# A Method for Incorporating User Modelling

Paul de Vrieze, Patrick van Bommel, and Theo van der Weide

Nijmegen Institute for Computing and Information Sciences
Radboud University
The Netherlands
{pauldv,pvb,tvdw}@cs.ru.nl

**Abstract.** In this paper a method is presented for adding user modelling to existing software systems. The method consists of seven steps that lead from initial analysis to the definition and evaluation of the elements needed for the adaptive behaviour.
Further the concept of an adaptation element is introduced. Such an adaptation element can be used to determine the impact of personalisations.

## 1 Introduction

The area of user modelling is becoming more and more popular in the recent years [1], [2], [3], [15]. Much of this research focuses on the different ways that programs can be adapted to users. For example in [13] the authors experiment with the adaptation of a hypermedia presentation based on a model of the user's cognitive abilities. Or in [14] the authors focus on using user modelling in web advertising.

In other works [5], [6], [16], authors have proposed systems for adaptivity. These systems however are often limited in their range of application [7] or the need to reduce complexity for authors [4].

Our current work mainly focuses on the ways in which user modelling can be integrated into computer applications. In this we do not limit ourself to specific strategies as for example in [12] In earlier work [8], [9] we have looked at the general properties of adaptive systems (with which we mean systems using user modelling), described an implementation of these ideas [11], and provided a generic model for adaptivity (GAM) [10]. In this paper we continue on this venue by looking at an analysis method for adding user adaptivity to an existing system.

The main concept in our method is the concept of a *personalisation*. A personalisation is a way in which the system can be changed to adapt to a user. If for example the system fills in your name at appropriate places, then that is a personalisation. In our work we focus on those personalisations that are based on dynamically deduced user models.

Our analysis method will be illustrated based on a case. The case is set in the context of an email reader. The main focus is on a particular feature of this mail reader: a wizard that helps users in creating filtering rules for automatically sorting email messages.

To make the case more tangible a user called Joe is introduced. Joe is a rather happy user of the email application, who until now manually organises all his read emails into appropriate folders. The purpose of the case is to make Joe even more happy by offering user modelling.

Today Joe has just read an email from the linux-kernel mailing list, and wants to sort it into the "linux-kernel" mail folder. As this list is a high volume list that has hundreds of messages per day, he decides to use the new wizard to create a rule to automatically sort the linux-kernel messages into the linux-kernel mail folder.

In the paper we will first discuss the seven steps of the method in section 2 (The steps go from section 2.1 to 2.7. After that section 3 discusses the results of the method for the case, and we conclude in section 4

## 2 A Method for Incorporating User Modelling

The method consists of seven stages (see fig. 1): (1) analysis of the initial state, (2) finding possible personalisations, (3) the identification of the questions about the user needed to determine these personalisations, (4) determining the needed attributes for the user model, (5) selection of the events needed to maintain this user model, (6) in the sixth stage the results of the second to the fifth step are combined in an adaptation graph, which is then cleaned of infeasible personalisations. (7) finally for those personalisations that have multiple possible implementations, the best implementation is selected.

Our method is based on our view of user modelling as described in [10] and [8]. In this view user modelling consists of two parts, push and pull modelling. Push modelling is that part that updates the user model (from left to right), while pull modelling is the part that queries the user model (from right to left). These parts are regarded as equally important in the user modelling process. Their processing is decoupled by the user model which basically is a persistent storage facility.

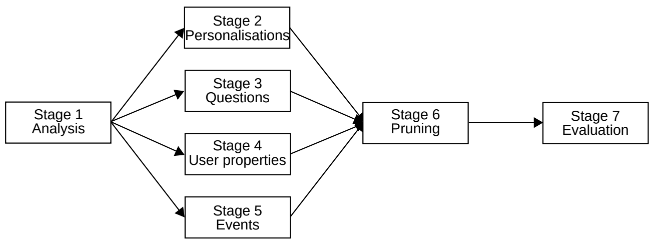


**Fig. 1.** The method

The focus of the method is the identification and selection of complete adaptation elements. An *adaptation element* is a set of elements that can achieve a personalisation. If an adaptation element is self-sustaining given the events it defines, it is called a *complete adaptation element.* The algorithms to determine complete adaptation elements are described in section 2.6.

## 2.1 Analysis of the initial state

The first stage of our method consists of the analysis of the system as it exists without user modelling. This helps to understand the system itself and to find the main points where a user may benefit from user modelling.

One appropriate way to look at the mail wizard case is to use the concept of goals. Here the assumption is that a user has a goal that he wants to achieve with the use of the wizard. For Joe the goal is to create a filter that will automatically filter all the linux-kernel mails to the linux-kernel folder.

