A Generic Adaptivity Model in Adaptive Hypermedia

P.t. de Vrieze, P. van Bommel, and Th. van der Weide

University of Nijmegen
{pauldv,pvb,tvdw}@cs.kun.nl

Published in the proceedings of the adaptive hypermedia 2004 conference

Abstract. For adaptive hypermedia there is a strong model in form of the AHAM model and the AHA! system. This model, based on the Dexter Model, however is limited to application in hypermedia systems. In this paper we propose a new Generic Adaptivity Model. This state-machine based model can be used as the basis for adaptation in all kinds of applications.
This Generic Adaptivity Model, when compared to AHAM has the following features: The Generic Adaptivity Model is more low-level than the AHAM model as it does not provide hypermedia specific concepts. Instead the GAM provides an additional Interface Model that describes the bindings of the adaptivity system with the application. Finally the concepts of push and pull modelling have been incorporated in the Generic Adaptivity Model allowing system designers to make better decisions on what to store in the user model. This should allow for more flexible adaptation systems.

1 Introduction

The AHAM model [1] forms an important model in the area of adaptive hypermedia. This model, based on the Dexter Model, however is limited to application in hypermedia systems. In an aim to provide such a model for generic user modelling we have developed the Generic Adaptivity Model (GAM). This model provides a general model for an adaptation engine that can be used in a modular way to provide adaptivity in applications. As the GAM model is more general than the AHAM it is also more lowlevel. The AHAM model is based on hypermedia concepts that the generic model cannot include.

The Generic Adaptivity Model is based on the two-dimensional classification framework as defined in [2] and [3]. It includes push and pull modelling in the shape of rules and questions. This inclusion allows designers of adaptive systems to make better decisions on what to store in the User Model. This could allow for more flexible adaptation systems.

In this paper we will first give a short overview of AHAM and the AHA! system that is an implementation of it. Next we will give a description of the Generic Adaptivity Model. Following we conclude with a short comparison and some points of further research.

2 AHA! and the AHAM

The AHA! system [4],[5],[6] is an implementation of the AHAM (*Adaptive Hypermedia Application Model*) [1],[5] model as developed by de Bra et al. This model is focused on adaptive hypermedia. It is based on the Dexter model [7],[8] for hypermedia.

The AHAM originally comes from the field of educational hypermedia, and these origins can still be found in the model. There are also several features which limit the unchanged use for general interactive systems.

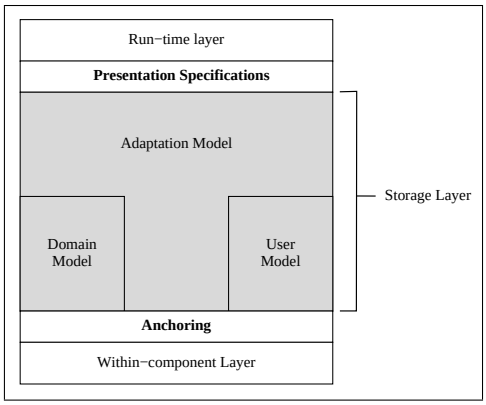


Fig. 1. The AHAM model as given in [5]

Figure 1 (which has been copied from [5]) gives a graphical overview of the AHAM model. It shows how the AHAM provides an extension of the storage layer of the Dexter model. It splits up the storage layer into an *Adaptation Model*, a *Domain Model*, and a *User Model*.

The purpose of the User Model is to store the information about one specific user. The Domain Model serves a dual purpose, both as a blueprint for the User Model, and as a specifier of the relationships between the concepts as specified by the Domain Model. The Adaptation Model defines the dynamic behaviour that performs the actual adaptation.

2.1 Domain Model

As in Dexter, the central notions of AHAM are concepts and concept relationships. An AHAM concept is an abstract representation of an information item. Below we will first discuss concepts and then the relations between them.

A concept has attribute value pairs, anchors and a presentation specification. There can be atomic an composite concepts. Concepts that are composite are composed of other concepts. The concept relationships must form a directed acyclic graph.

Anchors specify locations that other concepts can link to. They do this basically by giving some kind of name to a location or a range within the concept.

