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. Adaptive hypermedia systems (or AHS for short) 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.

We have previously developed a reference model for the architecture of adaptive hypermedia applications, named AHAM (for Adaptive Hypermedia Application Model) [DHW99]. AHAM divides the different aspects of an AHS into a *domain model* (DM), a *user model* (UM) and an *adaptation model* (AM). This division provides a clear separation of concerns when developing an adaptive hypermedia application at the conceptual level. The author writes the adaptation rules in the AM.

In this paper, we concentrate on the *adaptation engine* (AE) that is responsible for performing the adaptation according to the *adaptation rules* of the *adaptation model*. We analyze the dependency of the authoring process and the adaptation engine. We then describe how the authoring process for adaptation can be simplified by means of a proper design of the 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, and Web-based systems in particular, are becoming increasingly popular as tools for user-driven access to information. Hypermedia applications 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, regardless of the path a user has followed to reach this page. Even when providing navigational help, e.g. through a map (or some fish-eye view) the system does not know which part of the link structure is most important for the user. The map can thus not be simplified by filtering (or graying out) links that are less relevant for the user. Having personalized links or maps would eliminate some *navigation problems* that users have with hypermedia applications.
- Navigation in ways the author did not anticipate also causes *comprehension problems* for the user: for every page the author makes an assumption about the foreknowledge the user has when accessing that page. However, there are too many ways to reach a page to make it possible for an author to anticipate all possible variations in foreknowledge when a user visits that page. A page is always presented in the same way. This often results in users visiting pages containing a lot of 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. Adaptive hypermedia is a recent area of research on the crossroad of hypermedia and the area of user-adaptive systems. The goal of this research is to improve the usability of hypermedia systems by making them personalized. The personalization or adaptation is based on a *user model* that represents relevant aspects about the user. 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, on-line help 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.
- The system must construct and maintain a fine-grained *user model* that represents a user's preferences, knowledge, goals, navigation history and possibly 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 to the application. 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 adaptation aspects in the adaptation rules when the AE can be made to handle these aspects automatically. This requires a division of tasks between the author, who takes care of the domain dependent aspects, and the AE, which takes care of the domain independent aspects.
- The interpretation (or execution) of adaptation rules by the AE should always terminate.
- The interpretation of the adaptation rules by the AE should be deterministic (or predictable).

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. In section 5 we discuss usability issues, affecting the author of a hypermedia application. Section 6 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. 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 a *teaching model* TM consisting of *pedagogical rules*. In this paper we use the terms *adaptation model* (AM) and *adaptation rules* to avoid the association with educational applications. 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*. 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 the AHAM model.

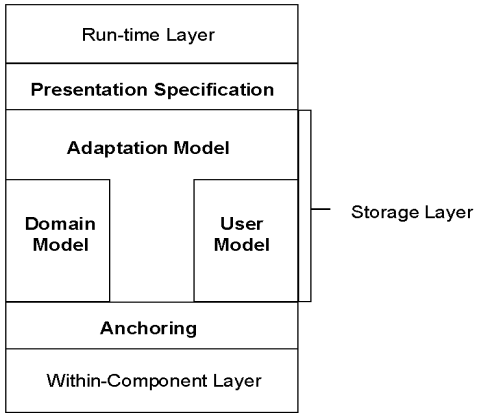

Figure 1: General structure of adaptive hypermedia applications.

In this section we 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 the following structure:

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

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 two "special" attributes:

- A sequence of children (sub-concepts);
- A constructor function (to denote 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 2 illustrates a part of a concept hierarchy.

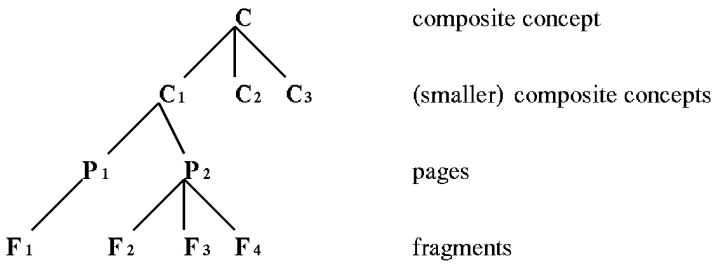

Figure 2: Part of a concept hierarchy.