The email filter wizard presents itself as a set of screens. At every screen the user needs to make a particular choice. This choice then leads to a new screen, but different choices can lead to different screens. When the screens are such seen as nodes, and the transitions between the screens as edges, the wizard can be seen as a graph. Before the analysis the assumption is that user modelling can help the user to find the most appropriate path through this graph.

The first step taken in the analysis of the case is to identify all paths that can be taken through the dialog, and to make up the goal a user might have taking that path. In this analysis the cases where the dialog is ended prematurely are not considered. (See also fig. 2)

The following paths and likely goals have been identified:

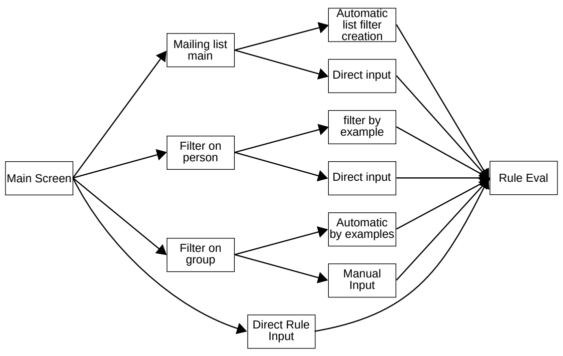


**Fig. 2.** A conversation model

- *Main screen → Mailing-list filter → Direct input of list → Rule evaluation*
  The goal of this path can be to make a rule for a mailing list for which the user (1) knows how to filter on and (2) wants to specify the rule manually.

– *Main screen → Mailing-list filter → Automatic Mailing list detection → Rule evaluation*
  The user goal could be to make a rule for some mailing list based on example mails from that list.
– *Main screen → Filter on person → Automatic sender → Rule evaluation*
  The user wants to create a filter based on the sender of messages, and wants the rule to be created based on an example message.
– *Main screen → Filter on person → Manual sender → Rule evaluation*
  In this path the goal of the user may be to create a filter based on the sender of messages, but wants to specify it manually.
– *Main screen → Filter on group → Filter by examples → Rule Overview*
  Possibly the user wants to filter on a group of senders/receivers based on a selection of emails that denote the group.
– *Main screen → Filter on group → Filter direct group → Rule Overview*
  The user's goal is to manually specify a group that he wants to filter on.
– *Main screen → Direct Rule Input → Rule evaluation*
  In this path the user wants to manually specify a rule.

When looking at these paths, a number of observations can be made:

1. There are multiple ways to achieve the same results. For example using manual or automatic rule creation.
2. There are multiple user goals, over different paths that lead to the same results.
3. There is a level of correspondence between goals and paths. Paths are however also influenced by external factors. For example when automatic rule creation fails, the user will need to use a manual approach, even when his goal is to do it automatically.

The above observations lead to an increased understanding of the possible ways to use user modelling in this system.

### 2.2 Finding personalisations

After the analysis of the original situation, it is necessary to look at possible ways in which the wizard can be personalised. In order not to reject options at an early stage, the feasibility of coming up with the needed data must not be considered for candidate personalisations. The feasibility is considered in the sixth, pruning, stage.

In the example case we can regard the personalisation to be perfect when all the user needs to do is to click the next button. The following two personalisation options can be identified. As first option the most likely transition or option can be offered as default. Many systems already have a static way to do this, but the aim in user modelling is to do this dynamically, based on the user model. The system could for example offer the creation of a mailing list rule as the default.

The second personalisation option is to have a way in which most options are already preselected for the user. The system might for example deduce that the likeliest example message is the last read message.

## 2.3 Finding the questions need to be answered

After determining what can be personalised according to the user model, the next step is to determine the information that is required to perform these personalisations. To that end, appropriate about the user must be determined.

A way to determine which screen the user will most likely visit next is to look at the user's goal. Having determined which paths lead to which goals, the most likely next transition can be determined. The question that needs to be answered to do this is: "What is most likely the goal of the user?" (likelyGoal)

To determine the most probable rules in a screen, the question asked can be: "What filter rules does the user most likely want to create given the current position in the dialog?" (probableRules) The answer in Joe's case should include a number of rules for moving messages to the linux-kernel folder.

To determine other options like the default example mail, the question that needs to be answered is: "Which message does the user want to use as example?" (exampleMessage) A good candidate answer is probably the last message the user watched.

## 2.4 Finding the user properties needed to answer the questions

Knowing the questions about the user that must be answered, the fourth step is to determine how these questions can be answered. In this step it is assumed that all desired general knowledge of the user is contained in the user model. The algorithms that answer the questions must translate this general knowledge into specific answers. As a side-product of determining the algorithms, the needed user properties are determined.

