GAM: A Generic Model for Adaptive Personalisation

Paul de Vrieze (pauldv@cs.ru.nl), Theo P. van der Weide (tvdw@cs.ru.nl) and Patrick van Bommel (pvb@cs.ru.nl)
Radboud University Nijmegen

Abstract. In this paper we formally define the Generic Adaptivity Model (GAM). This model provides a strong theoretical foundation for adaptive personalisation. Staying true to existing approaches in user modelling, the GAM can be used descriptively as well as prescriptively.

The GAM consists of a number of pillars bound together by a common foundation. In order to allow for extensibility the GAM is domain independent and has little restrictions in applicability. The GAM is embedded in a method for the design of adaptation models for new as well as legacy systems.

Keywords: User Modelling, Adaptive Personalisation, Generic Adaptivity Model, Adaptive Hypermedia

1. Introduction

The following situation described by Dijkstra is a good example of the human nature:

> *To end up my talk I would like to tell you a small story, that taught me the absolute mystery of human communication. I once went to the piano with the intention to play a Mozart sonata, but at the keyboard I suddenly changed my mind and started playing Schubert instead. After the first few bars my surprised mother interrupted me with "I thought you were going to play Mozart!". She was reading and had only seen me going to the piano through the corner of her eye. It then transpired that, whenever I went to the piano, she always knew what I was going to play! How? Well, she knew me for seventeen years, that is the only explanation you are going to get.* (Dijkstra, 1982)

The situation Dijkstra described above poses questions like: What happened here? Why was his mother surprised when he started playing Schubert instead of Mozart?

The example shows clearly that humans constantly monitor the world around them. They make models of the objects and people in it. This way Dijkstra's mother knew what he was going to play. More important, his mother was surprised when her prediction did not come true.

Computers, however, generally do not exhibit such modelling behaviour. Computers are built to do what they are told to, nothing more, nothing less. Most times they are not told to make a model, but behave deterministic even to the casual observer. This, however, steps over the fact that humans are attuned to others knowing their preferences. It would be awkward if couples would have to discuss every evening whether they will have what coffee when. Most computer applications, however, do exactly this. Adaptive personalisation aims to remedy this.

User adaptive systems may improve the experience of users when handling computer systems. These user adaptive systems perform incremental behaviour analysis to model the user, and use this knowledge to personalise themselves. As not all users are equal, this allows systems to also take into account a minority, instead of being forced to do that what is best for the majority of users.

Besides taking into account the specific needs of users, user adaptive systems also provide new opportunities for improvement of the user experience. A user adaptive search engine might for example provide a personalised search option that takes into account the user's interests (Google, 2006). Similarly user adaptive systems can be used for recommending books to users who have shown interest in similar books (Linden et al., 2003).

In this paper we will present the Generic Adaptivity Model (GAM). The GAM is a generic theoretical model for describing the user adaptive behaviour in a system. An early version of this model was described in (de Vrieze et al., 2004a).

In the past there has been interest in generic engines for user adaptivity (in e.g. (Hohl et al., 1996), (Beaumont, 1994), (Finin, 1989)). Those systems however have limitations as discussed in 2 and many of them are written from the point of view of one particular machine learning approach.

Our approach aims to be independent of the particular learning strategies used and as such does not suffer from problems caused by a limited scope. Consequently however our approach does not necessarily offer the same level of guidance in the development of adaptive systems as other approaches do.

The GAM is a model that aims at describing a system for incorporating user adaptivity into interactive systems. While the focus is mainly on single systems and single users the model also applies to contexts where multiple users are modelled as groups and contexts where multiple applications cooperatively maintain a single user model.

The next section discusses related work relevant to our model. Then section 3 continues with an informal sketch of the model. The formal and theoretical model is provided in section 4. Then section 5 highlights

some of the strengths of the model, including a summary of a method to create adaptation models that correspond to the GAM. Finally section 6 concludes the paper.

2. Related Work

The area of user modelling is related to many other areas such as human computer interaction, artificial intelligence, social psychology, developmental psychology (McTear, 1993), (Höök et al., 1996). These widely varying sources of research in the area of user modelling have lead to a field that has many people working only on the aspect of their expertise. We will however try to focus on the common grounds that bind these areas of research.

2.1. What is user modelling

The first question that we will answer is that of what user modelling is. While most works do not give an explicit definition, there are three views on what user modelling is. First one could see it literally and then user modelling is the acquisition of a model of a user. Second would be the usage of an explicit user model to adjust a system. The third approach would be to combine the two. This last approach implies that the user model is explicitly available to the application. If the modelling is done automatically by the system, we call it *adaptive personalisation*.

Finin et al. (Finin, 1989) describe a user model as "The information that a system has of its users is typically referred to as its user model". The task of a user modelling system given in that work are:

- Maintaining a database of observed facts about the user.
- Inferring additional true facts about the user based on the observed facts.
- Inferring additional facts which are likely to be true based on default facts and default rules.
- Maintaining the consistency of the user model by retracting default information when it is not consistent with the observed facts.
- Providing a mechanism for building hierarchies of stereotypes which can form initial partial user models.
- Recognising when a set of observed facts is no longer consistent with a given stereotype and suggesting alternative stereotypes which are consistent.

(Finin, 1989)

While this list of tasks gives a definition of a user modelling system, it is also specific to the use of default logic and stereotypes to perform this user modelling. Such an approach to user modelling is typical of that time (Kobsa, 2001). It is a good example of adaptive personalisation.

In 1993 McTear describes a user model as "Firstly, the user model can be seen as a knowledge source which contains explicit assumptions on those aspects of a user that might be relevant to the dialogue behaviour of the system." ... "The second point is that the information in a user model is typically kept in a separate knowledge base, rather than distributed throughout the system." (McTear, 1993). He further continues to describe the functions of the user modelling component to include "construction of a user model; storing, updating and deleting entries; maintaining consistency of the model; and supplying other components of the system with relevant information about the user." This definition allows for the maintenance of the user model to be performed by the user instead of the system.

The description by McTear still contains some notion of a logic based approach on user modelling. Contrasting this to Fink et al. in 2000: "For exhibiting personalised behaviour, software systems rely on a model of relevant user characteristics (e.g. interests, preferences, proficiencies, knowledge). Acquisition of these models is carried out by a dedicated user modeling component"(Fink and Kobsa, 2000), we see that all notion of a logic based approach is gone. The mentioning of a user modelling component does however point towards automatic user modelling.

In an observational approach, Kobsa (Kobsa, 1995) found the actual tasks of user modelling shell systems to be:

- Representing assumptions on users.
- Representing assumptions on groups.
- Classification of users to belong in a group.
- Recording user behaviour.
- Make assumptions about the user.
- Making stereotypes by generalising.
- Drawing additional assumptions.
- Consistency maintenance.
- Providing assumptions to applications with justifications.

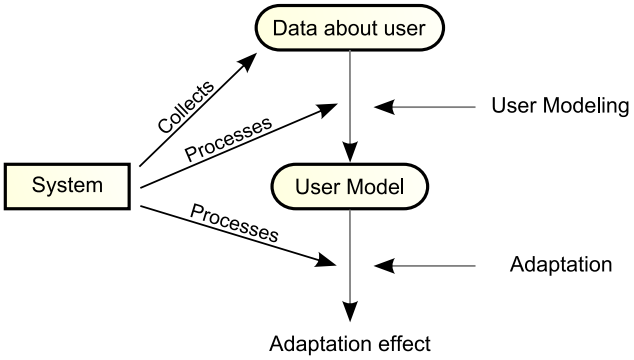

Figure 1. Classic loop "user modelling - adaptation" in adaptive systems (Brusilovsky, 1996)

– User model evaluation.

Given these definitions we define a user model to be: "A model of the relevant characteristics of a user that is or can be used to personalise the behaviour or presentation of a system". In this definition no means of acquisition of the model is given. A user model may be gotten adaptively, but also given explicitly by the user himself.

User modelling could then be described as the usage of user models. This however conflicts with the predominant meaning that includes the adaptive acquisition of those user models. To avoid confusion, and to focus on what is achieved instead of how it is achieved we prefer to speak of *adaptive personalisation* or *user adaptive systems*. A user adaptive system in this context is a system that employs adaptive personalisation.

2.1.1. *Models for user modelling*

In (Brusilovsky, 1996), Brusilovsky presented a graphical model for user modelling in adaptive systems. While the paper is on adaptive hypermedia, this model is not specific to hypermedia, but on adaptive personalisation in general. The model is given in Figure 1. This picture illustrates well how most adaptive systems work. We do believe however that this should be extended. This picture does not take into account the fact that the user model does not necessarily directly contain the answers needed about the user. Some answers can better be calculated on demand. It also does not allow a clear separation of the user modelling from the rest of the system.

It is the application that knows how it should change itself according to the user. It is however the user modelling system that knows how to answer questions about the user. Putting the reasoning that transforms user properties into answers about the user together with the actual

changing of the interface blurs this difference. As such we think that the model as presented in Figure 2 is more accurate.

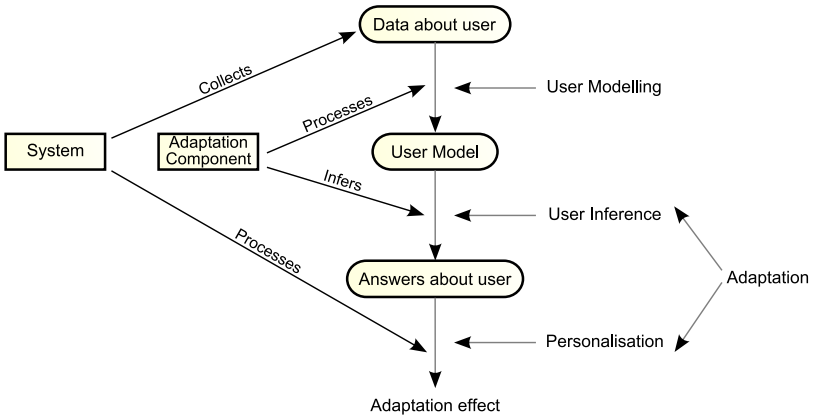

Figure 2. Improved user modelling loop for adaptive systems

If desired the adaptation component and the system could be merged in our model. This however ignores the different roles to be played.

