AHAM: A Dexter-based Reference Model for Adaptive Hypermedia

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

ABSTRACT

Hypermedia applications offer users the impression that there are many meaningful ways to navigate through a large body of information nodes. This rich link structure not only creates orientation problems, it may also be a source of comprehension problems when users follow paths through the information which the author did not foresee. Adaptive techniques have been used by a number of researchers [1, 2, 4, 5, 6, 7, 8, 9, 10, 17, 19, 20, 22] in an attempt to offer guidance through and orientation support for rich link structures. The majority of these adaptive hypermedia systems (AHS) have been used in educational applications. The terminology used in this paper also has an educational "flavor". However, there are some adaptive on-line information systems (or "kiosk"-systems), adaptive information retrieval systems, and other adaptive hypermedia applications.

In this paper we describe a reference model for adaptive hypermedia applications, called AHAM, which encompasses most features supported by adaptive systems that exist today or that are being developed (and have been published about). Our description of AHS is based on the Dexter model [15, 16], a widely used reference model for hypermedia. The description is kept somewhat informal in order to be able to *explain* AHAM rather than formally *specify* it. AHAM augments Dexter with features for doing adaptation based on a *user model* which persists beyond the duration of a session. Key aspects in AHAM are:

- The adaptation is based on a *domain model*, a *user model* and a *teaching model* which consists of *pedagogical rules*. We give a formal definition of each of these (sub)models (but only describe the pedagogical rules informally through examples).
- We distinguish the notions of *concept*, *page* and *fragment*. In some AHS these notions are confused.
- We provide a formalism which lets authors write pedagogical rules (about concepts) in such a way that they can be applied automatically.

We illustrate various aspects of AHAM by means of some features of some well-known AHS [6, 10].

Keywords: adaptive hypermedia, user modeling, hypermedia reference model.

INTRODUCTION

Hypermedia systems in general, and Web-based systems in particular, are becoming increasingly popular as tools for user-driven access to information. Many hypermedia systems have been developed during the past thirty years. In 1988 and 1990 a number of researchers and developers came together to define a common reference model for "modern" hypermedia systems. The resulting "Dexter model" was published at a NIST workshop [15], together with some other models [12, 18], and later also in the Communications of the ACM [16]. The Dexter model describes an architecture that is more powerful in some areas than any hypermedia system that exists today. However, the model is showing its age in other areas. This has resulted in some newer proposals for models like the "Tower model" [11], HDM [13] and OOHDM [23]. However, the Dexter model remains by far the most widely used reference model, which is still suited for modeling most kinds of hypermedia applications. This is also why we use Dexter as the basis in this paper.

In recent years a number of *adaptive hypermedia systems* (AHS) have been developed [2, 4, 5, 6, 9, 10, 19, 22]. In adaptive hypermedia applications the system keeps track of evolving aspects of the user, such as preferences and domain knowledge. This permanent and continuously updated record is called a

*Paul De Bra is also affiliated with the University of Antwerp, Belgium, and with the "Centrum voor Wiskunde en Informatica" (CWI) in Amsterdam.

†Geert-Jan Houben is also affiliated with the University of Antwerp, Belgium, and with Origin in Eindhoven.

user model. It is used to guide the user towards interesting new information and away from information the system considers not to be appropriate or relevant for the user. The AHS may do this by dynamically altering the hyperdocument's link structure and/or by dynamically generating or changing the content of the information nodes. AHS are finding their way into several different application areas, such as information retrieval systems and educational systems that offer guidance to students who are exploring an information space.

AHS are usually also *adaptable*, which means that the user can set certain preferences explicitly or initialize the user model through a registration form or "pre-test". In this paper we concentrate on automatic adaptation based only on browsing, not on questions and answers. Testing and setting of preferences is considered external functionality. We give a brief general description of how to combine AHS with external functions, and of how to let different AHS communicate with each other.

Brusilovsky [3] describes adaptive hypermedia as follows:

By adaptive hypermedia systems we mean all hypertext and hypermedia systems which reflect some features of the user in the user model and apply this model to adapt various visible aspects of the system to the user. In other words, the system should satisfy three criteria: it should be a hypertext or hypermedia system, it should have a user model, and it should be able to adapt the hypermedia using this model.

The first aim of this paper is to describe AHS using known and generic terminology. For that purpose we try to fit adaptive hypermedia in with (a slightly extended version of) the Dexter model. This enables us to not only characterize and compare different adaptive hypermedia systems, but also to formally define AHS as (Dexter-based) hypermedia systems and show how to plug adaptive techniques into the Dexter model. Readers not familiar with reference models for hypermedia in general, and with the Dexter model in particular, are urged to review references [15, 16].

A second, but not secondary aim of the **A**daptive **H**ypermedia **A**pplication **M**odel (AHAM) which this paper defines is to provide a sound basis for the development of new AHS. This is achieved through a clear distinction between the following items, which are often confused and mixed in AHS:

- The *domain model* describes how the information is structured and linked together. It corresponds (roughly) to the *storage layer* of the original Dexter model.
- The *user model* describes of which information about the user an AHS keeps a permanent record. This includes a representation of the knowledge which the user gains but also a record of the nodes visited by the user.
- The *teaching model* consists of *pedagogical rules* which define how the domain model and the user model are combined to provide ways to perform the actual adaptation. (The terms *teaching* and *pedagogical* are not meant to imply that AHAM would only be useful for adaptive hypermedia used in educational applications.)
- An *adaptive engine* performs the actual adaptation by adapting or dynamically generating the content of nodes and the destination and "class" of links in order to guide each individual user differently.

