Adaptation Control in Adaptive Hypermedia Systems

Hongjing Wu, Paul De Bra, Ad Aerts, Geert-Jan Houben

Department of Computing Science
Eindhoven University of Technology
PO Box 513, 5600 MB Eindhoven
the Netherlands
phone: +31 40 2472733
fax: +31 40 2463992
email: {hongjing,debra,wsinatma,houben}@win.tue.nl

Abstract. A hypermedia application offers its users a lot of freedom to navigate through a large hyperspace, described by a *domain model*. Adaptive hypermedia systems (AHS) aim at overcoming possible navigation and comprehension problems by providing adaptive navigation support and adaptive content. The adaptation is based on a *user model* that represents relevant aspects about the user. In this paper, we concentrate on the *adaptation engine* (AE) that is responsible for performing the adaptation according to the *adaptation rules* specified in the *adaptation model*. We analyze the dependencies between the authoring process and the functionality of the adaptation engine. From this we conclude how the authoring process can be simplified by a more powerful AE. In particular, a well-designed AE should be *general purpose* (i.e., not application domain specific) and should guarantee that the interpretation of the rules is deterministic, always terminates and produces the results desired by the author.

Keywords: adaptive hypermedia, user modeling, adaptive presentation, adaptive navigation, hypermedia reference model, adaptation rules.

1. Introduction

Hypermedia systems (including the Web) are becoming increasingly popular as tools for user-driven access to information. They typically offer users a lot of freedom to navigate through a large hyperspace. Unfortunately, this rich link structure of the hypermedia applications causes some serious usability problems:

- A typical hypermedia system presents the same links on a page to all users. To eliminate *navigation problems* the system should offer each user (some) personalized links or navigation tools (such as a table of contents or a map). The system should thereby take into account what the user read before, and possibly what the user's interests are.
- Navigation in ways the author did not anticipate also causes *comprehension problems* for the user: for every page the author makes an assumption about what foreknowledge the user has when accessing that page. However, this is an impossible authoring task because there are more ways to reach a page than any (human) author can foresee. A page is always presented in the same way. This may result in

users visiting pages containing redundant information and pages that they cannot fully understand because they lack some expected foreknowledge.

Adaptive hypermedia systems (or AHS for short) aim at overcoming these problems by providing *adaptive navigation support* and *adaptive content*. The adaptation (or personalization) is based on a *user model* that represents relevant aspects of the user such as preferences, knowledge and interests. The system gathers information about the user by observing the use of the application, and in particular by observing the *browsing* behavior of the user.

Many adaptive hypermedia systems exist to date. The majority of them are used in educational applications, but some are used, for example, for on-line information systems or information retrieval systems. An overview of systems, methods and techniques for adaptive hypermedia can be found in [B96]. We have developed a reference model for the architecture of adaptive hypermedia applications: AHAM (for **A**daptive **H**ypermedia **A**pplication **M**odel) [DHW99], which is an extension of the Dexter hypermedia reference model [HS90, HS94]. AHAM acknowledges that doing "useful" and "usable" adaptation in a given application depends on three factors:

- The application must be based on a *domain model*, describing how the information content of the application or "hyper-document" is structured (using concept).
- The system must construct and maintain a fine-grained *user model* that represents a user's preferences, knowledge, goals, navigation history and other relevant aspects.
- The system must be able to adapt the presentation (of both content and link structure) to the reading and navigation style the user prefers and to the user's knowledge level. In order to do so the author must provide an *adaptation model* consisting of *adaptation rules*. An AHS itself may offer built-in rules for common adaptation aspects. This reduces the author's task of providing such rules.

The division into a *domain model* (DM), *user model* (UM) and *adaptation model* (AM) provides a clear separation of concerns when developing an adaptive hypermedia application. The main shortcoming in many current AHS is that these three factors or components are not clearly separated [WHD00].