2.1.2. *What is adaptive hypermedia*

Adaptive hypermedia is the area of research that applies user modelling in the context of hypermedia. After the mid nineties a lot of the research efforts have been focused on this sub-area of user modelling (Kobsa, 2001). As a result it is important to review this area to get a good understanding of user modelling.

Adaptive hypermedia removes some of the vagueness of the term user modelling as it is clear about its adaptiveness. At the same time, it does not reflect however that the predominant usage of adaptive hypermedia is used to personalise on users only. There are however also applications of adaptive hypermedia that explicitly take issues like user location or device capabilities into account (Fink and Kobsa, 2002).

Kobsa et al. (Kobsa et al., 1998) define adaptive hypermedia as "Adaptivity in hypermedia is proposed as a means to meet users with different needs, background knowledge, interaction style, and cognitive characteristics"(Kobsa et al., 1998). A definition that is quoted by Höök et al. in (Höök et al., 1996).

In his overview papers (Brusilovsky, 1996) and (Brusilovsky, 2001) Brusilovsky defines adaptive hypermedia as: "Adaptive Hypermedia systems build a model of the goals, preferences and knowledge of the individual user, and use this throughout the interaction for adaptation to the needs of that user" This definition summarises adaptive hypermedia quite clearly. Both definitions are however rather specific

though in what is modelled of a user. All things actually adapted on in the above definitions fall under user properties.

We would like to define adaptive hypermedia as: "adaptive hypermedia systems observe their users to deduce their properties and adapt their interface and behaviour accordingly." Some of the observation can have the form of the user's device informing the system of its capabilities.

2.1.3. *Models for adaptive hypermedia*
There are various models for adaptive hypermedia. We describe some of them below. They are fairly similar. We choose to go into most details on the AHAM model for adaptive hypermedia, as this model is well documented.

2.1.3.1. *AHAM* The AHAM (*Adaptive Hypermedia Application Model*) (de Bra et al., 1999), (Wu, 2002) model has been developed by de Bra et al. This model is focused on adaptive hypermedia. It is based on the Dexter model (Halasz and Schwartz, 1990), (Halasz and Schwartz, 1994) for hypermedia. It is implemented in the AHA! system (de Bra and Calvi, 1998), (Wu, 2002), (de Bra et al., 2003).

The AHAM originates in 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.

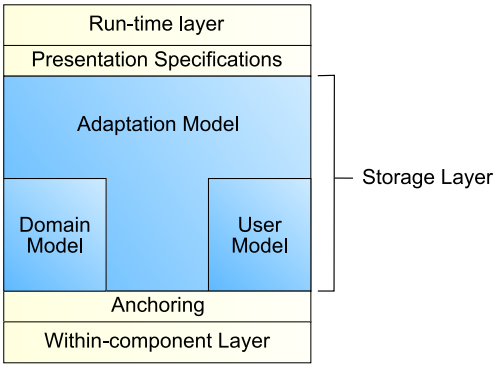

Figure 3. The AHAM model as given in (Wu, 2002)

Figure 3 (which has been copied from (Wu, 2002)) 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 how the user model influences the actual system behaviour.

2.1.3.2. *Other models* The paper (Benyon and Murray, 1993) by Benyon et al. presents a refreshing approach on user models for intelligent tutoring in splitting them up into a student model, a user profile, and a psychological model. It also recognises that in most systems the domain model is implicit. If we look at the model as we present in section 4 we can see that the domain is represented in various places. In the adaptation model, as well as the application that can perform personalisations. As such a domain model is sometimes hard to make explicit. This aspect is very similar to the AHAM.

Baumeister, et al. in (Baumeister et al., 2005) present an aspect oriented model for adaptive web applications. This is however focused on hypermedia. Further it focuses on the system, not on the user modelling heuristics.

In (Koch and Wirsing, 2002), Koch et al. describe the Munich reference model for adaptive hypermedia applications. Besides an adaptive component it contains a strong web based component. Not unlike the AHAM (de Bra et al., 1999). The user model is basically an attribute value pair approach. It does however also link a domain meta-model to the user model.

The adaptation meta model that is used, uses rules similar to AHA. The model does however, like the model we present, not aim to ensure confluence and termination. It does not specify the language to be used either. This model further does not identify the possibility of using inference on the user model to derive specific answers based on a general user model.

2.1.4. *Model elements*
While many models for adaptive hypermedia are different they share elements. For example the concept of domain model is common for many adaptive systems.

2.1.4.1. *Domain model* In (Brusilovsky and Cooper, 2002), Brusilovsky et al. give a fairly detailed description of the adapts system. This system is basically an adaptive reference of helicopter maintenance. This work is typical of many adaptive hypermedia systems in that its design is strongly based on the idea of using a domain model for the adaptation. Each "concept" in the reference is linked in the domain

model, and has its representation in the user model in various attributes that describe the users abilities with respect to the concept.

The use of a domain model approach allows easy understanding of the user model. The main limitation of the work presented in the paper is however that the learning used is quite unsophisticated. In the context of the system however it is sufficient, reading about the performance of a task and performing the task map quite well to abilities on that task.

2.1.4.2. *Machine learning* In (Müller, 2003) Müller show how user modelling may be seen from a a machine learning point of view. He points out how various methods such as naïve Bayesian classifiers, Bayesian networks, finite state machines, hidden Markov models, artificial neural networks and relational learning can be used. The paper also discusses some ways to avoid the problems that are typical of the combination of user modelling and machine learning.

2.1.4.3. *Personalised web advertising* In (Kazienko and Adamski, 2004) Kazienko et al. propose a way to perform personalised web advertisement. In this approach an advertisement is given a document vector by analysing the pages belonging to the advertiser. Similarly a document vector is calculated for the different usage patterns exhibited by the user and one for the subject section on the publishers web. These three vectors then together are used for determining the "best" advertisement.

2.1.4.4. *Cognitive user modelling* The CUMAPH environment (Habieb-Mammar and Tarpin-Bernard, 2004) uses a static user model for determining the most suitable combination of interactive elements in a hypermedia presentation. The modelling techniques used aim to model a user's cognitive abilities and choosing from various alternatives based on this.

To create a user model the authors use a sequence of interactive exercises to determine 25 indicators divided into 5 sectors. Each exercise stresses a number of cognitive properties and an individual's performance is compared with the population average and the population standard deviation. A score of 50 is average, and a score of 90 belongs to a $\overline{x} + 2\sigma$ performance.

This user model is then used to calculate a compatibility factor for each permutation of a "page". The best permutation is chosen and presented to the user. The calculation of this compatibility factor is based on plain addition of factors without using weights to distinguish between the importance of factors.

3. Framework

3.1. Interactive Systems

The generic adaptivity model is a model for describing user adaptive systems, i.e. systems that adapt themselves to users. The model assumes that these user adaptive systems are also interactive systems.

3.1.1. *Definition*

Interactive systems are systems where a user interacts with the system. This interaction can be modelled as a sequence of events initiated by the user, and actions that the system performs as a reaction to these events. We consider an application to be interactive when at least one interaction with the user happens in a normal case. This means that a dialog-box is interactive, while an application that gets all its information when it is called is not.

We also consider that in the case that other programs are used as intermediary, the interactive application is that application in the chain of applications that determines the reaction to the user. This means that in an interactive web application the interaction is in the description of when to generate which pages, not the web server or the web browser.

3.1.2. *Model of Interactive Systems*

In this section the core interaction model will be presented. This model describes the basic interaction within an adaptive system.

As explained before, in the GAM, an application without user modelling is modelled as a UML state diagram as seen in Figure 4.

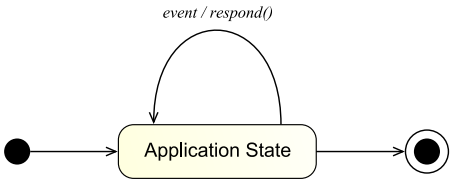

Figure 4. The application state machine

When the application starts, it will first respond to this "starting" event by initialising to some initial state. In a typical application this would for example mean that the user is presented with a screen that asks the user whether he wants to create a new document, or wants to open an existing document. After this initial state, the user will perform actions that lead to new events in the system. The system will respond to these events. As a result the system will get into a new state. This

happens until at some point a user action tells the application to close itself, and the system closes.

Besides user actions, system responses can also be triggered by external factors like timers. Such timers also lead to events in the system.

This leads to a minimal model of interactive systems. We introduce S as the set of all states that the system can be in. The state of the system includes all such things as the layout of the screen. E is the set of possible events. Then a program can be seen in its most abstract form as a function $respond_0$ that implements a response to the user on an event as its effect on the overall state of the system.

$$respond_0 : S \times E \rightarrow S$$

This respond function for interactive systems, extended for adaptivity will in the rest of the chapter be used as the basis of our theory. The extension will mean that we do not only look at the state of the system, but also have a notion of a User Model that is more or less independent of the general system state. The formal respond function for adaptivity will be introduced in section 4.3.

3.1.3. *Running example*

In this chapter a small example will be used for illustration purposes. The example is that of an adaptive coffee machine (as illustrated in Figure 5). This coffee machine recognises users by their payment card. It then tries to help the user by predicting the drink of choice based on the time of day and his user model.

The coffee machine also needs to be refilled and cleaned regularly. To determine the optimal time to do this, the machine uses an aggregated

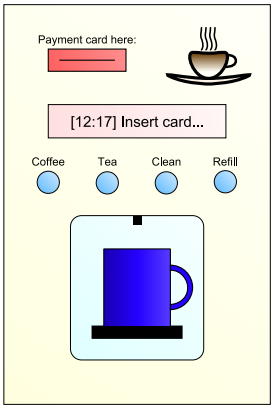

Figure 5. The application state machine

user model of all users. Successfulness of this model can be determined when cleaning and refilling happens at times that usage of the machine is low.