Concept relationships implement links as seen in Dexter or HTML. In AHAM concept relationships can only be between concepts or anchors. Concept relationships do not need to result in links though. They can also be used to specify relationships like *prerequisite* or *inhibits*.

Concepts and concept relationships together form the *Domain Model* (DM) as used in AHAM. Thus the Domain Model contains the concepts and the relationships between these concepts.

2.2 User Model

In AHAM the user specific information is stored in the *User Model* (UM). This user model is an overlay of the domain model. All properties in the Domain Model are used from the User Model when present.

The User Model can be changed based on events. Possible events are: "The user follows a link (to a different page)", "the user performs a test", "external information about the user is imported", or "the user explicitly sets or changes information about himself".

2.3 Adaptation Model

The actual adaptive behaviour in AHAM is defined by the *Adaptation Model* (AM). The Adaptation Model defines how events and concept relationships lead to changes to the UM.

In AHAM there are two ways to implement an Adaptation Model. The first way is to have default rules that describe how adaptation is done based on the concept relationships in the Domain Model. Such an approach is implemented in for example Interbook [9].

The second way is to have specific rules bound to specific concepts or groups of concepts. Such an approach is more flexible although it can make writing the system more involved as an explicit adaptation model needs to be written. Such an approach is used in AHA! [4]. The AHA! however provides an editor that does generate the rules for you based on graph relationships.

Adaptation in AHAM is only performed using rules. The rules used are condition action rules. These condition action rules are basically Event Condition Action (ECA) rules [10] where the Event has been merged into the Condition part. For the result this does not matter, although splitting out the event can achieve greater efficiency.

2.4 The AHA! adaptation engine

For the actual implementation of the AHAM model in the AHA! system an adaptation engine is necessary. According to the description in chapter four of [5], the *Adaptation Engine* (AE) used in AHA! has some web concepts build

in such as session management. We will only look at the adaptive functionality though.

The Adaptation Engine performs the adaptation task in a number of stages. The tasks performed by the Adaptation Engine are:

- Initialising the UM from the defaults given in the DM.
- Loading the user's stored UM overlaying it on the initial UM.
- Determine which concept "C" corresponds to the user's request and the current UM state.
- Evaluate the rules associated with the accessing of that concept "C".
- Display concept "C".
- Evaluate the rules that need to be fired post-access of concept "C".
- Update the UM. *Note though that in AHAM properties can have different persistency values. For example properties can be persistent for a session, one access, or forever. This would be equivalent to having rules that would reset the properties to default values at certain events.*

As explained earlier AHAM and AHA! perform rule based adaptation. This has as a consequence that the AE only implements rule based adaptation. In the AE it has been tried to lessen this restriction by having non-persistent properties and post and pre concept access rule execution. This does however not eliminate the problem.

3 Generic Adaptivity Model

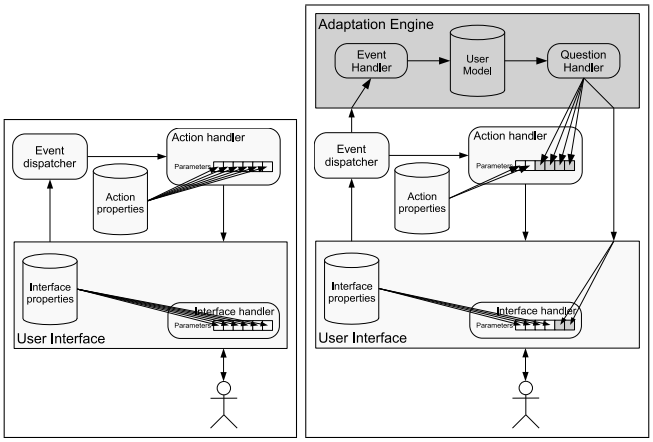
Aiming at a theory for providing adaptation to applications in general we have developed the Generic Adaptivity Model (GAM). The basis of the GAM is formed on a state-machine based view on interactive systems.

In the description of the Generic Adaptivity Model we will use terms and abbreviations that are equal to those used in the AHAM description. This does not mean that these terms have an equal meaning. Where there might be confusion we will subscript the use of the abbreviations or terms with the model they are from.