A plausible way to determine the most likely goal of the user, is to determine the probability distribution of all goals, and select the goal with the highest probability. There are three factors that influence this probability distribution, namely the frequency of occurrence of each goal in history (the history property), the current position in the dialog (a parameter from the system), and third those rules that the user is most likely to want to create (the probableRules property).

To determine the most likely rules given the current state, the user model must contain a list of likely rules (again the probableRules property), and filter them according to the current position in the dialog.

For answering the question what message must be used as example, the most straightforward heuristic is to choose the last viewed message. This means that the last viewed message must be recorded in the user model (the lastMessage property).

Concluding this analysis there are three elements that must be contained in the user model: A history of previous goals of the user, a list of likely rules, and the last viewed message.

## 2.5 The events to record

Having determined the required elements of the user model, the next step is to determine how these user properties can be determined. All things that happen in the application can be seen as events. The heuristics to translate these events into a changed user model determine the events that need to be recorded.

The most simple heuristic in our case is the heuristic to determine the last viewed message. We can simply update the user model each time a message is viewed. To do this we need that a mailViewed event is passed to the user modelling system, with the viewed email message as a parameter.

A little more complex is the history of previous user goals. To maintain such a history it is necessary to react to the successful completion of the wizard by recording the user's goal. The hard part here is to determine the user's goal based on the interaction history. A way to circumvent this is to assume that goals have a one to one correspondence to the followed path. This also solves the problem of determine the most likely next transition given a goal and a current state.

To record the last path of the user, it is however necessary not only to record the result of the wizard, but also the path followed. To this end, the user modelling system must record every transition, and the completion or cancellation of the wizard. We also need to introduce a new temporary property in the user model to record the path as it is followed. At the end of the wizard this property is reset, and at a successful completion the path is added to the history.

Third to maintain a list of likely rules, the mailViewed event can be reused. Based on the email a set of rules can be developed that creates candidate rules and their chances. The resulting candidate rules (one message can result in multiple candidate rules) are then recorded in the user model. Assuming that newer rules have a higher likelihood, a timestamp must be recorded with the rules. In the querying of the candidate rules, this timestamp is used to adjust the probabilities.

Summarising the following events need to be recorded: the viewing of a message, the transitions in the wizard, the completion or cancelling of the wizard.

## 2.6 Pruning

The sixth step in our method for adding user modelling behaviour is to review the results of the previous four steps. In this step we will remove all user model properties, user questions and personalisations that need information that can not be determined.

Looking at the results of the previous steps a directed labelled graph (see fig. 3) can be created. We call this graph the *adaptation graph*. The nodes of this graph are events, user model properties, user questions and personalisations. The edges and their labels are formed by the heuristics or algorithms that transform between them. The directions are determined by the prerequisiteness relation. For example in our case there is an edge from the lastMessage user model property to the mailViewed event.

A formal definition of the adaptation graph is given in Definition 1

**Definition 1.** *Let $\mathbb{E}$ be the set of all events, $\mathbb{U}$ be the set of all user model properties, $\mathbb{Q}$ be the set of all question prototypes, and $\mathbb{P}$ be the set of personalisations,*

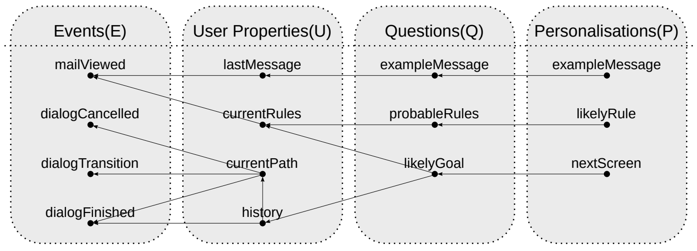


**Fig. 3.** The adaptation graph

*then the set $N$ of nodes of the adaptation graph is defined as:*

$$N = \mathbb{E} \cup \mathbb{U} \cup \mathbb{Q} \cup \mathbb{P}$$

*The set $A$ of arcs of an adaptation graph is restricted to:*

$$A \subset \mathbb{P} \times \mathbb{Q} \cup \mathbb{Q} \times (\mathbb{Q} \cup \mathbb{U}) \cup \mathbb{U} \times (\mathbb{Q} \cup \mathbb{U} \cup \mathbb{E})$$

Further two functions begin and end are defined that return the starting and end points of the arcs respectively.

This graph (fig. 3) can now be used to determine *adaptation elements*. This process works as follows: for each personalisation consider the arcs starting in that personalisation. The subgraph that is reachable from such an arc is an adaptation element. So there is an adaptation element for each edge out of a personalisation.