3.2. User adaptive systems

In the previous section we described interactive systems in general. In this section a general view of user adaptive systems is sketched.

3.2.1. *Definition*

The term *user adaptive systems* has been used. This thesis regards user adaptive systems as systems that in some way automatically personalise themselves for a user. There are two parts, personalising for a user and automatic personalisation.

Personalising for a user means that the system knows about the user and changes its appearance or behaviour accordingly. For example in the case of the information retrieval system this means that it knows that the user is not interested in programming and as such does not return programming language results on a query for the word "java".

Automatic personalisation, or adaptivity, means that the system creates a model of the user in an automatic, machine learning way. This means that the system may not rely on asking a user to describe himself. Automatic personalisation does not preclude the ability to ask users whether the conclusions of the inference process are correct. In most cases an implicit feedback system is preferable though.

3.2.2. *Why based on interactive systems*

As user adaptive systems need to create in some way a model of the user, a user adaptive system must monitor the user in some way. While non-interactive monitoring is a possibility, the generic adaptivity model limits itself to those applications where the user adaptive application is an interactive application.

The limitation to interactive systems is not major as with little effort many non-interactive systems could be seen as interactive systems where all interaction happens at once at the start of the program.

In describing how a user adaptive system works, it is important to know what the difference is between an interactive system and a user adaptive system. Let's regard an interactive system as a system where events happen that trigger actions to be performed. For a user adaptive system the events are the source of user information. The system must use these events to build an understanding or model of the user.

The actions in an interactive system determine the way the system reacts to the events induced by the system. For user adaptive systems

this is where the system can be personalised. In Figure 6 the interaction model of a user adaptive system is shown. The adaptation component monitors the events, and influences the way the system responds. Besides an adaptation component, the generic adaptivity model also

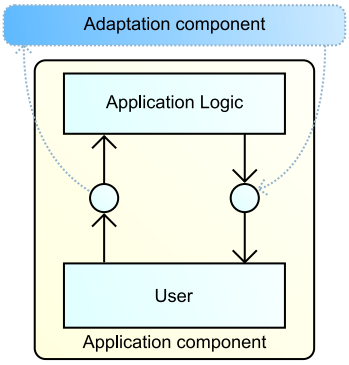

Figure 6. The interaction of a user adaptive system

distinguishes the application component. This application component would be the application if there would be no user adaptiveness. As the application component takes care of the actual personalisation it is not entirely the same as a normal application, but as in the GAM the inference is split from the application logic, the difference is minimal. The rationale behind this is that one could assume that any application makes choices that could depend on the user. In non-adaptive applications those choices are made at design time (and the alternatives are not implemented) while in adaptive applications those choices are made dynamically based on the user.

3.2.3. *Introduction to push and pull*
Throughout this thesis we use the concepts of push and pull adaptation. These terms are based on the general communication information concepts of push and pull. Pull adaptation means that the initiative for the calculations comes from the application that wants information about the user. In effect this means that pull adaptation is responsible for processing the user model to get the answer that the application wants.

Push adaptation means that the initiative is at the application. This means that the application knows when an event happens and notifies the adaptation component of this occurrence.

3.2.4. *Layer model of user adaptive systems*
The GAM model divides user adaptive systems into four layers. These layers represent the two components of a user adaptive systems. This

layer model is shown in Figure 7. As the main focus of the GAM is on the adaptation component the application component is represented by a single application layer. The interface layer represents the interface

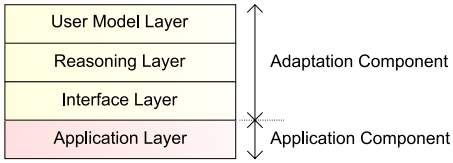

Figure 7. Layering of a user adaptive system

through which the adaptation interacts with the application component. The reasoning layer is where machine learning strategies are used to determine how to influence the actions based on the event history of the user. Finally the user model layer contains the user model as stored by the system.

3.2.5. *Componentisation of the layers*
The interactive view on user adaptive systems leads to the following model of the adaptation system (see Figure 8). First we have the application that supplies the events that have occurred. The adaptation system takes these events, and uses them to update the user model. While it is possible to directly store the events, they are normally processed in some way. This processing forms the push adaptation process.

Then, using the adaptation system, the application might have questions about the user. The answers to these questions are used as the parameters to the application logic. While the answers to the questions can be retrieved directly from a user model, it is also possible to derive the answers from the user model. This is pull adaptation.

We can split up the diagram of Figure 8 into four parts. The application, the interface, the adaptation logic and the user model. The application is straightforwardly the normal logic of the program, without the adaptive parts. The interface model is formed by the events that are handled by the adaptation system as well as the questions about the user that it can answer. As such, the interface forms a description of the way the application interacts with the adaptation system.

The user model contains the information about the user that has been collected so far. In principle it can be stored in any way desired. In our work we will use attribute-value pairs, but at this point it does not matter.

To go from the interface model to the user model and back, the adaptation logic is formed by the push and pull logics. The push logic transforms the user model based on the events that occur. The pull

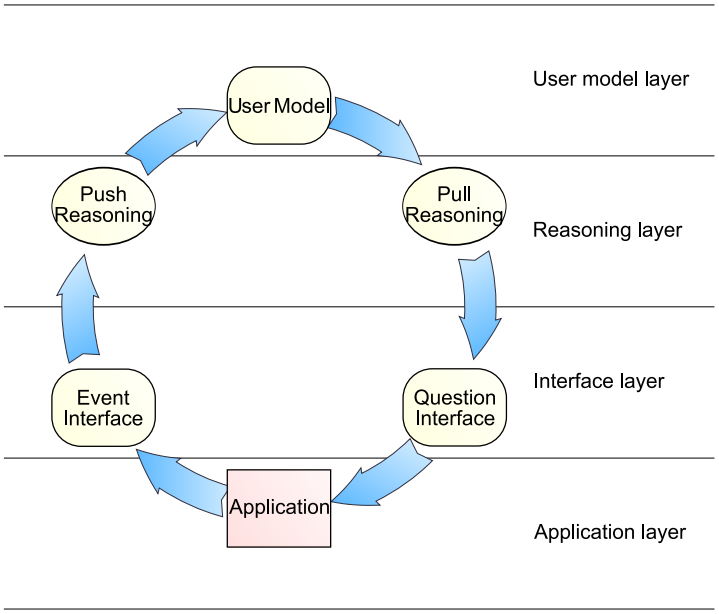

Figure 8. Overview of the Generic Adaptivity Model

logic answers the application questions, based on the information in the user model.

4. A formal theory of adaptive systems

In the rest of this chapter section we will introduce our model of adaptive systems. It is based on the view on adaptive systems that was given before.

4.1. Overview

The *General Adaptivity Model* (GAM) is a theory that is based on interactive systems. It assumes that any adaptation system can be seen as one where events happen that can be forwarded to an adaptation component. Further it assumes that the system can formulate questions to ask about the user. In this section the GAM is formally described.

In describing the GAM the layer model (see Figure 8) is used in decomposing the model into smaller parts. First some auxiliary concepts will be introduced in Section 4.2. Next the general interaction between the components is given in section 4.3. Sections 4.4 to 4.7 will then successively discuss the application layer, the interface layer, the reasoning layer and the user model layer.

4.2. Auxiliary concepts

In the formalisation of the generic adaptivity model we will use a number of general constructs as basis of the model. This section will describe these constructs.

4.2.1. *Collection*

The first construct is that of a collection. The generic type $\mathcal{C}\langle T\rangle$ represents an unordered collection of items of type T. $\mathcal{C}\langle T\rangle$ is defined by the following functions:

Definition $\mathcal{C}$1:

$$\begin{aligned} \varepsilon &: \rightarrow \mathcal{C}\langle T\rangle \\ ext &: \mathcal{C}\langle T\rangle \times T \rightarrow \mathcal{C}\langle T\rangle \\ elem &\subseteq T \times \mathcal{C}\langle T\rangle \\ take &: \mathcal{C}\langle T\rangle \rightarrow \mathcal{C}\langle T\rangle \times T \end{aligned}$$

Concerning the collection set the following axioms are defined:

Axiom $\mathcal{C}$2: Induction
Let Φ be a property for collections such that

$$\begin{aligned} &\Phi(\varepsilon) \\ &\Phi(c) \Rightarrow \Phi(ext(c,x)) \text{for all } c,\ x \end{aligned}$$

then we may conclude $\forall_c[\Phi(c)]$.

Axiom $\mathcal{C}$3: No elements for the empty collection

$$\neg elem(x, \varepsilon)$$

Axiom $\mathcal{C}$4: Elements of a collection
If x has been added to a collection C then it is an element of C

$$elem(x, ext(C, y)) = (x = y) \vee elem(x, C)$$

Axiom $\mathcal{C}$5: Extensionality

$$\forall_x \left[elem(x, C) = elem(x, D)\right] \Rightarrow C = D$$

Axiom $\mathcal{C}$6: Taking elements
take returns an element from a collection and the collection without that element

$$take(ext(C,x)) = (D,y) \Rightarrow (ext(D,y) = ext(C,x)) \wedge \neg elem(y,D)$$

Lemma $\mathcal{C}$7: Double addition has no effect:
Let $ext(ext(C,x),x) = ext(C,x)$ for any C or x.
Proof
consequence of Axioms $\mathcal{C}5$ and $\mathcal{C}4$. □

Lemma $\mathcal{C}$8: Order independence:
For all C, x: $ext(ext(C,x),y) = ext(ext(C,y),x)$.
Proof
This follows as a consequence of Axioms $\mathcal{C}5$ and $\mathcal{C}4$. □

4.2.2. *Sequence*
The second construct is that of a sequence. The generic type $\mathcal{S}\langle T\rangle$ represents a sequence of items of type T. $\mathcal{S}\langle T\rangle$ is defined by the following functions.