Below we will discuss our Generic Adaptivity Model, in short GAM. First we will shortly give an overview of the model. Then we will discuss the components of the model.

3.1 GAM overview

As said the GAM model is based on a state-machine view on applications. In this state-machine based view we see an application as a state-machine. At each interaction an event gets generated. This event in someway induces an action that results into a state change in the system. Such a state change can be paramaterised by external values. A user model can be used as the source of such values.



(a) A normal interactive system as a state-machine

(b) An interactive system with user modeling

Fig. 2. Comparison of normal and user modeling systems

In figure 2(a) we show a normal interactive system's state-machine. The parameters have been split out into interface properties and action properties as to separate what they influence.

In our model of interactive systems employing user modelling we have enhanced the state machine model in such a way that events also get fed into the adaptation engine. This engine will adapt the user model to these events (see Figure 2(b)).

User model consultation happens in the action handler. Actions are now not only paramaterised by static properties. They can also be parameterised by user model information.

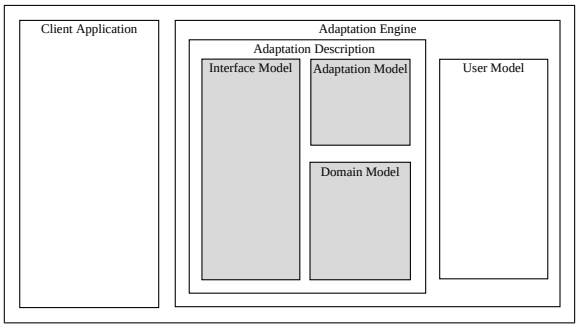


Fig. 3. The Generic Adaptivity Model

As seen in Figure 3, the two main components of the GAM are formed by the Adaptation Description and the User Model. The Adaptation Description defines how the adaptation should be performed, the User Model contains the information about the user. Both are discussed below.

3.2 Adaptation Description

An Adaptation Engine requires a description of the adaptation it should perform. This description consists of several parts that together form a logical union. This logical union is the *Adaptation Description* (AD).

The parts of the Adaptation Description are: the *Interface Model*, the *Domain Model*, and the *Adaptation Model*. Besides the Adaptation Description we also have the notion of a User Model that is used to store specific user properties as defined by the AD. In the following sections we will first describe all of these parts after which we will explain how they fit together.

3.3 Interface Model

As our AE is general and does not limit itself to hypermedia there are many events that it might need to react on. Similarly there are many questions that a program might want to ask to the AE about the user.

To allow the programs that are clients of the AE to change the actual adaptation logic without needing a change itself, we have the *Interface Model* (IM). The Interface Model describes exactly which events and which questions are available to clients of the AE. It also describes their parameters. The IM does however not contain any logic and as such can be equal for two different adaptation models.

The IM can be split into two parts. The *Event Interface Model* and the *Question Interface Model*. These models split up the IM into the event part which is used when feeding the AE with events, and the question part which is used when a client wants to query the AE.

Splitting the IM is an advantage. It allows for the situation where the modelling application is not the application that actually uses the build model.

Event Interface Model The *Event Interface Model* describes which events are provided. The IM only provides the interface. For that reason there can be additional events or questions in the AD. One of those events is the *change* event that is generated whenever the User Model changes. It is not possible to have this event in the IM.

The events in the IM are paramaterised. While for model simplicity paramaterised events could be replaced by one event for each parameter combination this would in effect lead to very complex Adaptation Descriptions where much would be duplicated.

Question Interface Model The *Question Interface Model* describes the questions that can be asked to an AE that runs this particular AD. The IM does however not need to contain all questions defined in the AD. There are many cases in which it is convenient to use intermediate questions that are internal to the Adaptation Model.

The IM as such provides only a subset of all the questions available in the AD. These questions, like events can have parameters for allowing less complexity in the actual AD.

3.4 Domain Model

Like the AHAM model the GAM also provides a *Domain Model* (DM). In GAM however there is no notion of "concept" as used in the AHAM model. As the GAM model is a general model we cannot use a high-level abstraction like the Dexter Model [7]. The GAM model is a low-level model in which the designer of the DM determines the meaning of the DM elements.