The algorithm for determining the adaptation elements is implemented by the getAdaptationElements function as follows:

```
1:  **function:** getAdaptationElements $\rightarrow \mathcal{P}(\mathcal{P}(A))$
2:    $X = \emptyset$
3:    **for all** $p \in \mathbb{P}$ **do**
4:      **for all** $a \in A \wedge \mathsf{begin}(a) = p$ **do**
5:        $X = X \cup \{\mathsf{closure}(a, \emptyset)\}$
6:      **end for**
7:    **end for**
8:    **return** $X$
9:  **end function**

10: **function:** closure($a \in A, s \in \mathcal{P}(A)$) $\rightarrow \mathcal{P}(A)$
11:   **if** $a \in s$ **then**
12:     **return** $\emptyset$
13:   **else**
14:     $r = \{a\}$
15:     **for all** $a' \in A \wedge \mathsf{end}(a) = \mathsf{begin}(a')$ **do**
16:       $r = r \cup \mathsf{closure}(a', r \cup s)$
17:     **end for**
```

```
        **return** $r$
    **end if**
**end function**
```

This recursive function returns a set of subgraphs of the adaptation graph. Every outgoing arc from every personalisation signifies one adaptation element. The closure function returns the transitive closure for these arcs such that the returned arcs form an adaptation element.

An adaptation element can either be complete or incomplete. An adaptation element is complete when all leaf nodes are events, and there are no further sinks. The algorithm for determining completeness of an adaptation element is defined by the function $\mathsf{complete}_1$ as follows:

```
**function:** $\mathsf{complete}_1(p \in \mathbb{P}) \rightarrow$ boolean
    **return** $\mathsf{complete}_2(p, \{p\})$
**end function**

**function:** $\mathsf{complete}_2(n \in N, v \in \mathcal{P}(N)) \rightarrow$ boolean
    **if** $n \in \mathbb{E}$ **then**
        **return** true
    **else**
        $S = \{a \in A | \mathsf{begin}(a) = n \wedge \mathsf{end}(a) \notin s\}$
        **if** $S = \emptyset$ **then**
            **return** false
        **else**
            $b =$ true
            **for all** $a \in S$ **do**
                $b = b \wedge \mathsf{complete}(\mathsf{end}(a), v \cup \{\mathsf{end}(a)\})$
            **end for**
            **return** $b$
        **end if**
    **end if**
**end function**
```

The $\mathsf{complete}_1$ function initialises the $\mathsf{complete}_2$ function that performs the actual work. The recursive $\mathsf{complete}_2$ function determines for every outgoing arc from a node whether the nodes reachable from this arc are complete. Event nodes are allways complete. When a node has no outgoing arcs, or only outgoing arcs that have allready been seen, it is incomplete if it is no event node.

Pruning consists of leaving only that part of the full graph that is part of complete adaptive elements. The result is that only those personalisations needed for at least one personalisation are left.

In our case there are no sinks in the adaptation graph, so all parts are retained. There are three complete adaptive elements. One element the next screen personalisation, and two elements for the default option personalisation.

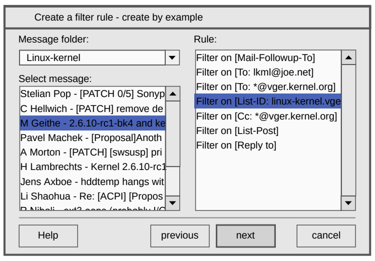
### 2.7 Selection

In the seventh and last stage the final adaptive elements are selected. The first step is to select the best adaptive element for each personalisation. There is a catch however, as is exemplified by our case, it is possible that the adaptive elements for one personalisation are different. The solution is to split the personalisation up. As a result in our case we now have three personalisations with each one adaptive element.

## 3 A Demonstration

As a result of our method in the previous section we have identified three adaptation elements in the case. As step 7 in section 2.7 has ensured that each personalisation is accomplished by only one adaptation element, we will address each adaptation element by the personalisation it accomplishes.

In this section we will discuss how the adaptation elements are used in a system, illustrating it along the lines of the email wizard case.