Definition $\mathcal{S}$1:

$$\begin{aligned} \varepsilon &: \rightarrow \mathcal{S}\langle T\rangle \\ enq &: \mathcal{S}\langle T\rangle \times T \rightarrow \mathcal{S}\langle T\rangle \\ deq &: \mathcal{S}\langle T\rangle \rightarrow \mathcal{S}\langle T\rangle \times T \\ app &: \mathcal{S}\langle T\rangle \times \mathcal{S}\langle T\rangle \rightarrow \mathcal{S}\langle T\rangle \\ elem &\subseteq \mathcal{S}\langle T\rangle \times T \end{aligned}$$

Axiom $\mathcal{S}$2: Induction
Let Φ be a property for collections such that

$$\begin{aligned} &\Phi(\varepsilon) \\ &\Phi(s) \Rightarrow \Phi(enq(s,x)) \text{ for all } s \in \mathcal{S}\langle T\rangle,\ x \end{aligned}$$

then we may conclude $\forall_c[\Phi(c)]$.

Axiom $\mathcal{S}$3: Getting an element from the front

$$deq(enq(\varepsilon, x)) = (\varepsilon, x)$$
$$deq(enq(enq(C, z), x)) = (enq(D, x), y) \quad \textbf{where } (D, y) = deq(enq(C, z))$$

Axiom $\mathcal{S}$4: Appending a list

$$app(C, \varepsilon) = C$$
$$app(C, enq(D, x)) = enq(app(C, D), x)$$

Axiom $\mathcal{S}$5: Extensionality

$$deq(C) = deq(D) \Rightarrow C = D$$

Axiom $\mathcal{S}$6: Elements of a sequence

$$\neg elem(\varepsilon, x)$$
$$elem(enq(S, y), x) = (x = y) \vee elem(S, x)$$

4.3. Model foundation

The main function in an adaptive interactive system is the *respond* function that determines the new state of the system and user model. When an event occurs, this *respond* function has two jobs. First it forwards the event to the adaptation system so it can update the user model. How this updating works is defined by the *update* function.

Second the *respond* function must determine the new state of the system. As the application is adaptive, the new state is dependent on the user model. More precisely, we see what needs to happen as a mapping from parameters to a new state. This mapping is described by the *Actions* set.

These parameters are then filled in with the answers to specific user questions. The *consult* function determines the answers to these questions.

Looking formally at this, in an adaptive system, the function *respond* has an extra parameter, the so-called user model. We introduce the set U as the set of all user models. What this set looks like will be discussed in section 4.7, but intuitively one can consider an element $u \in U$ to contain a number of attribute-value pairs.

With the set U of all user models, the function *respond* for adaptive systems can be defined as:

$$respond : S \times E \times U \mapsto S \times U$$

So $respond(s, e, u) = (s', u')$ expresses the fact that the system (being in state s) responds to event e from a user known as u by entering state s', for example by adding an image to the screen, and updating the knowledge about the user to u'.

Example

If for example the system is a coffee machine with three states, "waiting", "making_coffee", and "making_tea", the response to the "get_coffee" event for a user with a user model u (the user is identified by his payment card) in the waiting state would be:

$respond($"waiting", "get_coffee", $u) =$ ("making_coffee", $u-1$ credit)

Where $u - 1$ credit represents a new user model where the balance of the card is decreased by the price of one cup of coffee. In this example the system did not adapt itself to the user, but the user model was updated.

Before giving a definition of *respond* that refines the description of the behaviour of the system first a number of functions need to be defined. Those functions correspond to aspects of the adaptation component and the application component.

First we describe the functions related to the adaptation component (*update* and *consult*). Next the functions that enable embedding of the logic in the application are described ($parms_{action}$, *action*, $apply_{action}$). The *update* function is responsible for updating the user model according to the events that occurred. It is defined as:

$$update : E \times U \to U$$

that updates the user model in response to the event. In a way it pushes information into the user model. The implementation of this function is system dependent and defines what information of a user is deduced and stored. In many cases the information that is put into the user model is not an event but more a general user property. So instead of storing "the user selected the colour red" the user model might be updated to contain "the favourite colour of the user is red". This function is further detailed in section 4.6.1.

Example

In our coffee machine example this would mean that the system changed the balance of the user. The adaptation system would also have recorded some information that can later be used to determine that the user's favourite drink is coffee instead of tea.

The second needed function is the *consult* function that uses the user model to determine the answers to questions. The set of possible answers is represented by A, the set of possible questions by Q. The *consult* function is defined as follows:

$$consult : Q \times U \rightarrow A$$

This function is the other side of the coin in modelling users. In our approach the application looks at the adaptation component as a black box. The application does not care what happens as long as it gets answers to its questions about the user. As an example, the *consult* function takes a question from the system like "what is the colour that should be used as background for the user?" and calculates an answer based on the corresponding user model. As the user model has red as the favourite colour of the user, the answer that this *consult* function gives to the application layer is "red".

Example

Let's imagine for our coffee example that there is a button labelled "make favourite drink". To perform this action, the coffee machine needs to know the user's favourite drink. To this end it uses the *consult* function to determine the answer to this question:

$$consult(\text{“favourite drink?”}, update(u, \text{“get\_coffee”})) = \text{“coffee”}$$

With the *update* and *consult* functions we have defined the behaviour of the adaptation component. This leaves the application component. Here we need to define how the personalisation is integrated into the application response to the user.

The behaviour of the application component is defined by the action it performs in response to the user event. As the system is adaptive, this action is parameterised. The set *Actions* contains all the actions that the application component knows. There are two functions, $parms_{action}$ and $apply_{action}$, defined on this set. These functions will be discussed after the example below.

Example

In our coffee machine example the action for the "make favourite drink" event would attribute to something implementing:

if ("favourite drink?" = "coffee") **then** "make_coffee"
else "make_tea"

As there is a specific set of answers needed to be able to get the actual response of the system we introduce another function *questions* that given the action response determines the questions that must be answered.

$$parms_{action} : Actions \rightarrow \mathcal{C}\langle Q \rangle$$

Example

The result of *parms* in the coffee example would then be:

$$parms_{action}(action(\text{"make\_favourite"}, u)) = \{\text{"favourite drink?"}\}$$

The *consult* function needs to be overloaded to be able to answer collections of questions. $consult : \mathcal{C}\langle Q \rangle \times U \rightarrow \mathcal{C}\langle A \rangle$ is defined by:

$$\begin{aligned} consult(\varepsilon, u) &= \varepsilon \\ consult(q, u) &= ext(consult(X, u), consult(r, u)) \\ &\quad \textbf{where } (X, r) = take(q) \end{aligned}$$

There is one thing that must be noted for this overloaded *consult* function. That is that it seems that there is no way to associate an answer to a question. This problem can be illustrated with two examples. The first example is that of the log function. As a collection does not have an order defined, $\log\{a, x\}$ might either mean $\log_a x$ or $\log_x a$.

If however we look at common Unix commands like `grep` we can see that the options do not have any actual order; `grep -E -r query` is equivalent to `grep -r -E query`. In this example the options themselves give enough identifying information such that order is irrelevant.

To solve this problem the *question* : $A \rightarrow Q$ must be defined to retrieve the question that is answered by an answer. In line with this there must also be a *value* : $A \rightarrow V$ function that retrieves the value of an answer where V is the set of possible values. One can think of an answer from set A as a tuple containing question and value pairs where the question identifies which question is answered by the answer.

Next, as it is also necessary to get the state that results from the action with the specified parameters, the $apply_{action}$ function is specified:

$$apply_{action} : Actions \times \mathcal{C}\langle A \rangle \rightarrow S$$

The $apply_{action}$ function does whatever is necessary to execute the action and get the new state. The $apply_{action}$ function, the *Actions* set, and the *parms* function are application specific. Their implementation is not relevant for our model.

Using the above functions we can then implement the *respond* function that models the system response to user actions as follows:

$$\begin{aligned} respond(s, e, u) = & (apply_{action}(a, p), u') \\ & \textbf{where } a = action(s, e) \\ & \textbf{and } u' = update(e, u) \\ & \textbf{and } p = consult(parms(A), u) \end{aligned}$$

First a new user model is retrieved using the *update* function. The original event and the application state are then used to determine the question about the user that must be answered. The *consult* function is then used to determine the answer to this question. This answer is used as parameter to the *action* function together with the current state and the event that occurred. The *action* function then respond to the user action by changing the application state.

If we look back at Figure 6, we can identify the functions in an interactive system. They are shown as labels in Figure 9.

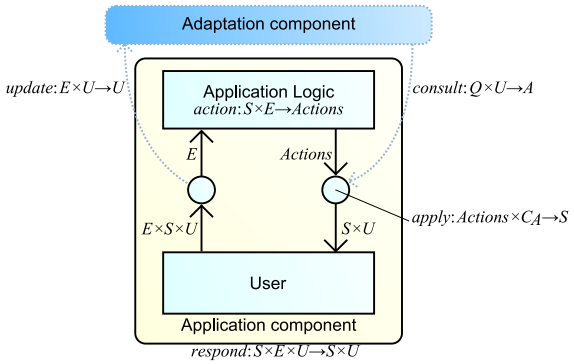

Figure 9. The interaction of a user adaptive system

Example

In the coffee machine example the "make_favourite" event event leads to the following values for the *respond* function:

a will be the action '**if** ("favourite drink?"="coffee") **then** "make_coffee" **else** "make_tea"'

u' will be the user model with a reduced balance taking into account the price of coffee ($u - 1$ credit).

p will be the set of answers to the questions about the user that are parameters to the action:

$$p = \{consult(\{\text{"favourite drink?"}, u)\} = \{\text{"coffee"}\}$$

The *respond* function is then:

$$respond(s, \text{"make\_favourite"}, u) = (apply_{action}(a, \{\text{"coffee"}\}), u - 1 \text{ credit})$$

Example

As another example, say that the user performs some "setBackground" action in which the user asks the application to set the background to a default background. This event does not lead to a user model update. There is however question that must be answered for the system to be able to respond to this event. This question is "What should be the background colour?". The pull function knows that a good way to answer the question is to use the user's favourite colour and answers "red". The *action* function consequently sets the background to red and the interaction is finished.

