A Generic Engine for User Model Based Adaptation

P.T. de Vrieze   P. van Bommel   Th.P. van der Weide

NIII, University of Nijmegen
P.O.Box 9010, 6500 GL Nijmegen
email:`{pauldv,pvb,tvdw}@cs.kun.nl`

Abstract

User model based adaptation becomes more and more important in interactive systems. In this paper we first review the different possible adaptation models and discuss the concepts of push, pull and hybrid adaptation. At this moment there are few published applications that explicitly use hybrid adaptation. We thus propose a way to add hybrid adaptation (which also provides push and pull adaptation) to interactive systems. Consequently such interactive systems combine the advantages of push and pull adaptation in a domain dependent way.

To be able to perform research on adaptation strategies in general, we have implemented an adaptation engine, which provides modular support for adding adaptation to existing or newly developed systems. It allows the authors of the programs using the adaptation to focus on the adaptation they are going to use instead of on the implementation. The adaptation engine works with pluggable adaptation descriptions and as such it is straightforward to change the adaptation description of a system, or to test out different adaptation descriptions.

Finally we give a short introduction into an adaptation description editor and an adaptation description viewer we developed for maintaining and developing adaptation descriptions.

1 Introduction

The area of adaptive systems is rapidly gaining scientific interest. Most research seeks to enhance human computer interaction by adapting the system to the user. This topic has already gained a lot of attention by various authors, [1], [2] and [5] in web applications, and [6] and [9] in information systems.
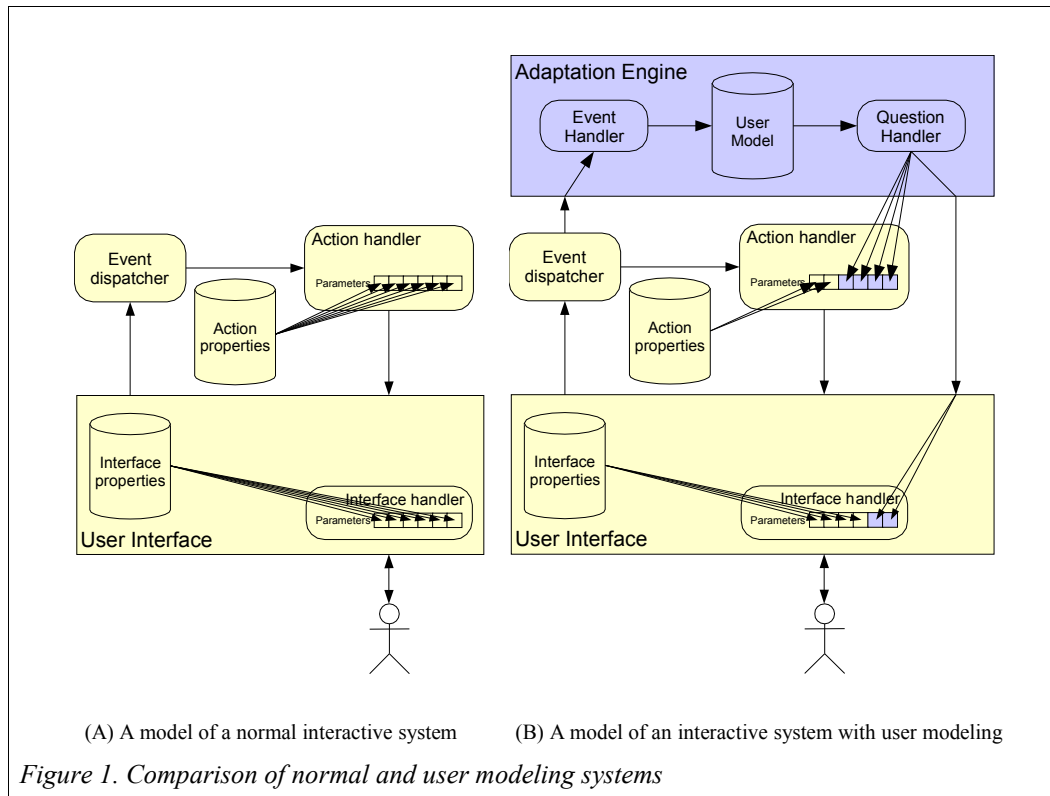
There are multiple ways to implement adaptive behaviour in an interactive system. In our research we focus on user model based adaptivity. As such we limit ourselves to user related adaptation. User based adaptation can be seen as always involving some kind of interaction. This warrants looking at adaptive systems as interactive systems.