This paper is organized as follows: the next section briefly recalls the basic concepts of adaptive hypermedia, as described in Brusilovsky's overview paper [3]. Subsequently we present the architecture of adaptive hypermedia applications in terms of the Dexter-based AHAM model. The paper ends with two short sections: one illustrates how AHAM can be used to define communication between adaptive hypermedia applications and between such applications and other system like systems for evaluation a user's knowledge; the final section describes future work on aspects of Dexter that are not yet covered by AHAM, e.g. in the area of authoring.

CONCEPTS OF ADAPTIVE HYPERMEDIA

Brusilovsky [3] distinguishes between high level methods for adaptive hypermedia support and lower level techniques that are used to realize or implement that support. By a method we mean a notion of adaptation that can be presented at the conceptual level. A technique is then a way to implement a specific method. Techniques operate on actual information content and on the presentation of hypertext links. It may be possible to implement the same method through different techniques and to use the same technique for different methods.

We distinguish between *content-adaptation* and *link-adaptation*. (Brusilovsky [3] calls these *adaptive presentation* and *adaptive navigation*.) We do so both at the level of methods and that of techniques. We only present a very brief overview of the methods and techniques. For a more detailed discussion of adaptive hypermedia we refer to [3].

Content-adaptation

It may be desirable to present information on a certain topic in different ways, depending on the user's (fore)knowledge, goals, preferences or other characteristic properties of the user. Introductory explanations may be added for novices, advanced details for experts. A description may exist in different versions, for users with different foreknowledge. The order in which items are displayed (on a page) may also be different for different users. At the level of adaptation methods we can thus distinguish three methods:

- additional, prerequisite, and comparative explanations
- explanation variants
- (content) sorting

Brusilovsky mentions the following techniques for content-adaptation (see [3] for details):

- conditional text
- stretchtext
- fragment variants
- page variants
- frame-based technique(s)

Conditional text can easily be used to implement stretchtext, fragment variants and page variants. It is the lowest level technique and is sufficient to implement the different types of additional explanations and explanation variants. When a content fragment is considered not desirable most AHS will leave it out. Preliminary evaluation of the "SAD" system [17] however suggests that users may prefer these fragments to be grayed out but still readable.
Frame-based techniques are used in AHS that dynamically create presentations by applying natural language generation techniques to paste together small fragments of information and turning them into fluent text [21].

Link-adaptation

The basic idea with link-adaptation is to change or annotate the rich link structure in such a way that the user is guided towards interesting, relevant information, and kept away from non-relevant information. Link-adaptation tries to simplify the rich link structure to reduce orientation problems, while maintaining a lot of navigational freedom, a typical property of hypermedia systems. Link adaptation methods are:

- guidance, either at a global or local level;
- global or local orientation support;
- managing personalized views (on the link structure).

Guidance is offered by somehow indicating which links are to be preferred over others. Orientation support depends on providing context. It requires some (possibly textual) map of the link structure around the "current" node (page). Note that generating such a map or hierarchical table of content may also be viewed as content-adaptation (or rather, content-generation). The techniques found in [3, 10] for link-adaptation are:

- direct guidance (e. g. a "next" button);
- link sorting (like in search engines);
- link hiding (hide non-relevant links, but keep anchor text);
- link annotation (e. g. use colors to indication relevance);
- link disabling (make non-relevant links not work);
- link removal (remove non-relevant link anchors);
- map adaptation (provide personalized overview).

There are no AHS that support all the methods and techniques presented in this section. Using all link-adaptation techniques simultaneously would lead to an unusable system. But in future systems an application designer (author) may be able to select the techniques he or she desires, and have one system that supports each choice. In the next section we present the AHAM reference model, in which it is possible to represent AHS that offer all these techniques.

THE ADAPTIVE HYPERMEDIA APPLICATION MODEL (AHAM)

In hypermedia applications the emphasis is always on the content of the information *nodes* and on the *link* structure. The Dexter model [15, 16] confirms this by concentrating on what it calls the *storage layer*. It represents a *domain model*, i. e. the author's view on the application domain.

In adaptive hypermedia applications the central role of the *domain model* is shared with a second part: the *user model*. A *user model* represents how the user relates to the *domain model*. The application domain deals with a number of *concepts*. The *user model* keeps track of how much the user knows about each of the concepts of the application domain.

In order to perform adaptation based on the domain model and user model we need to specify how the user's knowledge influences the way in which the information from the domain model is to be presented. We do this by means of a *teaching model* which consists of *pedagogical rules*. The rules are used by an *adaptive engine* in order to generate what the Dexter model calls the *presentation specifications*. Figure 1 shows the AHAM model as an extension of the Dexter model.