In the next sections we will first describe the user model in more detail, then describe the push adaptation logic of the *update* function, and after that give a more detailed description of the *consult* function that implements the pull adaptation logic.

4.4. Application layer

The application layer is defined by the application itself. As this is application dependent the application layer has no further component in the model besides the functions defined in the foundation.

4.5. Interface layer

The interface layer describes the interface in which the adaptation component interacts with the application component. This model allows for pluggable adaptation components. As far as the application is concerned, when two adaptation components have the same interface layer, they are equivalent.

Note that this pluggability does not extend to the user model. As the reasoning layer implementation is only partly dependent on the interface layer, different reasoning or machine learning strategies could be used to achieve similar effects, with very different user models.

The interface layer consists of two parts, the event component and the question component. The event component contains the events that the adaptation system can process. The question component defines the questions that the adaptation system needs to be able to answer.

4.5.1. *Event component*

The event component defines the events that may occur for a particular function. Formally we model this as an *events* function that for an implicitly given adaptation system returns the events that the system can handle.

$$events :\rightarrow \mathcal{C}\langle E\rangle$$

4.5.2. *Question component*

Similarly the question component defines all questions that may be asked about a user by the application component in a given adaptation system. This function *possibleQuestions* has the following signature:

$$possibleQuestions :\rightarrow \mathcal{C}\langle Q\rangle$$

4.6. Reasoning layer

The reasoning layer can be split up into two components. The push reasoning and the pull reasoning. The push reasoning is responsible for processing events, and updating the user model accordingly. The pull reasoning is responsible for retrieving information from the user model. The pull reasoning is not allowed to change the user model.

In the following two subsections we will first describe the push reasoning, and then the pull reasoning.

4.6.1. *Push adaptation component*
Push reasoning is responsible for acting upon events and updating the user model accordingly. While the push logic can be treated as a monolithic black box, we try to look at it in smaller pieces.

The set R is defined as the set of possible rules. In any model a number of functions must be defined over R. These functions are:

Definition R1:

$cond \subseteq R \times E \times U$	This function determines whether the rule is triggered to be executed for the current event and user model.
ac: $R \times E \times U \to U$	When the conditions of a rule are met and the right event has occurred, the *ac* function will be executed. This function takes care of updating the user model
cq: $R \times E \times U \to \mathcal{S}\langle E \rangle$	In certain rule models it is possible that changes to the user model may trigger new rules. In this case this function returns the events that result from the execution of the given rule

Axiom R2: Rule equality
If rules have equal behaviour for *cond*, *ac* and *cq* then they are equal.

Axiom R3: There is a rule ρ with special properties for all $fn : E \times U \times \mathcal{S}\langle R \rangle \to U$.
It is always possible to find a rule that has no consequences, whose condition holds, and whose action is equal to the result of a given function *fn*:

$$\forall_X \exists_\rho \forall_{e,u} [cond(\rho, e, u) \wedge ac(\rho, e, u) = fn(e, u, X) \wedge cq(\rho, e, u) = \varepsilon]$$

Now that the functions over the set R of rules have been defined, the *update* function can now be given as:

$$update : E \times U \times \mathcal{S}\langle R \rangle \to U$$

The *update* function takes the current user model, an event, and a sequence of rules that describe how the updating should function. It returns the new user model. It is important that the rules are a sequence as the order of evaluation determines the resulting user model. Any implementation of the *update* function should be according to Axiom R4.

First a number of auxiliary functions should be defined:

$$
\begin{array}{l}
hasR \subseteq \mathcal{S}\langle R\rangle \times E \times U \\
hasR(X,e,u) = \exists_r[elem(X,r) \wedge cond(r,e,u)] \\
\\
sW \subseteq \mathcal{S}\langle E \times R\rangle \times E \\
\neg sW(\varepsilon, e) \\
sW(enq(S,(f,s)),e) = (f = e) \\
\\
lnk(\varepsilon, e, r, E) \\
lnk(enq(S,(f,s)),e,r,E) = (f = e \wedge s \neq r) \vee elem(E,f) \\
\\
proc : \mathcal{S}\langle E \times R\rangle \times U \times \mathcal{S}\langle R\rangle \times \mathcal{S}\langle E\rangle \rightarrow U \\
proc(\varepsilon, u, x, E) = u \\
proc(enq(S,(e,r)),u,X,E) = \\
\qquad \left\{ \begin{array}{l}
\textbf{if } (cond(r,e,u) \wedge elem(X,r) \wedge lnk(S,e,r,app(E,cq(r,e,u)))) \\
\quad \textbf{then } proc(S, ac(r,e,u), X, app(E,cq(r,e,u))) \\
\quad \textbf{else } \bot \\
\textbf{fi}
\end{array} \right.
\end{array}
$$

The function *hasR* determines whether there is a rule that is triggered by the given event, user model combination. The function sW defines a relation between two update traces. It is true if the first event-rule pair in the trace contains the given event. *lnk* is a function that determines whether the next elements of a trace are valid for the current event and user model. This ensures that there is no duplicate rule execution for the same event instance and that the new event-rule pair is a consequence of the current or previous events. Finally the *proc* function takes everything together that both checks for invalid traces and determines the resulting user model for valid traces.

By $\psi(X,Y,e,u)$ the condition (from Axiom R4) of a well-behaved function is abbreviated:

$$\psi(X,Y,e,u) = (sW(X,e) \wedge proc(X,u,Y,\varepsilon) = update(e,u,Y))$$

All update functions must be well-behaved:

Axiom R4: update well-behavedness

$$hasR(X,e,u) \Rightarrow \exists_Y [\psi(X,Y,e,u)]$$

4.6.2. *Comparing update functions*
In this section different update functions are compared. Update functions may be nondeterministic. It can however be proven that given the

above Axiom R4 that for a specially constructed set of rules the outcome is deterministic, and can even be determined independent of the particulars of that update function to be the action for that particular rule. This specially constructed set of rules consists of only one rule, that has a condition that always holds and has no consequences.

Lemma R5:

$(cond(\rho, e, u) \wedge cq(\rho, e, u) = \varepsilon) \Rightarrow update(e, u, ext(\varepsilon, \rho)) = ac(\rho, e, u)$

Proof

Suppose ρ is a rule whose condition always holds, and that has no consequences.

$$cond(\rho, e, u) \wedge cq(\rho, e, u)$$

From the definition of *elem* we know that $elem(ext(\varepsilon, \rho), \rho)$ is always valid.

$$elem(ext(\varepsilon, \rho), \rho)$$

Using this and the fact that the condition of ρ always holds, we know that $hasR(ext(\varepsilon, \rho), e, u)$ always holds.

$$hasR(ext(\varepsilon, \rho), e, u)$$

Now by Axiom R4 we know that there must be a trace T that describes the update function in terms of a pair of (rule, event) pairs.

$$sW(T, e) \wedge proc(T, u, ext(\varepsilon, \rho)) = update(e, u, ext(\varepsilon, \rho))$$

By Axiom R4 we also know that such a trace must always start with the event parameter of the update function. Resulting we know that trace T has the form $T = enq(S, (e, r))$.

$$T = enq(S, (e, r))$$

Looking to the update equality, we know from the definition of *proc* that the guard must hold.

$$cond(r, e, u) \wedge elem(ext(\varepsilon, \rho), r) \wedge lnk(S, e, r, app(\varepsilon, cq(r, e, u)))$$

By the definition of append we know that appending a sequence to the empty sequence results in that sequence.

$$cond(r, e, u) \wedge elem(ext(\varepsilon, \rho), r) \wedge lnk(S, e, r, cq(r, e, u))$$

Then by definition of *elem* we know that $r = \rho$ as ρ is the only element of the set containing only ρ.

$$cond(\rho, e, u) \wedge elem(ext(\varepsilon, \rho), r) \wedge lnk(S, e, \rho, cq(\rho, e, u))$$

The first two elements of the guard are now found to be true. So concentrating on the *lnk* element we know that its final parameter must be ε.

$$lnk(S, e, \rho, \varepsilon)$$

Suppose that S has the form $S = ext(U, (g, s))$ we then know from the definition of link that $(g = e \wedge s \neq \rho) \vee elem(\varepsilon, g)$. The second part is obviously false, and the first part can not be true either as then the recursive invocation of process would require that $s = \rho$. We thus know that $S = \varepsilon$.

We have thus found that $T = enq(\varepsilon, e, \rho)$. As these are all bound variables we know that this T is unique. From then evaluating *proc* we find that in this case $update(e, u, ext(\varepsilon, \rho)) = ac(\rho, e, u)$, and introducing the initial assumptions we come to a conclusion for any update function for a special rule set with only one rule that has no consequences and is always true. We know that the value of the update function in this case is equal to the action of the rule in the special set.

$$(cond(\rho, e, u) \wedge cq(\rho, e, u) = \varepsilon) \rightarrow update(ext(\varepsilon, \rho), e, u) = ac(\rho, e, u)$$

□

Using the above Axiom R4 we can prove that all implementations of the update functions are equivalent in terms of the achievable user model update results. For this we will first introduce a number of abbreviations. We will write *realisesUpdate* (I) as an abbreviation for:

$$(I : E \times U \times \mathcal{S}\langle R\rangle \rightarrow U) \wedge (I \models \{R4\})$$

We will write $I_1 \leqq I_2$ to mean that I_2 is as complete as or more complete than I_1:

$$I_1 \leqq I_2 = \forall_X \exists_Y \forall e, u[I_1(e, u, X) = I_2(e, u, Y)]$$

Given a function with the signature of an update function I_1 with a rule set X, and a proper update function I_2 it is possible to find for each X a rule set Y such that $I_2(e, u, Y)$ is equivalent to $I_1(e, u, X)$.