Interactive systems can be classified into two kinds: conventional interactive systems and adaptive interactive systems (discussed in our previous work [8]). A conventional interactive system that does not employ user modelling is shown in Figure 1(a). Figure 1(b) shows an interactive system with user modelling.

Conventional interactive systems can be seen as state machines that interact with a user. This interaction is handled by a *user interface* (the gray box in Figure 1(a)). Each user action can induce a state change, after which new user actions are possible. Figure 1(b) presents an interactive system with user modelling. In this figure an adaptation engine is added which provides the adaptation functions.

In previous work [7, 8], we elaborated three different kinds of adaptation models: push

adaptation models, pull adaptation models and hybrid adaptation models. The AHA! hypermedia adaptation system [4] is a typical push adaptation system using rules. In their paper [10], Linton et. al. describe a pull adaptation system that uses event logging and post-fact analysis. There exist however no explicit hybrid adaptation systems in current literature. Our research is motivated by this fact.

(A) A model of a normal interactive system   (B) A model of an interactive system with user modeling

Figure 1. Comparison of normal and user modeling systems

In this paper, we explain the principles of hybrid adaptation systems in Section 2. The issues involved with hybrid adaptation systems are also discussed, how those issues reflect on a prototype is presented in Section 3. The paper ends with a conclusion and some future research in Section 4.
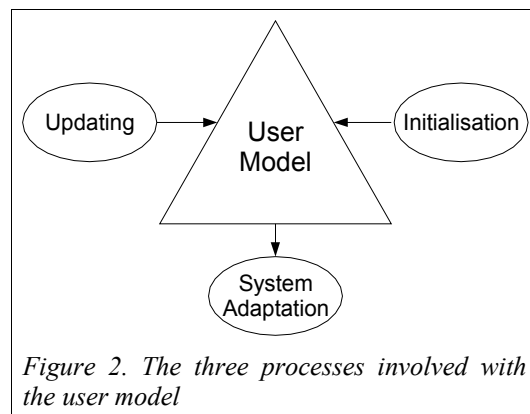
2   Principles of adaptation engines

A user modelling system is a system that shows adaptive behaviour concerning its interaction with the user. In general, adaptivity involves the use of incremental behaviour analysis for acquiring user models on the one hand. It also involves adaptation of the system behaviour to the user model on the other hand.

The client system uses the adaptation engine by feeding the user behaviour to the adaptation system. The user model is updated in reaction to the user's actions. Finally the user model is used for system adaptation. As such the user model is a main part of an adaptive system. However a user model does not appear out of thin air. The model needs to be first initialised before being used for system adaptation.

In short, we distinguish three different processes: user model initialisation, user model updating, and system adaptation. These three processes are depicted in Figure 2. The three processes together are included in the *adaptation engine* as seen in the top box of Figure 1(b).

The user model initialisation process makes the user model ready for use. The user modelling updating process processes the events and updates the user model during the runtime of the system. Similarly the adaptation process continuously uses the user model to adapt the system. The *adaptation description* defines the events that the engine recognises,

the attributes of the user model (including the initial values), and how the user model attributes are used for performing adaptation. As such the adaptation description is domain dependent.

Figure 2. The three processes involved with the user model

In most cases the adaptation cannot be performed by directly storing events and retrieving them from the user model without extra processing. There is a choice where in the adaptation this extra processing happens. It can happen before and after storage in the user model. The decision where depends also on the choice of the adaptation algorithm. For example a neural network algorithm has a specific storage of its parameters.

Our implementation as yet does only provide generic support for adaptation algorithms. This means that scripting can be used to implement any desired algorithm. In the future however we will look at providing easily accessible implementations for e.g. neural or Bayesian networks.

Concerning the location of the processing we can distinguish push, pull and hybrid strategies [7, 8]. A push strategy performs most processing before storing a user model. A pull model performs most processing at the point where information is needed about the user. A hybrid model tries to get on the middle ground in this. The choice of strategy is made when an adaptation description is created.