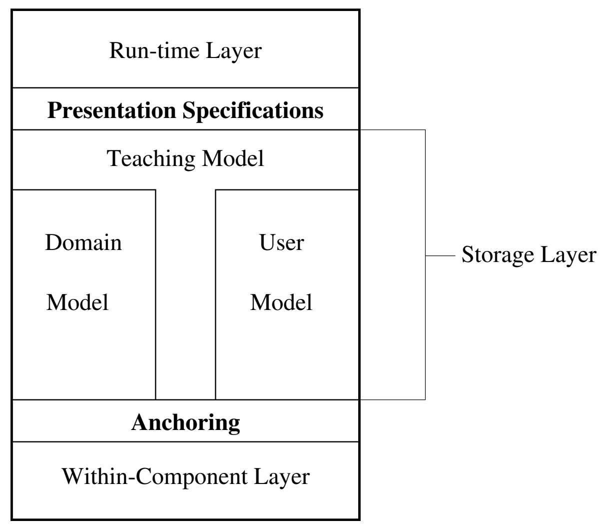

Figure 1: The AHAM model

Like the Dexter model, AHAM focusses on the *storage layer*, the *anchoring* and the *presentation specifications*. In Dexter, the central notion of the storage layer is the *component*. This notion covers both nodes and links. In adaptive hypermedia the central notions are the *concepts* and *concept relationships*.

Definition 1 *A* concept component *(or* concept *for short) is an abstract representation of an information item from the application domain. A concept is a pair* <uid, cinfo>*, where* uid *is a globally unique (object) identifier for the concept, and* cinfo *is the component information. A component's information consists of:*

- *a* set of attribute-value pairs*;*
- *a* sequence of anchors*;*
- *a* presentation specification*.*

The structure of attribute values, anchors and presentation specifications is defined below.

The Dexter model distinguishes atomic from composite components. AHAM does the same for concepts.

Definition 2 *An* atomic concept component *corresponds to a* fragment *of information. It is primitive in the model. Its attribute and anchor values belong to the* within-component layer. *They are not described in AHAM (or Dexter).*
A composite concept component *has a* children *attribute which (has a value that) is a sequence of concepts. It may also have a* constructor *attribute which indicates a possible "structure" of the composite, and which may indicate "how much" of the composite each subcomponent represents. In this paper we only consider two kinds of composite concepts:*

- *An* abstract *composite concept has only children which are composites themselves.*
- *A* page *concept has only children which are atomic concepts.*

The composite concept component hierarchy must be a directed acyclic graph, i. e. no component can be a subcomponent of itself, either directly or indirectly. Also, in AHAM every atomic (fragment) concept is required to be a subcomponent of at least one page concept.

Note that composites that have composite as well as atomic children can be simulated by introducing extra intermediary composites. The restriction in our definition simplifies the implementation of *accessor* functions that translate uid's to components in order to allow their presentation. In order to decide how to present a concept, the accessor function uses the *constructor* of abstract composites to select one or more subcomponents. This process is repeated until the subcomponent is a page concept. The constructor for the page helps to build a presentation by selecting (and possibly sorting) fragments.

The *sequence of anchors* of a concept component provides a way for links to be attached to a specific part of a component:

Definition 3 *An* anchor *is a pair* <aid, avalue>*, where* aid *is a unique (object) identifier for the anchor within the scope of its component and* avalue *is an arbitrary value that specifies some location, region, item or substructure within a concept component. Anchor values of atomic components belong to the* within-component layer *and are not elaborated in AHAM (or Dexter). Anchor values of composite components are* uid-aid *pairs, where the* uid *identifies a subcomponent of the composite and the* aid *identifies an anchor within that subcomponent.*

In the Dexter model, a component is either an atom, a composite or a link. Links represent relationships between components. While the term *link* suggests that these relationships are used for navigation, the model does not require that. In AHAM we chose the term *concept relationship* to make it more explicit that these relationships are used for many purposes, not just for hypertext navigation. In this paper we only consider relationships between concept components. The Dexter model also allows links of which (some) endpoints are links. We do not (yet) consider such relationships, mostly because we know of no existing AHS that offers this feature.

Concept relationships in AHAM, just like *links* in the Dexter model, consist of sequences of components. These components however are not given as a unique identifier, but rather as a *specifier* which needs to be *resolved* to an identifier (or a set of identifiers). In AHAM, unique identifiers are used as specifiers, but these are not (always) the identifiers of the "real" endpoint of a link or the element in a concept relationship. When the endpoint or element is a composite concept component, it needs to be "resolved" by traversing the composite hierarchy down to the page level. This is necessary because only pages can be displayed. Like in the Dexter model AHAM requires that there exists a *resolver* function which performs this translation. This function is part of the *adaptive engine* of the AHS.

Definition 4 *A* specifier *is a tuple* <uid, aid, dir, pres> *where* uid *is the uid of a concept component,* aid *is the id of an anchor,* dir *is a direction (which is* FROM*,* TO*,* BIDIRECT *or* NONE*), and* pres *is a* presentation specification.

This definition suggests that there are no "computable" specifiers. However, the computable part is hidden in the Dexter model's *accessor* and *resolver* functions that must translate the uid of an abstract concept to the uid of one or more pages to be presented.

Definition 5 *A* concept relationship component *is a component* <uid, ss, cinfo> *where* uid *is the identifier of the relationship component,* ss *is a* sequence of (one or more) specifiers *and* cinfo *is the component information which consists of:*

- *a* set of attribute-value pairs*; this set must include an attribute* type*;*
- *a* sequence of anchors*;*
- *a* presentation specification.

The Dexter model allows for relationships with a sequence of just one specifier (see e. g. [14]). While we allow such relationships for conformance with Dexter, current AHS only consider relationships between at least two specifiers.