Lemma R6: Update implementation equivalence

$$(I_1 : E \times U \times \mathcal{S}\langle R\rangle \rightarrow U) \wedge realisesUpdate(I_2) \Rightarrow I_1 \leqq I_2$$

Proof

Suppose a function I_1 that has the update signature and a function I_2 that makes the update Axiom R4 true. From Axiom R5 we know that we can choose a special rule ρ. Given this rule ρ and the rule set containing only ρ we know by Lemma R3 that I_2 can

be parameterised by this rule set to be equal to $ac(\rho, e, u)$. By definition of ρ we know that $I_2(e, u, ext(\varepsilon, \rho)) = ac(\rho, e, u)$. As such we know that a rule set exists that makes I_2 as complete as, or more complete than I_1. □

As the rules can do their own updating, the main responsibility of the *update* function lies the definition of rule evaluation semantics. There are various choices that can be made for this model. While each model has its merits we have chosen to give two alternative example definitions. Our theory does not necessarily prescribe a certain model of rule evaluation. Furthermore by Lemma R6 above we have proved that all implementations are equivalent. As such the choice only influences the form of the rules, not the possible results. The two example definitions are similar, and only differ in whether they allow event propagation.

The first implementation of a rule evaluation model is provided by the $update_1$ function. This function implements a simple rule model where rule executions cannot trigger new events. Further $update_1$ function is also a recursive function where *update* is not. The $update_1$ function is defined as:

$$update_1 : E \times U \times \mathcal{S}\langle R\rangle \times \mathcal{S}\langle R\rangle \to U$$

where $update(e, u|M) = update_1(e, u, M|M)$. The $update_1$ function is then defined as follows:

$$update_1(e, u, \varepsilon \quad |M) = u$$

$$update_1(e, u, enq(P, r)\,|M) = \begin{cases} \textbf{if } (cond(r, e, u)) \\ \quad \textbf{then } update_1(ac(r, e, u), e, P|M) \\ \quad \textbf{else } update_1(e, u, P|M) \\ \textbf{fi} \end{cases}$$

When all rules have been processed, the $update_1$ function returns the new user model. If not, it possibly updates the user model and processes the next rule, taking one rule from the sequence of rules that must still be processed.

Example

In the coffee machine example the rules would for example be:
'r_1 :"**on** "make_coffee" **if** *true* **then** $ext(u,$"favourite_drink"$,$"coffee"$)$',
'r_2 :"**on** "make_tea" **if** *true* **then** $ext(u,$"favourite_drink", "tea"$)$',
'r_2 :"**on** "make_coffee", "make_tea", "make_favourite" **if** *true* **then** $ext(u,$ "balance", $get(u,$ "balance"$) - 1)$'

With $M = r_1r_2r_3$, $e =$ "make_coffee", $fd =$ "favourite_drink", the steps of the $update_1(e, u, ext(P, r), M)$ function would then be as follows:

function	**new user model** u	**P**
$update_1(\{(\text{"balance"}, 10)\}, e, r_1r_2r_3)$	$(fd, \text{"coffee"}), (\text{"balance"}, 10)\}$	r_2r_3
$update_1(\{(\text{"balance"}, 10)\}, (fd, \text{"coffee"})\}, e, r_1r_2)$	$(fd, \text{"coffee"}), (\text{"balance"}, 10)$	r_3
$update_1(\{(\text{"balance"}, 10)\}, (fd, \text{"coffee"})\}, e, r_1)$	$(fd, \text{"coffee"}), (\text{"balance"}, 9)$	ε

While the push logic as described in $update_1$ is sufficient, for the purpose of code reuse it is also possible to have changes of the user model to trigger events. In this case we define a new function $update_2 : E \times U^* \times R^* \times R^* \rightarrow U$ that implements a different rule evaluation model as follows:

$$update(e, u|M) = update_2(u, \{e\}, M|M)$$

The $update_2$ function is defined as follows:

$$
\begin{array}{l}
update_2(u, \varepsilon\ , P\ \ \ \ \ \ \ \ |M) = u \\
update_2(u, E, \varepsilon\ \ \ \ \ \ \ \ |M) = update_2((u, F), M|M) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{where}\ (F, e) = deq(E) \\
update_2(u, E, enq(P, r)\,|M) = \\
\qquad \left\{ \begin{array}{l}
\textbf{if}\ (cond(r, e, u)) \\
\quad \textbf{then}\ \ update_2(ac(r, e, u), append(E, cq(r, e, u)), P, M) \\
\quad \textbf{else}\ \ \ update_2(u, E, P|M) \\
\textbf{fi} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{where}\ (F, e) = deq(E)
\end{array} \right.
\end{array}
$$

When there are no events to be processed, the $update_2$ function returns the user model. When all rules have been processed, the $update_2$ function continues with processing the next event and reinitialises the set of rules. In the other case, there are two possibilities. Either the events and conditions of the rule match the current event and user model. In this case, the $update_2$ function is called again with the user model that results from the action, a new sequence of events that has the events resulting from the rule appended, and the sequence of rules that excludes the current rule. Otherwise the $update_2$ function is called with the rest of the rules that still need to be evaluated for this event.

The implementation in the prototype adaptation engine as described in section 5.4 conforms to this second definition. The strategy used in

this implementation is to first fully evaluate the current event, before evaluating new events. It is also possible to immediately evaluate the events resulting from an action. Secondly this implementation evaluates all rules that get triggered for a certain event. Please note though that there are other valid implementations of the *update* function. For example AHA! (de Bra et al., 2000) uses a different rule evaluation model. Alternatively one could only evaluate the first rule for which the condition is met.

Another point of interest is that for avoidance of code repetition it is possible for the condition and action functions to use the questions as defined in the pull logic (see section 4.6.3). This however does not change the nature of the GAM model.

4.6.3. *Pull adaptation component*
Recalling from section 4.3, the *consult* function that determines an answer to a question about the user given a user model has the following signature:

$$consult : Q \times U \rightarrow A$$

In this section we split up this pull logic into question implementations. A question implementation then being a function that answers a specific question.

First the set I of all functions that provide implementations for questions is defined by two functions *apply* and *parms*. The function *apply* determines the answer based on the user model and parameters in the way defined by the implementation object:

$$apply : I \times U \times \mathcal{C}\langle A \rangle \rightarrow A$$

The *parms* function returns analogously to $parms_{action}$ on *Actions* elements the parameters that are needed to get the answer to the question:

$$parms : I \rightarrow \mathcal{C}\langle Q \rangle$$

Now we need some way to map a question to its implementation. For this the function *impl* is defined to give the implementation for a question.

$$impl : Q \rightarrow I$$

Axiom Q1: One-to-one relationship between implementations and questions
There is a one-to-one relation between implementations and questions in a specific system.

$$impl(q) = impl(r) \Rightarrow q = r$$

Using this set I of question implementations and the extended *parms* function, the $consult : Q \times U \rightarrow A$ function can be defined as:

$$consult(q, u) = apply(impl(q), u, consult(parms(impl(q)), u))$$

In the pull logic stage the user specific user properties get transformed into answers that the application system needs to be able to personalise itself. As these questions need to be answered at the moment they are needed by the system, the algorithms for the functions in the pull logic should take speed issues into consideration. If the time needed to respond to the user is too long, all advantages of adaptive personalisation are abolished.

4.7. User model layer

In this and the following sections the actual implementation of the adaptation functions is described starting with the user model.

The user model contains the knowledge of the adaptation system about the user. The notion of user model used is a narrow notion where a user model does not consist of information that can be deduced from the user model.

The set U represents the set of user models. For applications the user model is hidden by the *update* and *consult* functions. These functions form a kind of user oracle where the application gives information about the user to be able to ask questions about this user.

If we look inside this "user oracle" the choice of what is contained in the user model is difficult. There are reasons to argue for storing all the events received through the *update* function. There are however also good reasons to argue for storing the answers to specific user questions. If we look at for example AHAM (de Bra et al., 1999) it is much more inclined towards this event storage. The structure of the user model is as such tightly linked with the reasoning layer.

In this respect it would seem that the best user model in the narrow sense would be an event log. There are however concerns that make this not universally true. These concerns will have been handled in detail in (de Vrieze et al., 2005) that deals with the evaluation of adaptation models.

In our model we assume a user model can be adequately described by providing a sufficient number of characteristics of that user. A characteristic is seen as an attribute-value pair. While describing something by giving attributes sounds naive we must state that the values of attributes can be complex and thus contain whatever is desired. A value could for example contain a measure of the certainty of user model on the value.

In our model we assume a set N of possible attribute types, and a set V of possible values. A user model instance is a value assignment to a subset of these attributes. It will be sufficient to describe how property sets are constructed and how their values are retrieved. Note that we will restrict ourselves to the minimum requirements, leaving as many possibilities for implementation as possible.

Mathematically seen, a user model is introduced constructively as follows by the following functions on the set of user models U:

Definition *U*1:

ε: $\rightarrow U$	This function creates the initial user model.
ext: $U \times N \times V \rightarrow U$	The *ext* function extends a user model with an attribute-value pair.
get: $U \times N \rightarrow V$	This function retrieves the value of an attribute.

Each user model is constructed from the empty model by adding attribute-value pairs. Their construction property is reflected in the following axiom.