Anchors are used to indicate endpoints of links. An anchor has a unique identifier and a value. A more detailed description of anchors is not relevant for this paper.
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* has the following structure:

- A unique object identifier;
- A set of attribute-value pairs;
- A sequence of specifiers;
- A concept relationship type (crt);
- A presentation specification.

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. (Links typically have at least one FROM element and one TO or BIDIRECT element.) In AHAM we consider other types of relationships as well, which play a role in the adaptation. A common type of concept relationship is **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. Every prerequisite must have at least one FROM element and one TO element. Figure 3 shows a small set of (only binary) concepts associated to one another by three types of concept relationship: prerequisite, inhibit, and link.

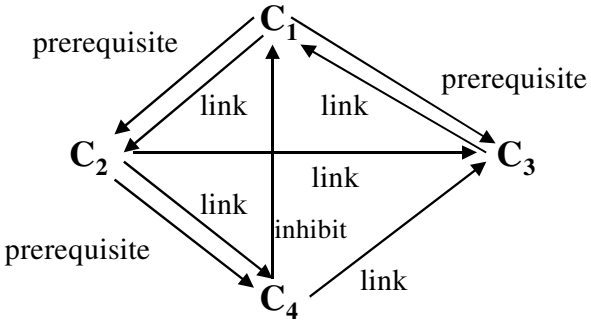

Figure 3: Example concept relationship structure.

The atomic concepts, composite concepts and concept relationships together form the *domain model* DM of an adaptive hypermedia application. We can also describe the DM as a pair (DM-concept, DM-cr-graph). *DM-concept* presents a set of concepts, and *DM-cr-graph* is a set of concept relationship graphs. (Figure 3 contains two concept relationship graphs: the prerequisite-graph and the link-graph.)

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. 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 features, such as preferences, goals, background, hyperspace experience, or a (stereotypical) classification like student, employee, visitor, etc. For the AHS (or the AHAM model) the actual meaning of concepts is irrelevant.

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 is usually an update to an attribute-value. For example, consider the rule:

if access(C) and F IN C.children and F.relevance = 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 of relevance to 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. Alternatively, an AHS may decide to not hide undesired fragments completely but to 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. We partition the rules into four groups according to the adaptation "steps" to which they belong. These steps are IU, UU-Pre, GA, and UU-Post. The rules in these groups are applied in the order specified. The rules in IU are used to **i**nitialize the **u**ser model for a particular user and are applied only once. The rules in UU-Pre specify the **u**pdates that are applied to the **u**ser model before generation of the page. GA contains the rules that apply to the **g**eneration of the **a**daptation, and UU-Post contains the rules for **u**pdating the **u**ser model after generating the page [WHD99].

We see from the previous paragraph that rules have information associated with them such as the step that they are assigned to. This information can be expressed in terms of attribute/value pairs as well.

A *generic adaptation rule* is a rule in which (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. The syntax of the permissible rules depends on the AHS. Generic adaptation rules are often system-defined, meaning that an author need not specify them. Such a rule may for instance define how the knowledge level of an arbitrary concept C_1 influences the relevance of other concepts for which C_1 is a prerequisite. Author-defined rules always take precedence over (conflicting) system-defined rules. (Some AHS do not provide the possibility for authors to define their own generic adaptation rules.) Specific rules always take precedence over generic ones.
While specific rules are typically used to create exceptions to generic rules they can also be used to perform some ad-hoc adaptation based on concepts for which DM does not provide a relationship. Only the author may define specific adaptation rules, whereas generic rules may be defined by the author or by the system designer).

The *adaptation model* AM of an AHS is the set of (generic and specific) adaptation rules.
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. Generic and specific adaptation rules (from UU-pre and GA) are used during page selection and construction.
- It performs adaptation by executing the page selectors and constructors. This means selecting a page, selecting fragments, sorting them, maybe 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-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.