The most common type of concept relationship is of course the type *link*. It corresponds to *link components* in the Dexter model that are used for hypertext navigation. However, in AHAM we consider other types of relationships as well that play a role in the adaptation. Which relationship types exist depends on the AHS. Some AHS may allow authors to "invent" new relationship types as desired. Some relationship types found in existing AHS are:

- A concept relationship of type *prerequisite* means that for the sequence of specifiers holds that the specifiers with a direction FROM represent *prerequisite knowledge* for the specifiers with a direction TO. We shall see later how this corresponds to a rule which says that all prerequisite knowledge must be acquired in order to gain (desired) access to the concepts identified by the TO-specifier(s). The relationship may also have an attribute-value pair for prerequisite concepts to indicate how much of a concept needs to be known in order to gain access to the TO-specifier(s).
- A concept relationship of type *inhibitor* is "complementary" to the type *prerequisite*. It means that for the sequence of specifiers it holds that in order to have (desired) access to the TO-specifier(s) the user must not have too much knowledge about (the concepts that correspond to) all the FROM-specifiers. Attribute-value pairs are used to indicate how much knowledge about each FROM-specifier is allowed in order to still have access to the TO-specifier(s).

Definition 6 *The* atomic concept components*,* composite concept components *and* concept relationship components *together form the* domain model *of an adaptive hypermedia application.*

Adaptive hypermedia systems distinguish themselves from other hypermedia systems by maintaining a (permanent and continuously updated) *user model*. From figure 1 we see that the user model in AHAM is part of the *storage layer*. According to the general design of the Dexter model this is a logical place for the user model. The *run-time layer* in Dexter would be able to perform some adaptive functions, but only within a single browsing session. Adaptive hypermedia applications need to maintain a permanent user model. Such permanent information which exists at the conceptual level also does not belong in the *within-component layer* because that layer deals with implementation-specific elements which are not elaborated in the Dexter model.

In adaptive hypermedia systems a user model is based on a user's knowledge about *concepts*. Different AHS may store different information in a user model, besides a representation of the user's knowledge. Most AHS also keep track of which pages (nodes) a user has read. In AHAM pages also correspond to concepts, so we can store knowledge and browsing history information as attributes of a concept in the user model.

Definition 7 *An AHS associates a number of (system or author defined)* user-model attributes *to each concept component of the domain model. For each user the AHS maintains a table (a relation in database terminology) in which for each concept component 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*. The "first" attribute of the user model is always the concept* uid*.*

AHAM does not require the presence of specific attributes in the user model (other than the concept uid). However, almost all AHS store at least the following two attributes:

- The *knowledge value* (or *value* for short) indicates how much the user knows about the concept. The concept-value pairs together form an *overlay model*, which represents the "knowledge" of the user. Some AHS use a "Boolean user model" [9, 10], meaning that for each concept the user either knows or does not know the concept. Other AHS use either a small set of knowledge values [5, 6], like "not known", "learned", "well learned" and "well known", or even a large set, such as a percentage or a (real) value between 0 and 1 [22].
- The *read* attribute indicates whether the user read something (a fragment, a page or a set of pages) about the concept. In Web-based systems the *read* attribute is used to generate a different presentation for anchors of links to pages that were read before than for links to previously unread pages. (By default the difference is a purple vs. blue color for the anchor text or image border.) The *read* attribute may have Boolean values in some AHS while it may be a list of access times in other AHS.

A less common attribute would be *ready-to-read*, which indicates whether the user is ready to read about this concept. (This means for instance that enough prerequisite knowledge has been acquired.) Figure 2 shows an example of a user model (instance). The example assumes that WWW_1 and WWW_2 are subconcepts of the composite WWW. By learning everything about WWW_1 (but still nothing about WWW_2) the composite WWW already becomes "learned". It is thus possible to already have learned something about a concept while "read" is still false. It is even possible that there is no page for the concept WWW, and that it thus can only be learned by reading about subconcepts.

uid (name)	knowledge value	read	ready-to-read
intro	well learned	true	true
Xanadu	learned	true	true
KMS	not known	false	true
WWW_1	well known	true	true
WWW_2	not known	false	true
WWW	learned	false	false

Figure 2: Example user model instance

The "table" representation of a user model in AHAM is only a conceptual representation. Actual implementations of AHS may implement this structure in a different way. The AHA system [9, 10] for instance uses a logfile (separate for every user) in which each time is logged at which a user requests a page, and each time when a user leaves the page. Also, an AHS may implement just one user model table for all users together, by adding a "user-id" attribute.

While most AHS to date provide a fixed set of attributes, future AHS may offer the possibility for authors to "invent" new attributes. For the AHAM model this makes no difference. In the sequel we shall use the notation *C.attr* as a convenient way to denote the value for the attribute *attr* for the concept with uid *C* and for the "current" user. When *Xanadu.read* is true, it means the user has read the page about concept Xanadu.

In AHAM the basis for adaptive functionality can be found in the *teaching model*, which combines information from the *domain model* and the *user model* in rules that determine how information is to be presented.