Axiom *U*2: Induction
Let Φ be a property for user models such that:

- $\Phi(\varepsilon)$
- $\Phi(u) \Rightarrow \Phi(U_{ext}(u, n, v))$ for all u, n, v

then we may conclude: $\forall_u[\Phi(u)]$

Axiom *U*3: Getting an element
The *get* function is defined inductively as follows:

$$get(\varepsilon, n) = \perp$$

$$get(ext(u, m, v), n) = \begin{cases} v & \text{if } m = n \\ get(u, n) & \text{otherwise} \end{cases}$$

Lemma *U*4: Double addition

We want to prove that for double addition, only the last addition has significance.

$$\forall_n[get(ext(ext(u, m, v), m, w), n) = get(ext(u, m, w), n)]$$

Proof
There are two cases: $n \neq m$, then for both cases we reduce to $get(u, n)$; $n = m$, then by applying Axiom $U3$ we get in both cases w as result.

$$
\begin{aligned}
& get(ext(ext(u, m, v), m, w), n) \\
& = (n = m \wedge w) \vee (n \neq m \wedge get(ext(u, m, v))) && \text{by axiom } U3 \\
& = (n = m \wedge w) \vee (n \neq m \wedge ((n = m \wedge v) \vee (n \neq m \wedge get(u, n)))) && \text{by axiom } U3 \\
& = (n = m \wedge w) \vee (n \neq m \wedge n \neq m \wedge get(u, n)) && \text{by simplification} \\
& = (n = m \wedge w) \vee (n \neq m \wedge get(u, n)) && \text{by simplification} \\
& = get(ext(u, m, w), n) && \text{by axiom } U3
\end{aligned}
$$

□

Axiom *U*5: Extensionality
If 2 user models look the same then they are, irrespective of the way of construction or history, equal.

$$\forall_n [get(u, n) = get(v, n)] \Rightarrow u = v$$

As a consequence, the order in which attribute-value pairs have been added has no meaning.

Definition *U*6: *UStartsWith*
UStartsWith defines a relation between two user models. It is true if the first user model is an extension of the second.

$$
\begin{aligned}
& UStartsWith \subseteq U \times U \\
& UStartsWith(u, \varepsilon) \\
\neg\, & UStartsWith(\varepsilon, ext(u, n, v)) \\
\neg\, & UStartsWith(\varepsilon, ext(u, n, v)) \\
& UStartsWith(ext(u_1, n_1, v_1), ext(u_2, n_2, v_2)) = \\
& \qquad (ext(u_1, n_1, v_1) = ext(u_2, n_2, v_2)) \vee UStartsWith(u_1, ext(u_2, n_2, v_2))
\end{aligned}
$$

5. Application and Validation

5.1. Method for creating adaptation models

In (de Vrieze et al., 2006) we have described a method for creating adaptation models that correspond with the GAM. Here we will provide a short overview of the method.

The method to create adaptation models consists of seven stages as shown in Figure 10. It is important here to distinguish the concept of a personalisation. A personalisation is a particular place in the application where adaptive personalisation can occur. A personalisation forms the link between the application and the adaptation model.

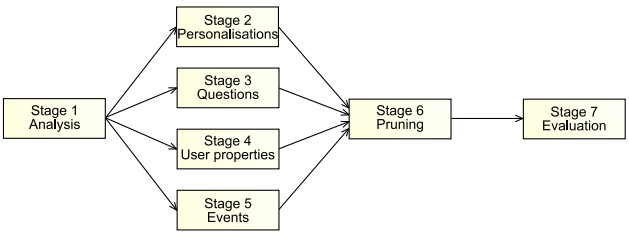

Figure 10. Method for adaptation model design

The first stage of the method is the analysis of the application at hand. This analysis is helpful for the next four stages in the understanding of the system.

The results of the analysis are used in stage 2 to find the possible personalisations in the system. Stages 3 to 5 are then concerned with finding the questions needed for the personalisations, the user properties needed to answer the questions, and finding the events needed to determine these user properties.

In Each of the stages from 3 to 5 it is possible that the needed information can not be provided. These parts are removed from the results in stage 6.

At this point there is a consistent adaptation model. There may however still be multiple ways in which certain information is acquired. In stage 7 the model is evaluated, and those elements which are too expensive, or are a less qualifying duplicate, are removed. The evaluation framework presented in (de Vrieze et al., 2005) may be used for this final stage.

5.2. Evaluation

If we look at the GAM as described in section 4 we see that it only makes modest assumptions on its application. In the rest of this section we outline these assumptions.

The first assumption made by the model is that the application is interactive. This restriction should be no problem. We see only two cases of non-interactive applications. First of all, there are applications that do not directly involve users (think for example of an application that monitors a network, or disk usage). Second, there are applications that do involve users.

If applications involve users, there is either the option that users can influence the application, or that users can not. If users can influence the application, this could be modelled as interactively asking the user for the parameters, and thus making the application modelled as interactive. Only the case in which users can not influence the application causes a case where no information can be gained about the user. In that case the application can not model the user. If, however, user information can be gained from other sources (shared user modelling) this information can still be used to perform personalisation.

The second assumption is that applications provide events to the user modelling part of the system. Given that, by nature, interactive applications react on input from the environment, there are events occurring in interactive applications. If these events occur, there should be no major obstacle in providing these events to a user modelling (sub)system.

The third assumption is that applications have personalisations that use the information from the user modelling subsystem. As the whole point of adaptive personalisation is to modify the presentation or behaviour of the system, the requirement to do so should not put restrictions on a system that provides adaptive personalisation.

The fourth assumption is that applications will be able to ask questions about the user. Questions can be seen as functions. As information about the user must be acquired in some way, using functions should be a simple way of doing so.

The fifth assumption is that the application is able to identify the user. The model assumes that the user model is known. The user model can come into life in two ways. It can be created from the meta user model, and it can be loaded from a persistent store. To load or store the model, it is necessary that a model is identified. As the model is based on the user, that effectively means that the user must be identified. The identification of users is a general requirement on user modelling systems. In certain cases this identification could be done automatically by recognising user behaviour, and thus finding the appropriate user model. Doing so should however be seriously reconsidered, as the chance for alienating users is high.

The sixth assumption is that the application logic can be modelled as producing actions. A central concept to our model is the concept of *Action*. This concept allows querying it for questions to ask, and using the answers to these questions to produce a change in the application state. Conceptually it is not needed to know the questions to ask. While it is not practical, the answers to all questions could be returned.

That leaves the concept of some action that can be parameterised by answers that returns a state change in the application. Given the

representation of the applications presentation to the user as a state, the action could be seen as a procedure or function in the native representation of the application (e.g. machine code) that has as one of its parameters the list of answers. This is no restriction as for a non-adaptive application there is some point where such a procedure could be defined except without the parameter.

In concrete terms the action concept can in many cases be replaced by the application asking questions about the user, and producing the result. Only in the case where the user's local system is responsible for the merging, this is not equivalent.

5.3. Validation

There are various aspects enabled by using the GAM. Below we will sketch the most interesting ones.

- *Speaking each others language.* By extending the GAM to include namespaces for identifying events, user properties and questions it is possible to allow sharing of user models between multiple systems.
- *Merging adaptation models.* Using the namespaces it is possible to have multiple applications cooperate in maintaining shared user properties. This requires merging adaptation models. As an adaptation model in the GAM can be seen as a directed dependency graph merging is rather straightforward. We will described this in more detail in future work.
- *User control and privacy.* The GAM separates user knowledge from application logic. As a result it is possible to have client-server based user adaptive systems where the user knowledge is kept and used at the client-side. If transferring a part of the application logic is impractical then it still is possible to maintain the user model at the client while protecting user privacy. For example by offering alternative answers without disclosing their truth.
- *Effectiveness of the user model.* The three features earlier in this list, allow user models to be shared. Thanks to keeping user control this is also possible when the application design did not foresee this particular cooperation. As such an increased amount of information leads to an increased effectiveness.
- *Disabling personalisations.* The concepts (1) *personalisation* and (2) *separation of personalisation and adaptation* make it possible to individually switch personalisations on and off.

– *Sharing user models between users.* Using user namespaces it is possible to share user model parts between users.

– *Non violating.* In (de Vrieze et al., 2005) a number of evaluation dimensions are described. The GAM model does not violate these.

5.4. Prototype

The GAM has been implemented in a prototype as described in (de Vrieze et al., 2004b). The prototype adaptation engine was developed in java to contain the following functions:

– Maintaining an adaptation description abstraction, including saving and restoring this adaptation description to and from an XML file.

– Maintaining a user model abstraction, including saving and restoring this user model to and from an XML file.

– Handling incoming events and updating the user model as a result.

– Handling incoming questions, and returning the resulting answers.

In the prototype there are some small deviations from the model. First of all, the user model is intialised lazilly (unset properties are retrieved from the meta user model). A second deviation is the introduction of an object concept that allows grouping and reuse of properties, events and questions for different similar concepts. These deviations are small though, and mainly just enhance the system, while still supporting the GAM model.

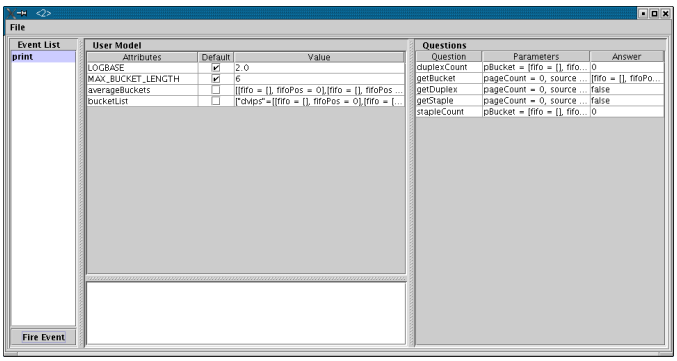

Figure 11. The adaptation model viewer

Besides the actual engine we have also implemented two applications. The first application is an adaptation model viewer (Figure 11).

This viewer can be used for simulating the functioning of an adaptation description. It loads an adaptation description. Then the user is able to ask questions to the (initially empty) user model. It also offers the possibility to post events. It is then possible to watch the effect of the events on the user model, and even possible to change user model attributes.