In this paper we focus on the *adaptation engine* (AE) that provides the implementation dependent aspects of AHAM. We divide the adaptive control in AHS into two levels, the author level and the system level. On the author level an author writes *adaptation rules*; on the system level the system designers build an *adaptation engine* (AE) to apply the rules. These two parts work together to control the adaptation in the AHS. In this paper we consider three main design goals for the AE:

- Authoring should be simplified as much as possible. The author should not have to include in the adaptation rules any aspects that a (smart) AE can handle automatically. The author should take care of the domain dependent aspects, and the AE should take care of the domain independent aspects.
- The interpretation (or execution) of adaptation rules by the AE should always terminate. Furthermore, the adaptation should not cause noticeable delays.
- The interpretation of the adaptation rules by the AE should be deterministic.

This paper is organized as follows. In Section 2 we briefly recall the AHAM reference model for adaptive hypermedia applications and propose two general constraints on (generic) adaptation rules. In Section 3 we define some terms and discuss system transition issues. In Section 4 we discuss termination and determinism of the AE, and ways to make authoring easier by making the AE smarter. Section 5 presents our conclusions and short-term research agenda.

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 [HS90,HS94] 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 (and content). 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 needs to specify how the user's knowledge influences the presentation of the information from DM. In AHAM [DHW99], this is done by means of an *adaptation model* (AM) consisting of *adaptation rules*. ([DHW99] uses slightly different terms.) An adaptation engine (AE) uses these rules to manipulate link anchors (from the Dexter model's *anchoring*) and to generate what the Dexter model calls the *presentation specifications*. In this section we only present the elements of AHAM that we will need in the following sections when we discuss the implementation aspects of the AHAM.

2.1 The domain model

The domain model of an adaptive hypermedia application consists of *concepts* and *concept relationships*. Concepts are objects with a unique object identifier, and a structure that includes attribute-value pairs and link anchors. (The remainder of the structure is not relevant for this paper.)

A *concept* represents 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. It is primitive in the model (and can thus not be adapted). Its attribute and anchor values belong to the "Within-component layer" and are thus implementation dependent and not described in the model.
- A *composite concept* has a sequence of children (sub-concepts) and a constructor function that describes how the children belong together. The children of a composite concept are either all atomic concepts or all composite concepts. A composite concept with (only) atomic children is called a *page*.

The composite concept hierarchy must be a DAG (directed acyclic graph). Also, every atomic concept must be included in some composite concept. Figure 1 illustrates a part of a concept hierarchy.

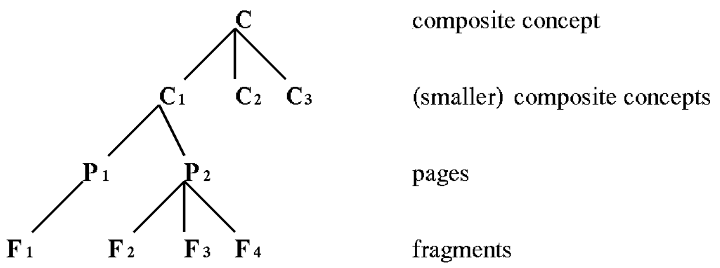

Fig. 1. : Part of a concept hierarchy.

A *concept relationship* is an object (with a unique identifier and attribute-value pairs) that relates a sequence of two or more concepts. Each concept relationship has a type. The most common type is the hypertext **link**. In AHAM we consider other types of relationships as well, which play a role in the adaptation, e.g. the type **prerequisite**. When a concept C_1 is a prerequisite for C_2 it means that the user should read C_1 before C_2. It does not mean that there must be a link from C_1 to C_2. It only means that the system somehow takes into account that reading about C_2 is not desired before some (enough) knowledge about C_1 has been acquired. Through link adaptation the "desirability" of a link will be made clear to the user.

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

2.2 The user model

A user model consists of named entities for which we store a number of attribute-value pairs. For each user the AHS maintains a *table-like structure*, in which for each concept in the DM the attribute values for that concept are stored. Because of the relationships between *abstract* concepts and *concrete* content elements like fragments and pages, a user model may contain other attributes than simply a *knowledge level* (typical in educational applications). For instance, the user model may also store information about what a user has actually read about a concept or whether a concept is considered relevant for the user. Concepts can furthermore be used (some might say abused) to represent other user aspects such as preferences, goals, background, hyperspace experience, or a (stereotypical) classification like student, employee, visitor, etc. For the AHS or the AHAM model the true meaning of concepts is irrelevant.