User adaptation is most effective when much information about the user can be collected. The way most information can be collected is to have many applications that support user modelling. To do so, it is important to make it simple to implement the user modelling. The adaptation engine serves this purpose in providing a standard interface for user modelling.

In the next section the implementation of the adaptation engine will be discussed in detail.

3   Adaptation engine implementation

In section 2 we presented the principles of user model based adaptivity. In this section we will describe the adaptation engine that we have developed in Java according to these principles.

There are a number of functions that the adaptation engine performs:

- 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.

The adaptation description abstraction is provided by the `AdaptationDescription` class together with a number of auxiliary classes. These classes for example provide typing, event types, attribute types and question types.

User model maintenance is provided by the `UserModel` class. This class stores all the properties of the user. In the implementation we have used a lazy initialisation approach that makes that attributes are only stored if their value has been set. For attributes whose value has not yet been set, the adaptation description is consulted.

Besides attributes the adaptation engine implementation provides limited support for objects. These objects allow for the grouping of attributes, events, questions, etc. in a way that multiple instances can be defined with the same code. While objects are not necessary, they make writing and maintaining adaptation descriptions simpler.

The handling of events happens in the `postEvent` method of the `UserModel` class. It is important to note that, because of object support, events can have a context. This context is the object for which the event occurred. Events that occur in a global scope do not have a context.

Our adaptation engine uses condition action rules for handling the events. This occurs according to the following pseudo code:

```
while (not eventList.empty) do
begin
  ev := eventList.pop ;
  obj := ev.object ;
  if ( obj != null ) then
  begin
    foreach rule in obj.rules do
      if rule.isTriggered(ev) then rule.fire(ev);
  end
  foreach rule in rules do
    if rule.isTriggered(ev) then rule.fire(ev);
end
```

An important note to make concerning the pseudo code is that user model changes lead to "change" events. These internal events are added to the `eventList` list. As a result one event can lead to a waterfall of rules to be triggered. As it is very well possible to create an endless loop, we have implemented a hard limit on the amount of events that can be processed as a result of one event posting. This limit is arbitrary but ensures that the engine does not become unavailable. It should be high enough though, to not limit the adaptation description design in any way.

Handling questions happens in the `askQuestion` method of the user model class. When a question is asked, its code is retrieved and executed. In the question and rule code, the user model attributes are available as variables. Questions are available as functions and can be accessed from the rule and question scripts.

An important property of questions is that they are not able to change user model attributes. This ensures that the user model stays consistent and that question results can be cached.

4 Auxiliary applications

Besides the actual engine we have also implemented two applications. The first application is an adaptation description viewer (Figure 3). 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.
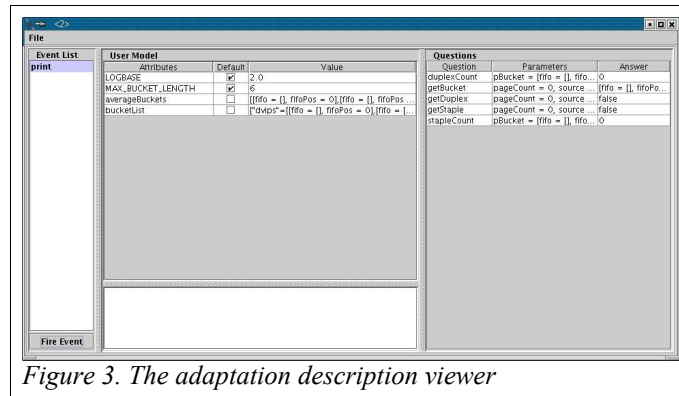
Figure 3. The adaptation description viewer