The second application is an adaptation description editor (Figure 12). This editor allows the adaptation model to be edited more conveniently than by editing the XML source. Editing the source XML files is especially cumbersome because the adaptation model scripts are in XML format and as such need both XML escaping and explicit line endings. With the editor this is automatically taken care of. Further the editor ensures that default values of an attribute are valid to the type of the attribute, and that attributes have a valid type. The editor also

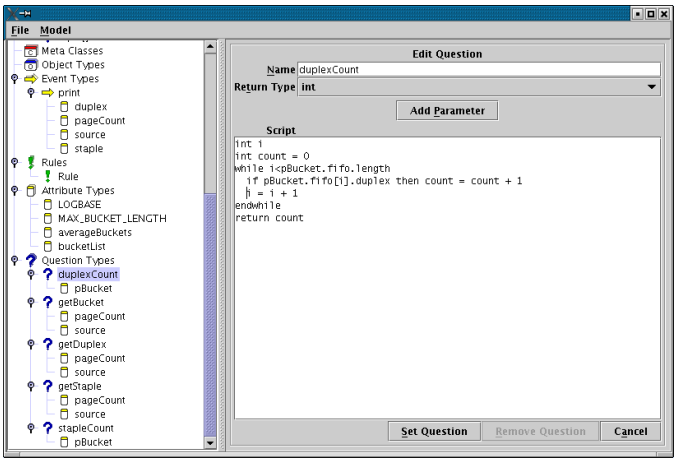

Figure 12. The adaptation model editor

offers the possibility to run the viewer (see Figure 11) on an adaptation model that is being edited.

6. Conclusions

In this work we have presented the Generic Adaptivity Model (GAM). This model distinguishes itself from existing work in a number of ways. First, it is independent of particular artificial intelligence techniques. Second, it is applicable to all kinds of user adaptive systems (e.g. not only adaptive hypermedia). Third, the GAM distinguishes two reasoning phases: push reasoning and pull reasoning, allowing all reasoning about the user to be separated from the system.

Finally, the GAM does not prescribe a particular way to look at users or the domain. This allows a great flexibility at the price of making adaptation model development more involved. It is however possible to create domain specific extensions to the GAM that do contain such assumptions. For example in educational hypermedia it is useful to use knowledge concepts. The user then has a level of proficiency in such a concept.

In future work we aim to further explore the possibilities of the GAM by elaborating how various features, such as merging adaptation models (de Bra et al., 2004) and ensuring user privacy (Kobsa, 2002) can be achieved. A particular point of interest is the relationship of the GAM with domain models. At some point the domain must be taken into account in the adaptation model. This is not limited at all by the GAM as described in this paper. A special challenge is the question how to share domain knowledge between the adaptation component and the application without this knowledge being duplicated.

References

Baumeister, H., A. Knapp, N. Koch, and G. Zhang: 2005, 'Modelling Adaptivity with Aspects'. In: *Web Engineering: 5th International Conference, ICWE 2005*, Vol. 3579 of *Lecture Notes in Computer Science*. pp. 406–416.

Beaumont, I. H.: 1994, 'User modelling in the interactive anatomy tutoring system ANATOM-TUTOR'. *User Modelling and User-Adapted Interaction* **4**(1), 21–45.

Benyon, D. and D. Murray: 1993, 'Knowledge-Based Systems'. *Adaptive systems: from intelligent tutoring to autonomous agents* **6**(4), 197–219.

Brusilovsky, P.: 1996, 'Methods and Techniques of Adaptive Hypermedia'. *User Modeling and User-Adapted Interaction* **6**(2–3), 87–129.

Brusilovsky, P.: 2001, 'Adaptive Hypermedia'. *User Modeling and User Adapted Interaction* **11**(1–2), 87–110.

Brusilovsky, P. and D. W. Cooper: 2002, 'Domain, task, and User Models for an Adaptive Hypermedia Performance Support System'.

de Bra, P., A. Aerts, B. Berden, B. de Lange, B. Rousseau, T. Santic, D. Smits, and N. Stash: 2003, 'AHA! The Adaptive Hypermedia Architecture'. In: *Proceedings of the ACM Hypertext Conference*. Nottingham, UK, pp. 81–84.

de Bra, P., A. Aerts, G. Houben, and H. Wu: 2000, 'Making General Purpose Adaptive Hypermedia Work'. In: *Proceedings of the WebNet Conference*. pp. 117–123.

de Bra, P., L. Aroyo, and V. Chepegin: 2004, 'The Next Big Thing: Adaptive Web-Based Systems'. *Journal of Digital Information* **5**(1). Article no. 247 (*2004-05-27*). `http://jodi.tamu.edu/Articles/v05/i01/DeBra/`.

de Bra, P. and L. Calvi: 1998, 'AHA! An open Adaptive Hypermedia Architecture'. *The New Review of Hypermedia and Multimedia* **4**, 115–139.

de Bra, P., G.-J. Houben, and H. Wu: 1999, 'AHAM: A Dexter-based Reference Model for Adaptive Hypermedia'. In: *Proceedings of the ACM Conference on Hypertext and Hypermedia*. Darmstadt, Germany, pp. 147–156.

de Vrieze, P., P. van Bommel, J. Klok, and T. van der Weide: 2005, 'Adaptation in Multimedia Systems'. *Multimedia Tools and Applications* **25**(3), 333–343.

de Vrieze, P., P. van Bommel, and T. van der Weide: 2004a, 'A Generic Adaptive Model in Adaptive Hypermedia'. *Lecture Notes in Computer Science* **3137**, 344–347.

de Vrieze, P., P. van Bommel, and T. van der Weide: 2004b, 'A Generic Engine for User Model Based Adaptation'. In: *Proceedings of the User Interfaces for All workshop*. Vienna.

de Vrieze, P., P. van Bommel, and T. van der Weide: 2006, 'A method for incorporating User Modelling'. *Journal of Digital Information Management* **4**(2), 135–140.

Dijkstra, E. W.: 1982, *Selected Writings on Computing: A personal Perspective*, Chapt. EWD480: "Craftsman or Scientist?" (Luncheon Speach to be held at "ACM Pacific 75" at San Francisco, Friday 18th April 1975 by Edsger W.Dijkstra, Burroughs Research Fellow.), pp. 104–109. Springer Verlag Berlin / Heidelberg. `http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD480.PDF`.

Finin, T.: 1989, 'GUMS: A General User Modelling Shell'. In: A. Kobsa and W. Wahlster (eds.): *User Models in Dialog Systems*. Springer Verlag, pp. 411–430.

Fink, J. and A. Kobsa: 2000, 'A Review and Analysis of Commercial User Modeling Servers for Personalization on the World Wide Web'. *User Modelling and User-Adapted Interaction* **10**(2–3), 209–249.

Fink, J. and A. Kobsa: 2002, 'User Modeling for Personalized City Tours'. *Artificial intelligence review* **18**(1), 33–74.

Google: 2006, 'Personalizing your search results'. `http://www.google.com/support/bin/topic.py?topic=1593`.

Habieb-Mammar, H. and F. Tarpin-Bernard: 2004, 'CUMAPH: Cognitive User Modeling for Adaptive Presentation of Hyper-documents. An Experimental Study'. *Lecture Notes in Computer Science* **3137**, 136–145.

Halasz, F. and M. Schwartz: 1990, 'The Dexter hypertext reference model'. In: *Proceedings of the NIST Hypertext Standardization Workshop*. Gaithersburg, MD, USA, pp. 95–133.

Halasz, F. and M. Schwartz: 1994, 'The Dexter hypertext reference model: Hypermedia'. *Communications of the ACM* **37**(2), 30–39.

Hohl, H., H.-D. Böcker, and R. Gunzenhäuser: 1996, 'Hypadapter: An adaptive hypertext system for exploratory learning and programming'. *User Modelling and User-Adapted Interaction* **6**(2–3), 131–156.

Höök, K., J. Karlgren, A. Wærn, N. Dahlbäck, C. G. Jansson, K. Karlgren, and B. Lemaire: 1996, 'A glass box approach to adaptive hypermedia'. *User Modelling and User-Adapted Interaction* **6**(2–3), 157–184.

Kazienko, P. and M. Adamski: 2004, 'Personalized Web Advertising Method'. *Lecture Notes in Computer Science* **3137**, 146–155.

Kobsa, A.: 1995, 'Editorial'. *User Modelling and User-Adapted Interaction* **4**(2), iii–v. Special issue on User Modeling Shell Systems.

Kobsa, A.: 2001, 'Generic User Modeling Systems'. *User Modelling and User-Adapted Interaction* **11**(1–2), 49–63.

Kobsa, A.: 2002, 'Personalized hypermedia and international privacy'. *Communications of the ACM* **45**(5), 64–67.

Kobsa, A., D. Müller, and A. Nill: 1998, 'KN-AHS: an adaptive hypertext client of the user modeling system BGP-MS'. *Readings in intelligent user interfaces* pp. 372–380.

Koch, N. and M. Wirsing: 2002, 'The Munich Reference Model for Adaptive Hypermedia Applications'. In: *Adaptive Hypermedia and Adaptive Web-Based Systems: Second International Conference, AH 2002*, Vol. 2347 of *Lecture Notes in Computer Science*. Malaga, Spain, p. 213.

Linden, G., B. Smith, and J. York: 2003, 'Amazon.com recommendations: item-to-item collaborative filtering'. *Internet Computing* **7**(1), 76–80.

McTear, M. F.: 1993, 'User modelling for adaptive computer systems: a survey of recent developments'. *Artificial Intelligence Review* **7**(3–4), 157–184.

Müller, M. E.: 2003, 'Learning for User Adaptive Systems: Likely Pitfalls and Daring Rescue'. In: *Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen, ABIS 2003*.

Wu, H.: 2002, 'A reference Architecture for Adaptive Hypermedia Applications'. Ph.D. thesis, Technical University of Eindhoven. isbn: 90-386-0572-2.

Table of Contents

List of Figures