In the sequel we will always consider UM as being the user model for a single user. In this paper we do not discuss adaptation to group behavior.

2.3 The adaptation (teaching) model

The adaptation of the information content of a hyper-document and of the link structure is based on a set of *adaptation rules*. A rule is typically of the form:

if <condition> then <action>.

Here condition may specify the occurrence of an external event, such as "page access" (or "click" for short), conjugated with Boolean expressions referring to attribute-values from the DM or UM. The action may be an update to an attribute-value in UM, or be an assignment to a presentation specification for an object: it may specify that a fragment will be shown, or that a link should be treated as being desirable. For example, consider the rule:

if access(C) and F IN C.children and F.relevant = true then F.pres := show.

This rule specifies that when a page concept C is accessed (by clicking on a link to this page) and that page contains a fragment F that is relevant for the user, then that fragment should become visible in the presentation of the page. The consequence of this rule is that all relevant fragments of this page will be shown to the user. Instead of hiding undesired fragments the AHS may gray them out, as described in [HH98].

Adaptation rules form the connection between DM, UM and the presentation (specification) to be generated, and their syntax is AHS-dependent. In fact, in many AHS a number of rules will be hard-coded into the system and not visible or accessible to authors. We partition the rules into four groups according to the adaptation "steps" to which they belong. These steps are IU (**i**nitialize **u**ser model), UU-Pre (**u**pdate **u**ser model before generating the page), GA (**g**enerate **a**daptation), and UU-Post **(u**pdate **u**ser model after generating the page). The rules in these groups are applied in the order specified. [WHD99]

In a *generic adaptation rule* (bound) variables are used that represent concepts and concept relationships. A *specific adaptation rule* uses concrete concepts from DM instead of variables. Other than that both types of rules look the same. Specific rules always take precedence over generic ones. The syntax of the permissible rules depends on the AHS and is irrelevant for this paper.

The *adaptation model* AM of an AHS is the set of (generic and specific) adaptation rules.

2.4 The adaptation engine

An AHS does not only have a domain model, user model and adaptation model, but also an *adaptation engine*, which is a software environment that performs the following functions:

- It offers generic page selectors and constructors. For each composite concept the constructor is used to determine which page to display when the user follows a link to that composite concept. For each page the constructor is used for building the adaptive presentation of that page.
- It optionally offers a (very simple programming) language for describing new page selectors and constructors. For instance, in AHA [DC98] a page constructor consists of simple commands for the conditional inclusion of fragments.
- It performs adaptation by executing the page selectors and constructors. This means selecting a page, selecting fragments, organizing and presenting them in a specific way, etc. It also means performing adaptation to links by manipulating link anchors depending on the state of the link (like enabled, disabled and hidden.).

- It updates the user model (instance) each time the user visits a page. It does so by triggering the necessary adaptation rules in UU-pre or UU-post. The engine will thus set some attribute values for each atomic concept of displayed fragments of a page, of the page as a whole and possibly of some other (composite) concepts as well (all depending on the adaptation rules).

The adaptation engine thus provides the implementation-dependent aspects, while DM, UM, and AM describe the information and adaptation at the conceptual, implementation independent level. An *adaptive hypermedia application* is a 4-tuple (DM, UM, AM, AE), where DM is a domain model, UM is a user model, AM is an adaptation model, and AE is an adaptation engine. The challenge, of course, is to design an AE that is not specific to a particular application, but that can handle a wide range of hypermedia applications.

2.5 General constraints

Performance is a frequently neglected aspect of adaptive hypermedia applications. However, when the execution of the adaptation rules takes a long time there will be a very noticeable delay between the click on a link anchor and the appearance of the link destination page. An easy way to ensure fast response is to require all adaptation rules to be mutually independent. This means that all rules are triggered by an event (such as a "click") and the result of a rule cannot be a trigger for other rules. This is a severe restriction. For instance, it does not allow "knowledge" to be propagated bottom-up through the concept hierarchy automatically.