Definition 8 *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 syntax of the permissible rules depends on the AHS. It will normally be much simpler than the syntax of the examples below. A rule uses and possibly changes variables which denote concept uid's, attributes, anchors, parts of presentation specifications and user-model attributes for concepts and concept relationships. The "phase" of a rule can have the value* pre *or* post. *The phase* pre *is executed before and during the generation of a page (or pages), while* post *is executed afterwards.*

Definition 9 *A* specific pedagogical rule *is a tuple* <R, SC, PH, PR> *where* R *is a "triggered" rule,* SC *is a set of concept components used in the rule,* PH *is the "phase" and* PR *is the Boolean "propagate" field. The rule uses and sets user-model attributes and "predicates" over the 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 reason for having two execution phases is that one may wish to do first some adaptation based on the "current" state of the user model (*pre*) and to then update the user model to a new state after generating the presentation of the page(s) that result from following a link (*post*).

Definition 10 *The* teaching model *of an AHS is the set of (generic and specific) pedagogical rules.*

We do not formally specify a language for expressing pedagogical rules. We only give a few examples of typical generic and specific rules, using an arbitrarily chosen syntax. For the application of these rules we assume that the AHS is displaying one or more pages, and that the user "clicks" on a link anchor. This activates the *followLink* operation from the Dexter model's run-time layer, which in turn results in a call to a *resolver function* for the specifier corresponding to the link anchor (on the given page). In AHAM the resolver *must* translate the given specifier to the uid of a composite concept component that *corresponds to a page*, or to a set of such uid's. (It may require several recursive calls to the resolver to go down from a high-level abstract concept down to the page level.) Which page(s) is or are selected depends on the domain model (that defines the hierarchy and structure of composites as well as concept relationships that may indicate a preferred reading order) and on the user model (that states what the user's knowledge about different concepts is). For each selected page an *accessor function* is called, according to the Dexter model, which returns the (page) concept component that corresponds to the resolved uid. The pedagogical rules in AHAM are "triggered" through this *accessor function*.

Example 1 *The following rule expresses that when a page is accessed the* read *user-model attribute for the corresponding concept is set to true in the* post *phase:*

$$< \mathit{access}(C) \Rightarrow C.\mathit{read} := \mathit{true}\ ,\ \mathit{post}\ ,\ \mathit{true} >$$

The rule also says that it will trigger other rules that have read *on their lefthand side.*

Example 2 *The following rule expresses that when a page is "ready-to-read" and it is accessed, the knowledge value of the corresponding concept becomes "well learned" in the* pre *phase. This is somewhat like the behavior of Interbook [6].*

$$< \mathit{access}(C) \text{ and } C.\mathit{ready\text{-}to\text{-}read} = \mathit{true} \Rightarrow$$
$$C.\mathit{knowledge\text{-}value} := \mathit{well\ learned}\ ,\ \mathit{pre}\ ,\ \mathit{true} >$$

In this example the phase was chosen to be *pre* because this is the behavior of Interbook and many other AHS. This choice is counterintuitive but illustrates a shortcoming of many AHS (but for which there is no easy solution): the presentation of a page is adapted to the knowledge the user will have *after* reading the page. This behavior is motivated by the need to present as "relevant" the anchors for links to pages that only become relevant after reading the page. By having two phases in the AHAM model it becomes possible to describe the behavior of future AHS that will register the new knowledge *after* the page has been generated and presented.

Example 3 *The following rule illustrates how a prerequisite relationship works: this generic rule states that a prerequisite relationship between two concepts is satisfied when the prerequisite concept is at least "well learned". For simplicity we assume that when CR is a concept relationship, the uid of the i-th specifier is CR.ss*$[i]$*.uid, the knowledge-value of the concept in the user model is CR.ss*$[i]$*.uid.knowledge-value, the relationship type is CR.cinfo.type, the direction of the i-th specifier is represented CR.cinfo.dir*$[i]$*, etc. We also assume that* *knowledge values can be compared using $>$ and $\geq$, where higher values mean more knowledge. (A completely correct syntax would be more complicated because of the complex nature of relationships and specifiers in the Dexter model, and thus also in AHAM, and because we would need to discriminate between attributes of concepts in the domain model and in the user model.)*

$$< \mathit{CR.cinfo.type} = \mathit{prerequisite\ and\ CR.cinfo.dir}[1] = \mathit{FROM}$$
$$\mathit{and\ CR.cinfo.dir}[2] = \mathit{TO\ and\ CR.ss.length} = 2\ \mathit{and}$$
$$\mathit{CR.ss}[1].\mathit{uid.knowledge\text{-}value} \geq$$
$$\mathit{CR.cinfo.required\text{-}knowledge}[1]$$
$$\Rightarrow \mathit{CR.ss}[2].\mathit{uid.ready\text{-}to\text{-}read} := \mathit{true}\ \ ,\ \mathit{pre}\ ,\ \mathit{true} >$$

Note that this rule only "works" if it is triggered. Example 2 shows that from an "access" event a change to the knowledge is generated which propagates as a new event. So if the knowledge value of CR.ss[1] is set through an "access" event, that triggers the rule given in this example.

We now turn to examples that deal with the presentation aspect of an AHS. In the Dexter model, and also in AHAM, the link between the storage layer and the run-time layer is formed by *presentation specifications*, which are not described in detail. We give a few examples of how pedagogical rules are used to generate (parts of) presentation specifications.

