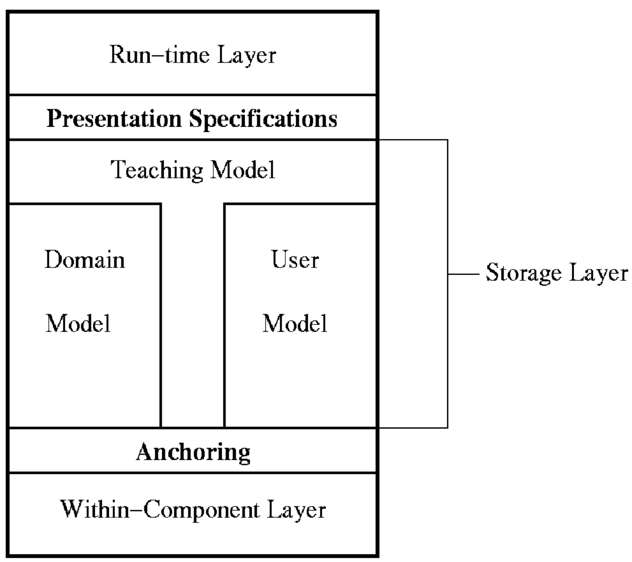

Figure 1: global structure of adaptive hypermedia applications.

In this section we present the elements of AHAM that we will use in Section 3 to illustrate the authoring functionality.

A component is an abstract notion in an AHS. It is a pair (uid, cinfo) where uid is a globally unique (object) identifier for the concept, and cinfo represents the *component's information* consisting of:

- A set of attribute-value pairs;
- A sequence of anchors (for attaching links);
- A presentation specification.

A *concept* is a component representing an abstract information item from the application domain. It can be either an *atomic concept* or a *composite concept*. An *atomic concept* corresponds to a fragment of information. Its attribute and anchor values belong to the Within-component layer and are thus implementation dependent and not described in the model. A *composite component* has two additional attributes:

- A sequence of children (concepts);
- A constructor function (to denote how the children belong together).

The children of a composite concept are either all *atomic concepts* (then we call it a *page* or in typical hypertext terms a *node*) or all *composite concepts*. The composite concept component hierarchy must be a DAG (directed acyclic graph). Also, every atomic concept must be included in some composite concept. Figure 2 illustrates a part of a concept hierarchy.

An *anchor* is a pair (aid, avalue), indicating the endpoint of a hypertext link. The avalue for an anchor in a composite concept is the identifier of a concept that belongs to that composite.

A *specifier* is a tuple (uid, aid, dir, pres), where uid is the identifier of a concept, aid is the identifier of an anchor, dir is a direction (FROM, TO, BIDIRECT, or NONE), and pres is a presentation specification.

A *concept relationship* is a component, with two additional attributes:

- A sequence of specifiers;
- A concept relationship type.

The most common type of concept relationship is the type *link*. This corresponds to the link components in the Dexter model, or links in most hypermedia systems. However, in AHAM we consider other types of relationships as well, that plays a role in the adaptation. A common type of concept relationship is *prerequisite*. When a concept C1 is a prerequisite for C2 it means that the user should read C1 before C2. It does not mean that there must be a link from C1 to C2. It only means that the system somehow takes into account that reading about C2 is not desired before some (enough) knowledge about C1 has been acquired. Figure 3 shows a small set of (only binary) relationships, both prerequisites and links. We treat links and other types of relationships in the same way in order to keep authoring support tools as generic and uniform as possible.

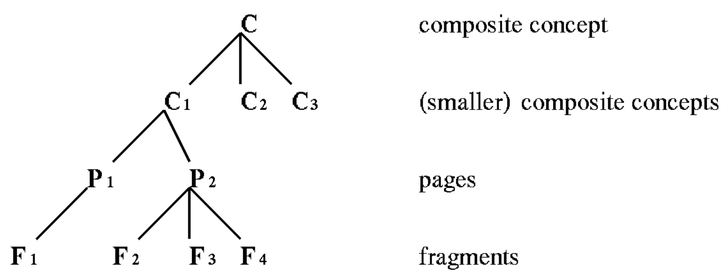

Figure 2: Example concept hierarchy.