In this paper we explicitly allow cascades of rule executions. The purpose is to lighten the burden on the author of the adaptive application. For instance, we allow an author to specify that in Figure 1 reading page P_1 or P_2 contributes 50% towards the knowledge of C_1 and that C_1 contributes e.g. 40% towards the knowledge of C. When the user accesses P_1 an adaptation rule will set the knowledge of P_1 to 100%. This rule will trigger another rule (instance) that adds 50% to the knowledge of C_1 and this will trigger yet another rule (instance) that will add 20% (or 40% of 50%) to the knowledge of C. The (current version of the) AHA system for instance allows such cascades of rules. It will be obvious that this recursive definition of user model updates is easier for an author than specifying explicitly that reading P_1 contributes 50% to C_1 and 20% to C, and that reading P_2 also contributes 50% to C_1 and also 20% to C. (It is easier when the author need not explicitly specify directly for each page how much reading that page contributes towards the knowledge of C.)

While we allow cascading rule execution we do propose the following restrictions on the propagation between rules:

- If the <condition> and <action> of a rule contain an attribute of the same concept then the <condition> must include an *event*. The propagation of attribute values within one concept and without an event indicates bad design and is forbidden.
- If the <condition> and <action> of a *generic* adaptation rule contain different concepts, then the <condition> must also include a *concept relationship* linking these concepts. Thus, propagation of user model updates between different concepts is only allowed through concept relationships.

An example of a rule that illustrates and satisfies the first constraint (for page P) is:

if *access*(P) and P.ready-to-read = true then P.knowledge := 'known'.

The update to the knowledge (attribute) of P depends on the ready-to-read attribute of the same P, and therefore must be triggered by an event such as *access*(P).
An example of a rule that illustrates the second constraint is:

if ∀P’, prerequisite(P’, P): P’.knowledge = ‘known’ then P.ready-to-read := true

3. System transitions

In this section we describe how an AHS works using triggers and rules. However, this does not imply that only rule and trigger based AHS’s can be described in AHAM. Triggers and rules are just our means for describing the system behavior.

An *event* in the system means that something outside the system triggers the system to change its state. The user or external programs can only observe the following:

$$\mathbf{UM_s} \xrightarrow{\text{event}} \mathbf{UM_f}$$

Here UM_s and UM_f are two states of the system. In general these states must differ in at least one attribute value. UM_s is the *start-state* and UM_f is the *final-state*. “Inside” the AHS the transition is, for instance, realized by sequentially executing a number of rules:

$$\mathbf{UM_s} \xrightarrow{R_1} \mathbf{UM_2} \xrightarrow{R_2} \mathbf{\ldots}\ \mathbf{UM_i} \xrightarrow{R_i} \mathbf{\ldots} \xrightarrow{R_m} \mathbf{UM_f}$$

Here R_i is an instance of a generic or a specific rule, i is in [1..m], m must be finite. Thus, internally the system applies a finite sequence of rules (or actually rule instances) to arrive at UM_f. Each step in this transition is called an *update*. When an event occurs, it triggers some rules that deal with that event. These rules will change some values in the UM, and these changes will propagate to other rules. The order in which rules are applied is called the *execution order*.

When a rule’s condition changes from false to true, it becomes an *active rule*. The system only executes active rules in the transition. After execution of an active rule, that rule becomes inactive. The execution of a rule may make other inactive rules become active. If there is no active rule anymore, then the transition *terminates*. In Section 4.1 we will consider the detection of potential infinite loops that would cause a transition to never terminate.

Apart from termination of a transition, also the predictability of its final state is an issue. If the execution order of the rules doesn’t affect the final state, we call the transition an *order independent transition* (OIT). If the execution order does affect the final state, we call this transition an *order dependent transition* (ODT). Order dependent transitions are not desirable because they make the result (the final state) unpredictable. Order-dependence arises, when there is a choice as to which rule to execute first. For instance, an event may activate more than one rule. In the sequential procedure presented above, we select one rule and execute it. This will produce a new (intermediate) state of the UM in which some other rules may have become active. In the course of the process of rule execution more rules may become active, but also some active rules may become inactive because other rule executions have made their condition become false again. The final result may also depend on the order in which subsequent updates of the same concept attribute take place. To provide a deterministic AE behavior, we have to resolve the order dependence. This issue is dealt with in Section 4.2.