Like the $\mathrm{DM}_{\mathrm{aham}}$, the $\mathrm{DM}_{\mathrm{gam}}$ is a source from which the User Model is determined. Every element of the Domain Model has it's reflection on the User Model.

The DM consists of definitions of attributes. These attributes have a type: string, integer, boolean, floating point value or an array of such attributes. Further the attributes have a name and a default value. This default value is the value that gets used for this attribute when no value is specified for the attribute in the user model.

Object Extension In our implementation we have added an extension to the DM for objects. In this model we have *classes*, *object prototypes* and *object instantiations*.

The main concept in this is the "object" concept. The difference between object prototypes and object instantiations is that object prototypes live in the Domain Model while object instantiations live in the User Model. Objects are basically a grouping of events, rules, attributes and questions under a common prefix.

The classes in the object extension allow for specifying common elements of objects. These classes cannot themselves be instantiated, but they form the basis of each object. They define the events, rules, attributes and questions that must be present in each of the objects that derive from this class.

Object prototypes specify the names of instantiations of the classes as they are part of the user model. Object prototypes however can add additional elements or override some of the elements from the class it is based on.

Object instantiations contain only attributes and live in the User Model which is not a part of the AD. Each object instantiation corresponds to one object prototype, and for each Object Prototype, User Model pair there is one object instantiation.

The events and the interfaces of the questions of the object prototypes actually are a part of the IM, and the rule and questions are a part of the Adaptation Model. We however have discussed the object extension here for the sake of clarity.

The object extension does not form a substantial change to the model and everything that can be done with the object extension can be done without it. We have however provided it as it simplifies the creation and maintenance of Adaptation Descriptions a lot.

3.5 Adaptation Model

The *Adaptation Model* (AM) specifies the actual dynamic behaviour of the AE with respect to this AD. In contrast to the AM_{aham} the AM_{gam} splits the dynamic behaviour into two phases. These phases correspond to the framework as we have described in [2] and [3]. Basically the two phases represent updating of the user model in response to events and the querying of the user model.

Handling of events happens with a system based on ECA rules similar to AHAM. For querying we have however also added a question (or function) based system. This question system allows to store intermediate values at the right level in the User Model. As pointed out in [3] there are several advantages and disadvantages of storing user properties instead of calculating them from more basic properties.

Rules The rules in the adaptation model correspond to push adaptation as described in [3]. They are also very similar to the rules as provided by AHAM. The rules in our model are triggered by the events posted to the AE.

Before we give the steps involved in handling an event we need to mention that as a result of the object extension events can have an object as context. This context is necessary to make the class based inheritance possible. Events that are defined outside an object do not have a context[1]. Event handling works as follows:

1. An event list is initialised with the posted event.
2. The first event in the list is removed and taken for evaluation.
3. If the event has an object as context, all rules associated with the object are evaluated. All the rules whose condition is true are evaluated, not only the first one!
4. All rules without an object context are evaluated.
5. Every change to an attribute in the user model generates a change event that has the name of the attribute as argument. All those events are added to the event list
6. If there is at least one event in the event list, go back to step 2.

[1] One can also say that context-less objects have a global context

Questions Besides rules the AM_{gam} also provides questions. Questions are basically functions as seen from a programming context. Questions correspond to pull adaptation as described in [3]. As questions are used for querying the user model they are not allowed to update any values in it.

Question evaluation is fairly straightforward. The questions are specified as a program. In these programs it is possible to use other questions as functions for intermediate values. User Model attributes are also accessible as variables.

3.6 User Model

Besides the Adaptation Description, the GAM defines the existence of a User Model (UM) for each user. Each UM is tied to a specific AD and describes the properties of the user as specified by the DM.

Like the UM_{aham} our UM_{gam} is an *overlay User Model*. When an element of the AM queries for the value of a specific attribute in the model the AE performs the following steps. First the UM is examined on whether it contains a value for this attributed. If this value is contained, it is returned. When this value is not contained the default value as specified by the DM is returned.

When an element of the AM wants to update a value in the UM this attribute has it's value set. When an attribute is set to the default value in the DM, the attribute will still be present in the UM. This behaviour has as a consequence that an attribute can also have an unset value.