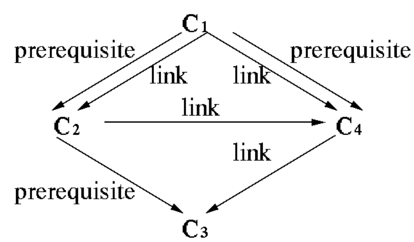

Figure 3: Example concept relationship structure.

The atomic concepts, composite concepts and concept relationships together form the *domain model* DM of an adaptive hypermedia application.

An AHS associates number of (system or author defined) user-model attributes with each concept component of DM. For each user the AHS maintains a *table* in which for each concept the attribute values for that concept are stored. The structure of this table is the *user model scheme*. The table for a specific user is a *user model instance*. If there is no confusion between scheme and instance we just use the term *user model*. One of the attributes of the user model is the unique (concept) identifier. Other attributes commonly found in AHS are the *knowledge value* and some "log" entry whether the user has *read* about the concept. The table below illustrates the (conceptual) structure of a user model, for a course on hypermedia. (In the example table the concepts Xanadu and KMS were at least partially learnt. The concept WWW, consisting of two sub-parts, is partially learnt because WWW-html has been read but WWW-http has not. One can see that WWW must be a composite concept that is not a page because it is already partially learnt while it has not been read at all.)

concept name (uid)	*knowledge value*	*read*	**...**
Xanadu	well learned	true	...
KMS	learned	true	...
WWW-html	well learned	true	...
WWW-http	not known	false	...
WWW	learned	false	...
...	...	...	...

A *generic pedagogical rule* is a tuple (R, PH, PR), where R is a “triggered” rule, PH is the “phase” for the execution of the rule and PR is a Boolean “propagate” field which indicates whether this rule may trigger other rules. The “phase” of a rule can have the value *pre* or *post*. The phase *pre* is executed before and during the generation of the page, while *post* is executed afterwards. These rules describe how an AHS works, but the AHS is not required to be rule based. In many existing AHS the behavior of the system is completely predefined. But in some newer systems, including AHA (De Bra et al., 1998), authors can add some kind of rules to the system.

A *specific pedagogical rule* is a tuple (R, SC, PH, PR) where (in addition to a generic rule) SC is a set of concepts used in the rule. The rule manipulates user-model attributes and “predicates” over specific concepts of SC.

An AHS may have predefined or implicit generic pedagogical rules. If these rules suffice there is no need for a language in which authors can write new rules. Author defined rules take precedence over predefined rules. Specific rules take precedence over generic rules, and are thus used to define exceptions to generic rules. The existence of the two execution phases makes it possible to do adaptation based on the “previous” state of the user model, and to update the user model to a “new” state afterwards.

A typical standard pedagogical rule for the *pre* phase is the rule that says that a link anchor will be *hidden* (or annotated as undesirable) if some prerequisite knowledge for the destination of the link is still not known. A typical standard rule for the *post* phase is the rule that says that a page-concept becomes *well learned* when the page is accessed and all prerequisite knowledge was present, and it becomes only *learned* when the page is accessed but some prerequisite knowledge was still not acquired.

The *teaching model* of an AHS is the set of (generic and specific) pedagogical rules. An *adaptive engine* (AE) is a software environment that performs the following functions (based on the rules):

- It offers generic page constructors.
- It optionally offers a (very simple programming) language for describing new page constructors.
- It performs adaptation by executing the page constructors.
- It updates the user model (instance) each time the user visits a page.

The adaptive engine thus provides the implementation dependent aspects while DM, UM and TM describe the information and adaptation at the conceptual level.

An *adaptive hypermedia application* is a 4-tuple (DM, UM, TM, AE), where DM is a domain model. UM is a user model, TM is a teaching model, and AE is an adaptive engine.

3. Authoring Support for AHAM

In this section we extend the (data) definitions of AHAM to include the operations that are needed for authoring. Because of lack of space we only give these definitions informally. We distinguish three main classes DM-Tool, UM-Tool, TM-Tool. These indicate which authoring functionality is needed for creating (and modifying) a domain model, a user model and a teaching model. When authoring (adaptive or non-adaptive) hypermedia applications it is difficult to verify global properties of the resulting hyperdocument. By separating domain model, user model and teaching model, the verification of properties becomes feasible and sometimes even easy.