2.4 General constraints

User acceptance of an adaptive hypermedia application depends on such properties as the timely and predictable response of the application to a user action. In order to meet the requirements on the application, a mutual dependence will arise between the rules that can be specified by an author and the adaptation engine that has to process them. For example, some early AHS had simple adaptation engines. The effect of a user action was laid down in a number of mutually independent rules. Following a particular user action the system could very easily detect which rules to apply. Applicable rules were executed, a new page was generated for the user, and the application was ready for the next user action. This combination of rules and rule processing resulted in systems that were acceptable in terms of response time.
Mutual independence is a severe restriction on rules. It forbids one rule to activate another one and requires the author to specify all updates to the user model in detail. This poses limitations on the kind of rules that can be used, and it places a burden on the author of the application. In this paper we will relax the restrictions on the rules, allowing cascades of rule executions. The purpose is to lighten the burden on the author. Relaxation of the restrictions on the rules will complicate, of course, the adaptation engine. Our purpose is to strike a balance between these two issues, while still meeting user requirements on the application. We propose the following two restrictions to control propagation between rules (note the different notation from that of the rules):

- $C.attribute_i \xrightarrow{event} C.attribute_j$.
- $C_m.attribute_i \xrightarrow{cr} C_n.attribute_j$, for $m \neq n$, and cr is a concept relationship in the DM-cr-graph.

The first constraint applies to the different attributes of the same concept. It means that if the <condition> and the <action> of a rule contain an attribute of the same concept, then the <condition> must also include an *event*. The constraint forbids the random propagation of changes to attribute values within one concept. The rationale for this constraint is that such attributes would not be independent and a signal of a bad design. An example of a rule that satisfies the first constraint is (P is some page concept):

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

The second constraint concerns attributes of different concepts. It means that the propagation of a change in $attribute_i$ of one concept to a change in $attribute_j$ of another concept is allowed only if there exists a concept relationship *cr* between these two concepts. An example of a rule that is allowed when P≠P1:

if P.knowledge = 'known' and prerequisite(P, P1) then P1.ready-to-read := true

The system may use the relationships in DM-cr-graph to apply the above generic rule automatically. The system also allows the author to define a new relationship type to create some specific relationships between concepts in order to control the propagation of updates between these concepts. An author can then use a concept relationship type to control the update order for a given set of concepts, e.g. the 'prerequisite' graph can be used to guide the propagation of updates between concepts subject to a 'prerequisite'-relationship, as illustrated in the rule above. When the cr-graph is acyclic, the propagation terminates.

3. System transitions

In the sequel we will simply consider only a single user. So when we say that "the" *state* of the system is the UM at that moment, we are considering only the user model for a single user.
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:

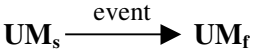

Figure 4: User view of a transition

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:

Figure 5: Internal view of a transition

Here R_i is an instance of a generic rule, or a specific rule, i is in [1..m], m is finite. $UM_i \neq UM_j$, for $i \neq j$.

Thus, internally the system applies a finite sequence of rules to arrive at UM_f. Each step in this transition is called an *update*. When an event happens, it will trigger some rules that deal with that event. These rules will change some values in the UM, and these changes will propagate through other rules. The order in which rules are applied is called the *execution order*. We require that no subsequence of the transition produce a null effect. (If $UM_i = UM_j$, for some $i < j$ then the whole subsequence between UM_i and UM_j could be eliminated, but could also be repeated indefinitely and thus forms a potential infinite loop.)

When a rule's condition becomes true, it becomes an *active rule*. The system only executes active rules in the transition. After execution of an active rule, it becomes inactive again. Should a rule be allowed to generate an update such that the rule's condition remains true after the update, then there is a potential for an infinite loop. The execution of a rule may make other inactive rules become active. If there are two rules such that each rule's action results in the other rule's condition to become true then again we potentially have an infinite loop. A similar thing could happen for a larger set of rules. We give some hints below on how to avoid such potential loops. If there is no active rule anymore, then the transition *terminates*.
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 rules may become inactive that would have been executed when they had been selected earlier. The final result may depend on this. It 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. One way to do this is to indicate precedence relations between rules. For instance, we could require the event rules to be executed first and the other (propagation) rules afterwards. We will discuss this option, which is bases on conflict *prevention*, and an alternative, based on conflict *detection*, below.

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 more powerful rules, while at the same time providing an adaptation engine that finishes its work on time and with a predictable result.