Example 4 *For atomic concepts (i. e. fragments) we assume the presentation specification is a two-valued (almost Boolean) field, which is either* show *or* hide. *When a page is being accessed, the following rule sets the visibility for fragments that belong to a "page" concept, depending on their* ready-to-read *state.*

$$< \mathit{access}(C)\ \mathit{and}\ F \in \mathit{C.children\ and}$$
$$\mathit{F.ready\text{-}to\text{-}read} = \mathit{true} \Rightarrow$$
$$\mathit{F.pres} := \mathit{show}\ \ ,\ \mathit{pre}\ ,\ \mathit{false} >$$

Here we again simplified things, by assuming that we can treat C.children as if it were a set, whereas it really is a sequence. A similar but complementary rule to set the visibility to hide *is straightforward. This presentation specification is used by the adaptive engine of the AHS to include only those fragments in a page that are* ready-to-read.

Note that the update to the presentation specification is not propagated: the presentation specification is passed on to the implementation-specific part of the AHS that is not part of the AHAM model. (Thus, a presentation specification is *set* by a pedagogical rule, but not *read* by other rules.) However, there may be other rules that are triggered by the "access" event, for instance a rule executed in the *post* phase that will set the *read* attribute of the fragments to *true*.

Example 5 *The following rules set the presentation specification of a specifier that denotes a link (source) anchor depending on whether the destination of the link is considered relevant (*ready-to-read*) and whether the destination has been read before. For simplicity we consider a link with just one source and one destination.*

$$< \mathit{CR.type} = \mathit{link\ and\ CR.cinfo.dir}[1] = \mathit{FROM\ and}$$
$$\mathit{CR.cinfo.dir}[2] = \mathit{TO\ and\ CR.ss}[2].\mathit{uid.ready\text{-}to\text{-}read} = \mathit{true}$$
$$\mathit{and\ CR.ss}[2].\mathit{uid.read} = \mathit{false} \Rightarrow$$
$$\mathit{CR.ss}[1].\mathit{pres} := \mathit{GOOD}\ \ ,\ \mathit{pre}\ ,\ \mathit{false} >$$

$$< \mathit{CR.type} = \mathit{link\ and\ CR.cinfo.dir}[1] = \mathit{FROM\ and}$$
$$\mathit{CR.cinfo.dir}[2] = \mathit{TO\ and\ CR.ss}[2].\mathit{uid.ready\text{-}to\text{-}read} = \mathit{true}$$
$$\mathit{and\ CR.ss}[2].\mathit{uid.read} = \mathit{true} \Rightarrow$$
$$\mathit{CR.ss}[1].\mathit{pres} := \mathit{NEUTRAL}\ \ ,\ \mathit{pre}\ ,\ \mathit{false} >$$

$$< \mathit{CR.type} = \mathit{link\ and\ CR.cinfo.dir}[1] = \mathit{FROM\ and}$$
$$\mathit{CR.cinfo.dir}[2] = \mathit{TO\ and\ CR.ss}[2].\mathit{uid.ready\text{-}to\text{-}read} = \mathit{false}$$
$$\Rightarrow \mathit{CR.ss}[1].\mathit{pres} := \mathit{BAD}\ \ ,\ \mathit{pre}\ ,\ \mathit{false} >$$

These rules say that links to previously unread but "ready-to-read" pages are "GOOD", links to previously read and "ready-to-read" pages are "NEUTRAL" and links to pages that are not "ready-to-read" are "BAD". In the AHA system [9, 10] this results in the link anchors being colored blue, purple or black respectively. In ELM-ART [5] and Interbook [6] the links would be annotated with a green, yellow or red ball.

The above examples illustrate how the *adaptive engine* of an AHS can use pedagogical rules to generate presentation specifications. In the examples this generation is very simple (setting the visibility of a fragment or the class of a link anchor). In general however the adaptive engine may have more difficult tasks, e. g. when the presentation of a page requires fragments to not only be selected but also sorted. Also, the engine is responsible for updating the user model after each event. The tasks performed by the adaptive engine when a user "follows a link" to a specifier S can be summarized as follows:

1. First the engine retrieves the stored user model. (The run-time layer which we do not elaborate in this paper ensures that through the "session" concept the identity of the user is known.) All stored attributes of all concepts are retrieved.

2. The other attributes that are used in pedagogical rules but that are not in the user model are initialized to a default value. Having default values reduces the number of rules that are required. For instance, if *ready-to-read* is not in the user-model but is used in rules, we can set the *ready-to-read* attribute for all concepts to *true* by default, and then only have rules to set it to *false* when needed.

3. The engine resolves the specifier S to a page concept C by applying pedagogical rules that are aimed at determining a "desired" page for specifier S, depending on the user model. (The rules use prerequisite, inhibitor and other similar concept relationships.)

4. The engine starts executing rules, starting with access(C) as the trigger. All triggered rules from the "pre" phase are executed. A page (or set of pages) is built, using the presentation specifications. (Fragments are selected and possibly sorted, for each page to be presented.)
5. The engine then does whatever is necessary to actually present the page(s). This action crosses the boundary between the "storage layer / presentation specifications" and the run-time layer. In a Web-based system this would also be the boundary between server and browser. The adaptive engine generates an HTML page and sends it to the browser.
6. Next all triggered rules from the "post" phase are executed. (These rules cannot change the presentation specification anymore in a meaningful way, because the presentation is already generated and handed over to the run-time layer.)
7. The updated values for attributes from the user model are saved in that user model.