DM-Tool must obviously offer methods to create and delete concepts and concept relationships. These methods may generate interesting side-effects that reduce the (manual) work for the author. E.g., when creating a new atomic concept, DM-Tool can show a dialog box to assign this atomic concept to a composite (which must be a page). When a page is deleted, DM-Tool can show a dialog box to let the author decide what to do with the fragments (atomic concepts) of this page. When a concept is added to a composite, DM-Tool can verify whether the composite only contains atomic concepts, or only composite concepts. We won’t go into more detail here, but only describe support methods that work on a global level, and that can be used to verify the integrity of the domain model:

- “verify-hierarchy-is-DAG” checks that no (composite) concept directly or indirectly contains itself (as a child or a lower descendant). This check can be performed automatically each time a new concepts is added.
- “get-orphan-concepts” is used to find the atomic concepts that no longer belong to a page. It is not desirable to force the deletion of fragments when a page is deleted. The author may wish to keep some fragments and add them to other (new) pages later.

Each concept relationship type may have its own constraints. The combination of concept relationship types may introduce additional constraints. Whether DM-Tool can verify these constraints depends on their definition. Some methods for verifying common constraints are:

- "verify-connected-links" checks whether every composite concept (including pages) can be reached by following links from a given starting page.
- "verify-acyclic-prerequisites" checks whether no concept is a prerequisite for itself, either directly or indirectly.
- "verify-conditional-links" checks whether for every composite concept (including pages) there exists a path leading to that concept, from a given starting page, starting with the "default" user-model instance, and without visiting concepts for which prerequisite knowledge is missing.

The third method is a nice example of how the verification of constraints becomes difficult when different types of concept relationships are combined. Also, the verification is based on implicit assumptions about how the user model evolves. Indeed, this verification only works if we make assumptions about how the user model evolves based on the user visiting certain concepts. Figure 3 for instance shows a *sound* structure, only under the assumption that visiting a concept C_i implies that enough knowledge is gained about C_i to warrant that prerequisites that depend on C_i are fulfilled.

In an object-oriented view the UM-Tool can be considered as an object that contains DM-Tool as a part. When UM-Tool contains DM-Tool it becomes possible to model the interaction between the user model and the domain model as being part of the functionality of UM-Tool.

The authoring support of an AHS must include a method for propagating updates to the domain model DM to the user model. In our view this means that UM-Tool needs a method for importing all concepts from DM into UM. However, an author may wish to keep the user model simpler by only keeping some concepts in the user model. Therefore, we have an auxiliary method "verify-all-concepts-in-DM" to check whether all the concepts from UM are in DM, but no method to verify the opposite (because the opposite may not be desirable).

UM-Tool has different kinds of methods:

- "add/delete-concept" is normally available to the author, to manipulate the user model and eliminate low level concepts that exist in DM but are not of interest. (The author may wish to define the adaptation based on knowledge of high level concepts only.)
- "add/delete-attribute" is a method to add a user-model attribute to every concept. It means that the AHS is able to record values for this attribute about each concept. While the AHAM model describes the user model as a table structure in which all attributes are available for all concepts, a particular AHS may have a more restrictive implementation. In the AHA system (De Bra et al., 1998) for instance, there are some "concepts" that represent color preferences for links. These concepts only have a "color" attribute, while the other (real) concepts only have "knowledge-value" and "read" attributes.

Whether authors can extend the user model with new attributes depends on the AHS. The limiting factor is the ability or inability to actually use new attributes. The *rules* in TM are the (only) means to use new attributes.

TM-Tool provides methods for creating *pedagogical rules*. In the object-oriented view TM-Tool contains DM-Tool and UM-Tool as parts. This makes it possible to describe the interaction between DM-Tool and UM-Tool as being part of the functionality of TM-Tool.

- "add-spec-pre/post-rule" are methods for creating *specific* rules. The author can add rules that work with *specific* concepts. The syntax of such rules depends on the AHS. An example of the use of specific rules in AHA (De Bra et al., 1998) is:

```
...
<!-- if readme and not intro -->
      This text only appears if "readme" is a known concept
      and "intro" is not yet known.
<!-- endif -->
...
```

- "add-gen-pre/post-rule" are methods for creating *generic* rules. Generic rules specify how the AHS should handle events and concept relationships of a given type. A *generic* rule about *prerequisite* relationships could say:
  - When a user visits concept C and all prerequisites are satisfied the knowledge about C becomes 0.8.
  - When a user visits concept C and not all prerequisites are satisfied the knowledge about C becomes 0.5.