4.1 Termination

In every practical system termination is always associated with time. When the user clicks on a link a transition starts, and after that transition terminates a new page is sent to the user's browser. So, not only infinite loops are not acceptable, but also very long (finite) transition times are not acceptable. There are two ways to ensure that a transition terminates (sufficiently quickly):

- 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.
- Another way is to have an adaptation engine (AE) that ensures that the transition will stop after some time, even when the rule definitions have loops. The author then does not need to use a tool to check for loops. There are two easy ways to make an AE terminate transitions:
  1. Method 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 C is allowed to happen during one transition.
  2. Method 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.

  Both methods can be modified so that they allow not one but a much larger (fixed or variable) number of updates or instances of rules to be executed. It may depend on the meaning of attributes and values in a specific application area which method and which limit is a reasonable upper bound.

4.2 Determinism

In the earlier versions of some AHS's, the author and system designer were often the same person. He or she then explicitly provided all the rules, and also provided the execution order of these rules. The AE in this case could simply follow the given order when executing the rules. The result was definite and as intended by the author. However, when we allow the execution of a particular rule to activate other rules, the AE will sometimes have to choose the order in which to execute the newly activated rules. In this

case, we allow the author to specify just *under what conditions* the result of a particular rule execution should propagate, not *when*. Different rules may become activated at the same time, without a specification by the author for their execution order. For instance, it may, as in the example of Method 2 in section 4.1, happen that a concept D can contribute twice to the knowledge of a concept E. When the contribution amounts to setting the knowledge-value of E to a definite value, the order of the updates matters. Note that the update-operations don't commute, so the order of their execution becomes important for the end result. In this case the transition may reach different final states, and the AE produces non-deterministic results.

Both methods from section 4.1 guarantee termination, but do not guarantee a deterministic result. We can see this also from the more elaborate example below, which also illustrates some other points. Assume there are three kinds of relationship types: 'prerequisite', 'link', and 'inhibit'. The 'prerequisite' graph is a directed acyclic graph used in UU-pre. The 'link' graph is a directed (possibly cyclic) graph used in GA. The 'inhibit' graph is a directed acyclic graph used in GA. The AM contains (among others) the following rules:

In UU-pre period

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

In GA period

```
if B.ready-to-read = true and B.read = false and link(A,B) then B.pre := 'Good'.
if B.ready-to-read = true and B.read = true and link(A,B) then B.pre := 'Neutral'.
if B.ready-to-read = false and link(A,B) then B.pre:= 'Bad'.
```

The rules above for instance show that there cannot simultaneously be a prerequisite and an inhibit relationship between A and B. If both prerequisite(A,B) and inhibit(A,B) are present then whether B.ready-to-read becomes true or false after accessing A depends on the order in which the rules are executed. If the system allows the assignment of different values to one attribute of one concept directly or indirectly, the order of these rules does matter to the final state. Such undesirable, ambiguous situations can be detected 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 has been specified by the author and 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.

The state reached after each period must be deterministic, otherwise it is hard to reach a well-determined final state. Method 1 only allows a single change of an attribute value, but when more than one rule applies, it is not determined beforehand, which rule will set the value. Method 2 explicitly allows more than one assignment to a single concept attribute. To guarantee termination and determinism, the methods above will have to be supplemented by adding suitable constraints on the rules. The constraints, of course, have to have the effect that the assignments to one concept attribute in one transition are either order independent or have an order specified for them, e.g., by means of a precedence relation between different types of rules. For instance, if one rule is a specific rule and the other is a generic rule, the specific rule has precedence over the generic one and only the specific rule will be executed.

In case all pertinent rules are specific (or generic), the author or the system will have to specify the precedence. We will return to this issue in the next section. To illustrate the point, in the example above we have two specific rules in GA that may become activated simultaneously. There are various possibilities to enforce an order (or precedence). For example:

- Choose the rule producing the lowest value for each attribute.
- Choose the rule producing the highest value for each attribute.

In either case, the AE needs to remember which concept attribute has been changed, and needs a strategy to compare the values. We see that in these simple cases, when only one attribute is involved, fixing the order of the updates is the same as executing only the last one. This need not in general be the case.