The second application is an adaptation description editor (Figure 4). This editor allows the adaptation description to be edited more conveniently than by editing the XML source. Editing the source XML files is especially cumbersome because the adaptation description scripts are in XML format and as such need both XML escaping and explicit line ends. With the editor this is automatically taken care of. Further the editor ensures that default values are of the type of the attribute, and that attributes have a valid type. The editor also offers the possibility to run the viewer on an adaptation description that is being edited.
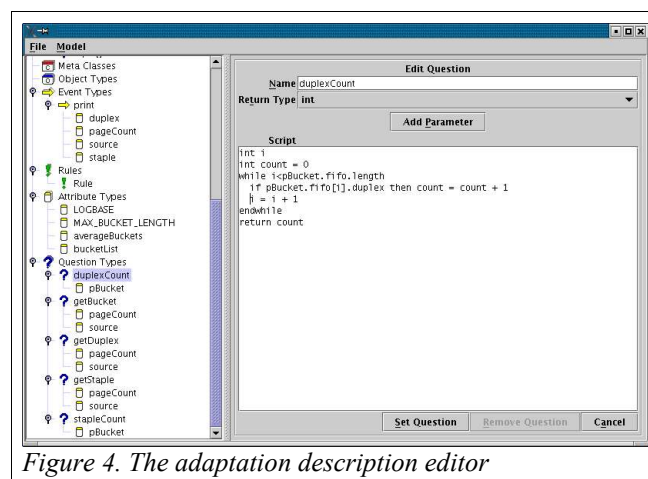

Figure 4. The adaptation description editor

In this section we have explained the key parts of the adaptation engine implementation. Further we have pointed out some of the main issues that have to be taken into account. Moreover we have presented two applications that help in creating the adaptation descriptions for the engine. In short, this adaptation engine shows that hybrid adaptation is a valid adaptation strategy that can be efficiently implemented.

5   Conclusion

In this paper we have described the main stages of adaptation. We have described an engine that can manage the adaptation and indicated its role in a program. As the first prototype of an explicitly hybrid adaptation engine our research provides a significant and original contribution to the area of user modelling research.

Our adaptation engine provides a simple interface to applications. The adaptation can be described using an adaptation description. It is possible to interchange different adaptation description for the same events and questions. This provides a way to update an adaptation description independently from the program.

The adaptation descriptions work on a script basis. As such it is possible to write use any concievable strategy in the engine. In the future we plan to provide built-in support for common strategies, but that will not limit the possible uses in any way.

In further work we plan to use the adaptation engine to implement a number of adaptive programs. We will look into an adaptive hypermedia server that can run AHA! hypermedia projects [3], while providing also pull based adaptation besides rules. We will further use the engine to perform a thorough comparison between push, pull and hybrid adaptation strategies.

6   References

[1] Johan Aberg and Nahid Shahmehri. User modelling as an aid for human web assistants. In M. Bauer, P.J. Gmytrasiewicz, and J. Vassileva, editors, *User Modeling: Proceedings of the Eight International Conference, UM01*, pages 201–203, Sonthofen, Germany, July 2001. Springer.

[2] ACM. The adaptive web. *Special Issue of Comm. o.t. ACM*, 45(5), May 2002.

[3] [3] Aha! adaptive hypermedia for all. `http://aha.win.tue.nl/`.

[4] P. De Bra, A. Aerts, G. Houben, and H. Wu. Making generalpurpose adaptive hypermedia work. In *Proceedings of the WebNet Conference*, pages 117–123, 2000.

[5] Brad Campbell and Joseph M. Goodman. Ham: a general-purpose hypertext abstract machine. In *Proceeding of the ACM conference on Hypertext*, pages 21–32, Chapel Hill, North Carolina, US, 1987. ACM Press.

[6] David N. Chin. Strategies for expressing concise, helpful answers. *Artificial intelligence review*, 14(4-5):333–350, October 2000.

[7] Paul de Vrieze, Patrick van Bommel, Jakob Klok, and Theo van der Weide. Towards a two-dimensional framework for user models. In *Proceedings of the MAWIS03 workshop attached to the OOIS03 conference*, Geneva, 09 2003.

[8] Paul de Vrieze, Patrick van Bommel, Jakob Klok, and Theo van der Weide. Adaptation in multimedia systems. *Multimedia Tools and Applications*, 2004. to appear.

[9] Josef Fink and Alfred Kobsa. User modeling for personalized city tours. *Artificial intelligence review*, 18(1):33–74, September 2002.

[10] F. Linton, D. Joy, and H. Schafer. Building user and expert models by longterm observation of application usage. In *Proceedings of the seventh international conference on User modeling*, pages 129–138. Springer- Verlag New York, Inc., 1999.