  This rule applies to *all* prerequisite relationships, while the above example from AHA only works for a specific instance, with two specific concepts.

If TM-Tool lets authors create their own generic rules, possibly using their own attributes added to UM, it may not be possible to automatically verify the consistency of the rules. In the current version of AHA (De Bra et al., 1998) only specific rules are permitted, using a simple and not very expressive language. While this limits the power of

AHA it makes it possible to perform the verification that all concepts (pages) can be reached from the given starting page without violating any of the conditions.

An advanced AHS will let authors define their own (generic and specific) pedagogical rules. However, in the near future (and in our redesign of AHA) TM-Tool will most likely only let authors *configure* predefined rules. In the example above, the knowledge values of 0.5 and 0.8 could be author-configurable for instance.

4. Conclusion and Future Work

We have defined authoring support by specifying three main classes UM-Tool, DM-Tool, and TM-Tool. In the class definitions we have described the data model and the support operations for authoring. The division of adaptive hypermedia applications into a *domain model*, *user model* and *teaching model* makes the task of authoring clearer by separating different concerns. We have also indicated the main support functions that are needed to ensure that the three generated parts are internally consistent and also consistent with each other.

We are currently working on a redesign (and re-implementation) of the AHA system (De Bra et al., 1998) which is being used at the Eindhoven University of Technology for a course on hypermedia (URL http://wwwis.win.tue.nl/2L690/), a course on graphical user interfaces (URL http://wwwis.win.tue.nl/2M350/), and a kiosk information system for students who wish to do an internship or master thesis in information systems (URL http://wwwis.win.tue.nl/IShype/). AHA is also being used for the development of a course in language learning (Calvi, 1998). AHA currently offers authors the ability to specify *specific rules* for the inclusion of fragments or for determining the desirability of links. We hope to extend AHA with the ability to specify *generic rules* as needed by richer tools for creating user models and teaching models.

5. References

Brusilovsky, P. (1996). *Methods and Techniques of Adaptive Hypermedia*. In: User Modeling and User-Adapted Interaction 6: 87-129, Kluwer academic publishers, 1996.

Brusilovsky, P., Schwarz, E., Weber, G. (1996a). *ELM-ART: An intelligent tutoring system on World Wide Web*, Proceedings of the Third International Conference on Intelligent Tutoring Systems, ITS-96, Montreal, 1996. (Lecture Notes in Computing Science, vol. 1086, pp. 261-269).

Brusilovsky, P., Schwarz, E., Weber, G. (1996b). *A Tool for Developing Adaptive Electronic Textbooks on WWW*, Proceedings of the WebNet'96 Conference, pp. 64-69, San Francisco, 1996.

Calvi, L. (1998). *A Proficiency-Adapted Framework for Browsing and Information Filtering in Web-Based Educational Systems. Methodological Implications for Language Learning on the WWW*, Doctoral Thesis, University of Antwerp, 1998.

Calvi, L., De Bra, P. (1997). *Using Dynamic Hypertext to create Multi-Purpose Textbooks*. Proceedings of ED-MEDIA'97, pp. 130-135, Calgary, 1997.

De Bra, P., Calvi, L. (1997). *Creating adaptive hyperdocuments for and on the Web*, Proceedings of the WebNet'97 Conference, pp. 149-165, Toronto, 1997.

De Bra, P., Calvi, L. (1998). *AHA: a Generic Adaptive Hypermedia System*, Proceedings of the Second Workshop on Adaptive Hypertext and Hypermedia, pp. 5-11, Pittsburgh, 1998.

De Bra, P., Houben, G.J., Wu, H. (1999*). AHAM: A Dexter-based Reference Model for Adaptive Hypermedia*, Proceedings of ACM Hypertext'99 Conference, pp. 147-156, Darmstadt, Germany, 1999.

Halasz, F., Schwartz, M. (1990). *The Dexter Reference Model*, Proceedings of the NIST Hypertext Standardization Workshop, pp. 95-133, 1990.

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

Wu, H., Houben, G.J., De Bra, P. (1998). *AHAM: A Reference Model to Support Adaptive Hypermedia Authoring*, Proceedings of the Conference on Information Science, pp.51-76, Antwerp, 1998.