The two rules above illustrate that in order for a particular AE to guarantee deterministic termination constraints will have to be imposed on the kinds of rules that can be added. The choice of these constraints depends on the application domain. The choice may also depend on the termination behavior of the AE.

5. Usability

In the previous section we discussed some general mechanisms that can be employed in the construction of an adaptation engine to ensure proper behavior. The choice, which mechanism to apply for which purpose, has to be decided at design time, when the division of tasks between the system and the author is made. The aim again is to provide an AE with a behavior, which is "intuitive" for a general author. For example, it should be clear, when a particular execution order be enforced by the system, and when the decision for a certain order be left to the author.

5.1 Order imposed by the system in AHAM

The system can be given some general knowledge about the order of execution of the rules, by grouping them. The author then only needs to specify the rule group and the rule type, and the order between rules in different groups is pre-determined. For instance:

- We divide all the rules into four groups: IU, UU-Pre, GA, and UU-Post. Rules in IU are executed first (and only once). In each transition the rules in UU-Pre are executed, to update the user model. Then the rules in GA are executed to generate the presentation on the page. Finally the rules in UU-Post will be executed. The system will know the order of the rules between the different groups, and the author only needs to mark to which group the rules belongs.
- Specific rules have precedence over generic rules. They take precedence over generic rules that would apply to the same concepts. The specific rule is executed in this case, not the generic rule.
- We distinguish between *event rules* and *propagation rules*. Event rules describe the operation the AE has to take, when an event happens. The propagation rules describe how the result from the event rules propagates the other rules. The event rules have the precedence over the propagation rules. E.g. we can write the rules for clicking on a button of a menu. Assume the menu has three buttons A, B, and C.

  If click(button A) then open(A). //event rule
  If open(A) then close(B), close(C). //propagate rule

The example clearly describes what causes a menu item to open and what will propagate to the other buttons.

Note that another consequence of the grouping of rules is the prevention of some infinite loops. In the concept relationship graph of Figure 3 cycles occur, because relationships of different types are intermingled. When we select only those relationships that are executed in one period, no cycles remain. Cycles in a 'link' relationship graph don't matter.

5.2 Ordering in more complicated applications

It makes authoring work easier, if the author can write the rules without the need to take into account the execution order. But it also makes the application limited because the AE has to force an order to the rules, for instance based on the rule ID, in order to guarantee termination of the transition and to produce deterministic results. The AE does not have any insight in the requirements of knowledge domain the adaptive hypertext is authored for, unless it is told so. Therefore, the order enforced by the AE on the basis of general rules, will not always be the one intended by the author. The author therefore should be relied on to provide, e.g., for the O.D.T's the execution order for the rules.

The effect of the procedure above is that the author has to anticipate all possible ordering conflicts and provide a solution for them. The system can support this in a test phase by detecting in each transition step (see Figure 5) whether active rules will produce updates for the same attributes or will cause other active rules to be deactivated. The author, when confronted with these situations, can analyze them and provide a proper prevention mechanism, by specifying some precedence relationship or a priority among rules.
The alternative to prevention is detection. In this case, we let the system detect conflicts, when they actually occur. This alternative is realized as follows. Instead of executing rules one by one, we apply all rules that are activated in a given state at the same time. This means, that in Figure 5 R_i is not a single active rule, but the set of all rules active in UM_i. The action of each active rule is determined in the same, initial state. This is a natural thing to do, since all rules are active in this state. The actions are not applied right away, but collected in a temporary data structure, where they can be examined and conflicts, such as multiple updates of the same attribute, can be detected. Possible conflicts can be resolved by taking into account the value-domain of the attributes involved. For instance, we can take the highest value in case of an integer attribute domain, or the conjunction, in case of a Boolean value-domain, or some more specific strategy. Once the conflicts have been resolved, all updates are applied, e.g., to the UM in case of rule execution in the UU-pre phase, and a new state, UM_{i+1}, results. In this state, the new set of active rules, R_{i+1}, is determined, and so on. The author in this case does not have to provide precedence rules, but a conflict resolution mechanism for each updateable object. It is very easy to provide a default behavior in this case, and let the author handle the exceptions by overruling the default.
The behavior of the application in this detection is the same as that of the prevention approach for OIT's, but will in general be different in case of ODT's.

6. 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.)
[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.