An adaptive engine, as described above, can easily operate as a CGI-script or a servlet in a Web-based adaptive hypermedia application. In fact, most recent AHS, including Interbook [6] and AHA [9, 10], are based on CGI-scripts or servlets.

The fact that following a link may result in multiple destinations is convenient for modeling AHS which divide the display (or browser window) into different parts, each showing different aspects of the adaptive hypermedia application. A "kiosk" system and a course text realized in AHA [10] for instance use an adaptive *table of contents* frame and an *information* frame. (See http://wwwis.win.tue.nl/IShype/ and http://wwwis.win.tue.nl/2M350/.) Applications of Interbook [6] also include frames that show concepts that are learned or that are still to be learned. Each time a link is followed all frames need to be updated. While the node (page) that is displayed in a frame may remain the same, the presentation of that node may change, because different fragments may be (conditionally) included, and because link anchors may need to be annotated differently, hidden or maybe even removed.

By now we have defined all the components that make up an adaptive hypermedia application:

Definition 11 *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.

COMMUNICATION BETWEEN ADAPTIVE HYPERMEDIA SYSTEMS

The Dexter model assumes that all "history information" is limited to a single browsing or authoring session. It even states that when closing a session *by default, pending changes to instantiations are not saved* [16]. In AHAM we explicitly model a permanent user model, thus taking into account that a user's interaction with a hypermedia information source may span several sessions. The next step is of course to extend the user model to include a representation of evolution of the user's state of mind throughout his or her interaction with many adaptive hypermedia applications. The exchange of user models is one of the areas for which the IEEE Learning Technology Standards Committee (LTSC)(P1484) is trying to come up with a standard for.

Modeling the exchange of user model information in AHAM is as simple as adding two events, in addition to the "access" event:

- An AHS may offer an externally accessible function:

getValue(user, auth, cuid, attr)

where "user" identifies a user, "auth" is a system-dependent authorization, "cuid" is a unique concept identifier, and "attr" is an attribute of the user model. The function returns a value (of the appropriate type) for that attribute. The caller of the function needs to know which data type to expect.
Here it is convenient that the Dexter model, and thus also AHAM, requires that the unique identifiers for all components are globally unique, not just within a hypermedia application but unique within the entire universe of discourse.

- An AHS may offer an externally accessible procedure:

setValue(user, auth, cuid, attr, value)

where "user" identifies a user, "auth" is a system-dependent authorization, "cuid" is a unique concept identifier, "attr" is an attribute of the user model and "value" is the new value for this attribute. This procedure (or void function) updates the user model.

Authorization is needed for obvious reasons: not every external application can be allowed to read and/or update arbitrary user models.

It may seem that it is very inconvenient to have these functions take or return values of specific data types which may be different in every AHS. It is possible to translate many (but not nearly all) data types to a "standard" one, like all real (floating point) numbers between 0 and 1. However, the biggest problem in the communication between systems is not the technical data conversion but the semantic conversion. In order for an application to use the *knowledge-value* for a concept C which is imported from a different AHS, (the author of) that application must know what the concept C means in terms of its own concepts, and it must know what the *knowledge-value* means. If a system that uses values between 0 and 1 wishes to retrieve a knowledge-value from another system that returns *well learned*, the system needs to be able to interpret which of its own values has the same meaning as *well learned* in the other system. Even when two systems use knowledge values between 0 and 1 the identity may not be the most appropriate conversion of knowledge values.

An immediate application of the functions “getValue” and “setValue” is in the communication with semi-external applications like an evaluation tool that uses multiple-choice tests, or an initial questionnaire that is used to initialize a user model and to set preferences. Apart from a user’s knowledge about concepts, many AHS also want to store a user’s preferences, like a preferred media type (text, audio, video), a desired verbosity level (terse, medium, verbose), link annotation type (hiding, annotation, and if so, using which color scheme), etc. The system may also want to store other background information about the user, like experience in the use of adaptive hypermedia systems, possible disabilities, age, education, etc. While all this information is semantically very different from “knowledge about concepts” it can be represented and implemented as if it were just “knowledge about concepts”. AHAM is thus able to model preferences, background, and other user aspects one may wish to include in a user model.

FUTURE WORK ON AHAM

The description of AHAM in this paper focusses on browsing. The Dexter model includes functions in the storage layer as well as the run-time layer for creating, modifying and deleting components. Our main motivation for designing AHAM was to develop a framework to aid the development of adaptive hypermedia systems in which authoring would be much easier than in existing systems like Interbook [6] and AHA [9, 10].

The main hurdle in facilitating authoring is the clear separation of the design of the *concept space*, the *actual information content*, the *link structure* at a conceptual level, and the *dependencies* between concepts (like prerequisites and inhibitors). This design aspect of AHAM has been realized, and leads us to believe that adaptive hypermedia authoring tools that maintain this separation of concerns will be much easier to use than any of the tools that exist today.

What remains to be done is to include functions for actually creating (atomic and composite) concepts and concept relationships, for modifying and for deleting them in the definition of the AHAM model. This will be done in a forthcoming extended paper that will also include a more formal definition of AHAM, using the specification language Z that was used to formally define the Dexter model [15].

CONCLUSIONS

In this paper we have introduced a reference model for adaptive hypermedia applications. This model, called AHAM, was explained in terms of the well-known Dexter model. The architecture of AHAM extends Dexter by dividing the *storage layer* into three parts: a *domain model*, a *user model* and a *teaching model*.