4. Issues in building the AE

As indicated in the introduction, we consider three design goals. We try to make the burden on the author lighter by providing *simple rules*, while at the same time providing an adaptation engine that generates *predictable* results and that executes rules in such a way that *termination* is guaranteed.

4.1 Termination

There are different ways to ensure that a transition terminates. One way is to write rules in such a way that infinite (triggering) loops are impossible and that long (deep) recursions are avoided. In order to ensure that the rules are written in this way one needs the authoring tool to check for potential (direct or indirect) loops each time an author adds rules to the AM. The added rule can either be rejected or the system may simply warn the author and give advice on how to break the loop in the rules. Whether a rule may lead to infinite loops also depends on whether the concept relationships used in the rule are allowed to be cyclic or not. E.g., link relationships can be cyclic, whereas prerequisite relationships must not be cyclic.

Another way is to have an adaptation engine (AE) that ensures that the transition will stop after some time, even when the rule definitions (together with cycles in DM) cause loops. This approach is easier for the author, because the author need not know or be informed about (potential) loops. There are three easy (and also simplistic) ways to make an AE ensure termination of transitions:

1. The system changes the attribute value of a concept at most once in one transition. Because UM has only a finite number of concepts and each concept a finite number of attributes, the number of possible update steps that do not change a previously updated attribute value is finite. Unfortunately this method inhibits some possibly interesting ways to update the UM. For example, if concept A contributes knowledge towards B and C, and B and C both contribute (a possibly different amount of) knowledge towards D, D's knowledge value can only be updated in a predictable way if knowledge propagation from B and from C are both allowed to happen during one transition. One can easily come up with similar examples using concept relationships instead of the concept hierarchy, and also leading to "premature" termination of parts of a transition.
2. Each rule instance (either an instance of a generic rule or a specific rule) is executed at most once in one transition. A generic rule may be used several times, but with different bindings of its (concept) variables to actual concepts. This method again guarantees termination. In the above example the knowledge value of D can be updated twice because both updates are different rule instances. Unfortunately, if in this example D also contributes knowledge to E, the resulting two knowledge contributions cannot both be propagated to E because that would be done through the same rule instance.

3. The AE can make use of properties of the value domain for each attribute (of concepts) to determine whether repeated updates to a concept or repeated execution of the same rule instance are potential sources of infinite loops. The AE of the AHA system for instance allows repeated monotonic updates to a concept's knowledge value. This poses no danger because the value domain consists of integers between 0 and 100. (All monotonic update loops terminate when the value reaches 100 or 0, depending on whether the value monotonically increases or decreases.)

The first two methods can be modified so that they allow not one but a larger (fixed or variable) number of updates or instances of rules to be executed. This may eliminate some of the negative side effects, but it also slows the AE down in case of an infinite loop that must be terminated. The third method is preferable, but for some value domains it may be difficult to come up with a property that provides a good basis for terminating potential infinite loops. More research on termination is needed, but research on active databases has already shown that termination quickly becomes undecidable.

4.2 Determinism

A single event may activate several rules, and each rule execution may activate some more rules. In the sequential model we presented in Section 3, active rules are executed one by one. The order in which rule instances are executed may influence the final result UM_f. Also, when a potential infinite loop is cut short by one of the methods described in Section 4.1, the order in which rules are executed may influence which rules are executed and which rules are discarded. The problem of non-deterministic results is a direct consequence of our desire to make authoring easier. The easiest way to guarantee deterministic behavior of AE is to define the rule language in such a way that the author has to indicate *when* a rule must be executed, and not only *under which conditions* a rule may be executed. We have chosen an intermediate approach: the author must assign rules to the four categories IU, UU-pre, GA and UU-post, but within each category the author does not indicate execution order.