4 Conclusion

In the previous we have first summarised the AHAM model. Further we have given the Generic Adaptivity Model. Comparing the AHAM model to the GAM model we can find the following differences:

- While the AHAM model is specifically geared towards adaptive hypermedia, the GAM aims to provide a generic adaptivity model. As such the GAM is more low-level than the AHAM model and does not provide high-level constructs by itself. The GAM however does allow the specification of more high-level models on top of the GAM model.
- The GAM in contrast to AHAM provides an explicit interface model. In effect the AHAM model can be seen as having an interface model. This model however would be implicit and static within the AHAM model. This is sufficient for AHAM as the set of events available within a web constant is fixed and limited to access to concepts/pages.
  While the events in a hypermedia context are constant, there is still the possibility to define the questions that can be asked in the context of the adaptive hypermedia document. This means that if pull adaptation is added to an adaptive hypermedia engine, an interface model must be added too.
- The biggest difference between AHAM and the GAM is that in the GAM the concepts of push and pull adaptation have been added. In that respect

the adaptation within the AHAM can be seen as performing only push adaptation. The GAM model has added the notion of pull adaptation.
The addition of pull modelling to the model allows for the developers of Adaptation Descriptions to consider at which level to store properties. In this context it is for example more straightforward to calculate the visibility level of a concept when it's value is needed, instead of calculating it at the moment that one of the values upon which it is based, changes.

From the above we can conclude that the GAM model is more suited as a description for generic adaptivity than the AHAM model is. In the area of adaptive hypermedia we are currently still missing a model that on top of GAM can provide specific hypermedia concepts. However we believe that, once such a model has been developed, the GAM model is able to provide functionality that extends the AHAM model's functionality.

In further research we plan to create such a hypermedia model on top of GAM. We plan to use this model to create an adaptive hypermedia engine based on the generic engine we have already implemented. To compare the functionality of this engine we plan to write a translator that can translate AHA! documents into a form that can be understood by our engine.

Additionally we also plan to research how artificial intelligence techniques such as agent based learning and Bayesian or neural networks can be integrated into our model.

References

1. de Bra, P., Houben, G.J., Wu, H.: Aham: A dexter-based reference model for adaptive hypermedia. In: Proceedings of the ACM Conference on Hypertext and Hypermedia, Darmstadt, Germany (1999) 147–156
2. de Vrieze, P., van Bommel, P., Klok, J., van der Weide, T.: Towards a two-dimensional framework for user models. In: Proceedings of the MAWIS03 workshop attached to the OOIS03 conference, Geneva (2003)
3. de Vrieze, P., van Bommel, P., Klok, J., van der Weide, T.: Adaptation in multimedia systems. Multimedia Tools and Applications (2004) to appear.
4. de Bra, P., Calvi, L.: Aha! an open adaptive hypermedia architecture. The New Review of Hypermedia and Multimedia **4** (1998) 115–139
5. Wu, H.: A reference Architecture for Adaptive Hypermedia Applications. PhD thesis, Technical University of Eindhoven (2002) isbn: 90-386-0572-2.
6. de Bra, P., Aerts, A., Berden, B., de Lange, B., Rousseau, B., Santic, T., Smits, D., Stash, N.: Aha! the adaptive hypermedia architecture. In: Proceedings of the ACM Hypertext Conference, Nottingham, UK (2003) 81–84
7. Halasz, F., Schwartz, M.: The dexter hypertext reference model. In: Proceedings of the NIST Hypertext Standardization Workshop, Gaithersburg, MD, USA (1990) 95–133
8. Halasz, F., Schwartz, M.: The dexter hypertext reference model: Hypermedia. Communications of the ACM **37** (1994) 30–39
9. Brusilovsky, P., Schwarz, E., Weber, G.: A tool for developing adaptive electronic textbooks on the world wide web. In: proceedings of World Conference of the WWW, Internet and Intranet, San Franciso, CA, USA (1996) 64–69

10. Aiken, A., Hellerstein, J., Widom, J.: Static analysis techniques for predicting behavior of active database rules. ACM Transactions on Database systems **20** (1995) 3–41