Message folder: Linux-kernel

Select message:
Stelian Pop - [PATCH 0/5] Sonyp
C Hellwich - [PATCH] remove de
M Geithe - 2.6.10-rc1-bk4 and ke
Pavel Machek - [Proposal]Anoth
A Morton - [PATCH] [swsusp] pri
H Lambrechts - Kernel 2.6.10-rc1
Jens Axboe - hddtemp hangs wit
Li Shaohua - Re: [ACPI] [Propos

Rule:
Filter on [Mail-Followup-To]
Filter on [To: lkml@joe.net]
Filter on [To: *@vger.kernel.org]
Filter on [List-ID: linux-kernel.vge
Filter on [Cc: *@vger.kernel.org]
Filter on [List-Post]
Filter on [Reply to]

**Fig. 4.** The third screen in the filter wizard

An adaptation element has two points of interaction with the system: the events, and the personalisations. For example when fig. 4 is displayed to the user, the system will have generated a dialogTransition event. This event has caused the currentPath property of the user model to be updated.

Besides this event, there are also two personalisations that are used in the dialog: the exampleMessage, and the likelyRule personalisation. The next screen personalisation is however not used as we can see from fig. 2 that there is only one transition from this screen.

This leads to the observation that this screen involves all three adaptation elements. The nextScreen adaptation element is however only used to update the user model, while the likelyRule and exampleMessage adaptation elements were only used for personalisations. We can also see that for this dialog the adaptation elements are completely independent as the dialogTransition event is not a part of the likelyRule and exampleMessage adaptation elements.

## 4 Conclusion

In this paper we have presented a method that can be used to extend legacy systems for user modelling. This method has seven steps that go from the initial analysis to the selection and evaluation of personalisation options. As the method does not presume special properties of the legacy system, it is generally applicable. The method is, with little change also appropriate for newly designed systems, as only step 1 needs to be adjusted for such systems.

As a side-effect of the method we have introduced the concepts of *adaptation graph* and *adaptation element*. An adaptation graph represents the various parts involved with user modelling such as: events, user properties, questions, personalisations, and their relationships. Adaptation elements the are smallest self-sustaining subgraphs of an adaptation graph starting at personalisations. If an adaptation element can be actually accomplished within the adaptation graph, then the element is complete. Complete adaptation elements are used for the selection of the personalisations of the system.

In furture work we will apply our method to other kinds of systems, such as information retrieval and adaptive hypermedia. Further we hope to create a formal framework for user modelling systems that allows for analysis of the properties of these systems.

## References

1. ACM. The adaptive web. *Special Issue of Communications of the ACM*, 45(5), May 2002.
2. ACM. Attentive user interfaces. *Special Issue of Communications of the ACM*, 46(3):30–72, March 2003.
3. A. Aiken, J.M. Hellerstein, and J. Widom. Static analysis techniques for predicting behavior of active database rules. *ACM Transactions on Database systems*, 20(1):3–41, 1995.
4. P. Brusilovsky, E. Schwarz, and G. Weber. A tool for developing adaptive electronic textbooks on the world wide web. In *proceedings of World Conference of the WWW, Internet and Intranet*, pages 64–69, San Franciso, CA, USA, October 1996.
5. Peter Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3):87–129, 1996.
6. Peter Brusilovsky. Adaptive hypermedia. *User Modeling and User Adapted Interaction*, 11(1-2):87–110, 2001.
7. Paul de Bra, Geert-Jan Houben, and Hongjing Wu. Aham: A dexter-based reference model for adaptive hypermedia. In *Proceedings of the ACM Conference on Hypertext and Hypermedia*, pages 147–156, Darmstadt, Germany, 1999.
8. 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.
9. Paul de Vrieze, Patrick van Bommel, Jakob Klok, and Theo van der Weide. Adaptation in multimedia systems. *Multimedia Tools and Applications*, 2004. to appear.
10. Paul de Vrieze, Patrick van Bommel, and Theo van der Weide. A generic adaptive model in adaptive hypermedia. *Lecture Notes in Computer Science*, 3137:344–347, August 2004.

11. Paul de Vrieze, Patrick van Bommel, and Theo van der Weide. A generic engine for user model based adaptation. In *Proceedings of the User Interfaces for All workshop*, 2004.
12. Josef Fink and Alfred Kobsa. User modeling in personalized city tours. *Artificial Intelligence Review*, 18(1):33–74, 2002.
13. Halima Habieb-Mammar and Franck Tarpin-Bernard. Cumaph: Cognitive user modeling for adaptive presentation of hyper-documents. an experimental study. *Lecture Notes in Computer Science*, 3137:136–145, August 2004.
14. Przemyslaw Kazienko and Michal Adamski. Personalized web advertising method. *Lecture Notes in Computer Science*, 3137:146–155, August 2004.
15. Alfred Kobsa. Generic user modeling systems. *User Modeling and User-Adapted Interaction*, 11(1-2):49–63, 2001.
16. Hongjing Wu. *A reference Architecture for Adaptive Hypermedia Applications*. PhD thesis, Technical University of Eindhoven, November 2002. isbn: 90-386-0572-2.