While we do not have a general solution, an AE can find out potential sources of non-determinism by searching for conflicting user model updates that result from active rules. For instance, consider the following three rules:

if access(A) and A.ready-to-read = true then A.knowledge := 'known'.
if A.knowledge = 'known' and prerequisite(A,B) then B.ready-to-read := true.
if A.knowledge = 'known' and inhibit(A,B) then B.ready-to-read := false.

What these rules actually imply is that there cannot simultaneously be a prerequisite and an inhibit relationship between A and B. If these relationships would exist however then whether B.ready-to-read becomes true or false depends on the order in which the rules are executed. A smart authoring tool can detect that the rules have a conflicting outcome, and thus warn the author of the error in either the given relationships or the supplied rules. There may be cases where conflicting updates are not easily detected during the authoring phase. Such undesirable, ambiguous situations can still be detected by the AE, by not simply executing rules but by first examining all active rules and checking them for a conflicting outcome. What is needed then is a conflict resolution strategy that must be specified by the author or the system and that can be used to eliminate the conflict. Once all conflicts between the active rules have been resolved, the order of execution is not important any more and the transition terminates in a well-determined state. In the case of the prerequisite and inhibit relationships one might for instance state that prerequisites take precedence and that in case of conflict the rule for the inhibit relationship is not executed. In the case of a conflicting generic and specific rule, the specific rule always has precedence.

5. Conclusion and future work

We have analyzed several ways to build an AE that is deterministic and produces results in an acceptable number of steps. A viable approach is to assign adaptation rules to groups and specify some general precedence relationships, which will constitute the AE default behavior. This will leave a number of situations in which the author has to provide some mechanism of choice for the order dependent transitions. An attractive option appears to be the collective application of all rules that are active in a particular state and provide a conflict resolution strategy. This way, we can build general AE's that provide a clear separation of responsibilities between the system and the author. The input of the author will, of course, always be required and is best put in the form of overruling general AE behavior with specific, domain dependent choices.

The next research step is to experiment with the various design alternatives for the AE, and to design an appropriate rule language for the author to write the application. The rule language should provide a general way to specify rules for desirable application behavior.

References

[B96] Brusilovsky, P., “Methods and Techniques of Adaptive Hypermedia”. User Modeling and User-Adapted Interaction, 6, pp. 87-129, 1996. (Reprinted in Adaptive Hypertext and Hypermedia, Kluwer Academic Publishers, pp. 1-43, 1998.)

[DC98] De Bra, P., Calvi, L., “AHA! An open Adaptive Hypermedia Architecture”. The New Review of Hypermedia and Multimedia, pp. 115-139, 1998.

[DHW99] De Bra, P., Houben, G.J., Wu, H., “AHAM: A Dexter-based Reference Model for Adaptive Hypermedia”. Proceedings of ACM Hypertext’99, Darmstadt, pp. 147-156, 1999.

[HS90] Halasz, F., Schwartz, M., “The Dexter Reference Model”. Proceedings of the NIST Hypertext Standardization Workshop, pp. 95-133, 1990.

[HS94] Halasz, F., Schwartz, M., “The Dexter Hypertext Reference Model”. Communications of the ACM, Vol. 37, nr. 2, pp. 30-39, 1994.

[HH98] Hothi, J., Hall, W., “An Evaluation of Adapted Hypermedia Techniques Using Static User Modeling”, Proceedings of the Second Workshop on Adaptive Hypertext and Hypermedia, pp. 45-50, 1998.

[PDS99] Pilar da Silva, D., “Concepts and documents for adaptive educational hypermedia: a model and a prototype”, Proceedings of the Second Workshop on Adaptive Hypertext and Hypermedia, Pittsburgh, pp. 33-40, 1998.

[WHD99] Wu, H., Houben, G.J., De Bra, P., “Authoring Support for Adaptive Hypermedia”, Proceedings ED-MEDIA’99, Seattle, pp. 364-369, 1999.

[WHD00] Wu, H., Houben, G.J., De Bra, P., “Supporting User Adaptation in Adaptive Hypermedia Applications”, Proceedings InfWet2000. Rotterdam, the Netherlands.