We have focussed on the description of *browsing* in an adaptive hypermedia application. The introduction of a permanent and continuously updated user model enabled us to include a complete navigation history in the model (unlike in Dexter where records are only kept within a session), and to use that history to generate attributes of presentation specifications that result in a user-adapted presentation of both the information content and the links (or link anchors) of the application. Future work includes the description of *authoring* in the model.

ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers who, through their remarks and questions, have helped us to generalize some definitions, and who pointed out some issues that needed additional clarifications.

REFERENCES

1. I. Beaumont. User modeling in the interactive anatomy tutoring system ANATOM-TUTOR. User Modeling and User-Adapted Interaction, Vol. 4, pp. 21–45, Kluwer academic publishers, 1994.

2. C. Boyle and A. O. Encarnacion. MetaDoc: an adaptive hypertext reading system. User Modeling and User-Adapted Interaction, Vol. 4, pp. 1–19, Kluwer academic publishers, 1994.

3. P. Brusilovsky. Methods and Techniques of Adaptive Hypermedia. User Modeling and User-Adapted Interaction, Vol. 6, pp. 87–129, Kluwer academic publishers, 1996.

4. P. Brusilovsky and L. Pesin. ISIS-Tutor: An adaptive hypertext learning environment. JCKBSE’94, Japanese-CIS Symposium on knowledge-based software engineering, Pereslavl-Zalesski, Russia, pp. 83–87, 1994.

5. P. Brusilovsky, E. Schwarz and G. Weber. ELM-ART: An intelligent tutoring system on World Wide Web. Third International Conference on Intelligent Tutoring Systems, ITS-96, Montreal, LNCS Vol. 1086, pp. 261–269, 1996.

6. P. Brusilovsky, E. Schwarz and G. Weber. A Tool for Developing Adaptive Electronic Textbooks on WWW. Proc. WebNet’96 Conference, pp. 64–69, San Francisco, 1996.

7. L. Calvi and P. De Bra. Improving the Usability of Hypertext Courseware through Adaptive Linking. Proc. 8th ACM Conference on Hypertext, Southampton, pp. 224–225, 1997.

8. L. Calvi and P. De Bra. Using dynamic hypertext to create multi-purpose textbooks. Proc. ED-MEDIA’97, Calgary, pp. 130–135, 1997.

9. P. De Bra and L. Calvi. Creating adaptive hyperdocuments for and on the Web. Proc. WebNet’97 Conference, Toronto, pp. 149–155, 1997.

10. P. De Bra and L. Calvi. Towards a Generic Adaptive Hypermedia System. Proc. Second Workshop on Adaptive Hypertext and Hypermedia, Pittsburgh, pp. 5–11, 1998.

11. P. De Bra, G. J. Houben, Y. Kornatzky. An Extensible Data Model for Hyperdocuments. Proc. 4th ACM Conference on Hypertext, Milan, pp. 222–231, 1992.

12. R. Furuta and P. D. Stotts. The Trellis Hypertext Reference Model. In *Proc. NIST Hypertext Standardization Workshop*, pp. 83–93, 1990.

13. F. Garzotto, P. Paolini, D. Schwabe. HDM - A model-based approach to hypermedia application design. ACM Transactions on Information Systems, 11:1, pp. 1–23, 1993.

14. K. Grønbæk, R. Trigg. Design issues for a Dexter-based hypermedia system. Communications of the ACM, Vol. 37, nr. 2, pp. 40–49, 1994.

15. F. Halasz and M. Schwartz. The Dexter Reference Model. In *Proc. NIST Hypertext Standardization Workshop*, pp. 95–133, 1990.

16. F. Halasz and M. Schwartz. The Dexter Hypertext Reference Model. Communications of the ACM, Vol. 37, nr. 2, pp. 30–39, 1994.

17. J. Hothi and W. Hall. An Evaluation of Adapted Hypermedia Techniques Using Static User Modelling. Proc. Second Workshop on Adaptive Hypertext and Hypermedia, Pittsburgh, pp. 45–50, 1998.

18. D. Lange. A Formal Model of Hypertext. In *Proc. NIST Hypertext Standardization Workshop*, pp. 145–166, 1990.

19. A. Kobsa, D. Müller and A. Nill. KN-AHS: An adaptive hypertext client of the user modeling system BGP-MS. Proc. Fourth International Conference on User Modeling, Hyannis, MA, pp. 31–36, 1994.

20. N. Mathé and J. Chen. A user-centered approach to adaptive hypertext based on an information relevance model. Proc. Fourth International Conference on User Modeling, Hyannis, MA, pp. 107–114, 1994.

21. M. Milosavljevic and J. Oberlander. Dynamic Hypertext Catalogues: Helping Users to Help Themselves. Proc. Ninth ACM Conference on Hypertext and Hypermedia, Pittsburgh, PA, pp. 123–131.

22. D. Pilar da Silva. Concepts and documents for adaptive educational hypermedia: a model and a prototype. Proc. Second Workshop on Adaptive Hypertext and Hypermedia, Pittsburgh, pp. 33–40, 1998.

23. D. Schwabe, G. Rossi. The Object-Oriented Hypermedia Design Model. Communications of the ACM, Vol. 38, nr. 8, pp. 45–46, 1995.