

Foundations for personalised documents: a scrutable user model server

Judy Kay
Bob Kummerfeld
Piers Lauder

Department of Computer Science
University of Sydney
judy,bob,piers@cs.usyd.edu.au

Personalisation of documents offers the promise of better support of users. Such personalisation relies on a model of the user.

This paper describes the Personis user model server. The novel and defining characteristic of Personis is that it was designed around the requirement that users be able to scrutinise and control their user models. The paper reviews the need for a scrutable user model server.

To illustrate the operation of the server, the paper provides an overview of the way it operates in conjunction with an application system which provides a personalised jazz music service. Finally, we discuss previous work, indicating how it relates to the Personis project.

1. Introduction

There is a large and growing interest in personalised documents. These offer the promise of a significant advantage for electronically delivered documents over conventional paper ones. For example, there has been work in personalised hypertext for teaching in systems like ELM-ART (Weber and Specht, 1997, Brusilovsky, Schwarz, and Weber, 1996) in customised documentation about computer systems (Boyle and Encarnacion, 1994) and the potential benefits have been assessed, summarised and discussed (Eklund and Brusilovsky, 1998, Brusilovsky, 1996).

At the core of personalisation is a *user model* which holds information about the user. Once we have a user model, it makes sense to reuse it. For example, consider a user model that can be used to customise a newspaper. This might model the user's music preferences so that it can include news about music. Another application system presents documents as part of a music appreciation course and this may well need some of the same information about the user's music preferences. Yet another application might deliver a customised music program: it, too, may need the same model of the user's preferences.

This is sensible from the user's point of view. It takes time to build a detailed user model: it would be irritating for a user to have to spend time with each system to develop its own user model independently.

Reuse of a user model is also sensible from the point of view of implementation. It can be costly to construct and maintain user models. It is appealing to amortise some of the costs over different systems, so reducing the cost of the user modelling for a particular interactive system.

This reuse can be achieved with the aid of a *user model server* which can be accessed by various applications. For a model to be reusable, it needs an agreed ontology and representation so that it can be understood and used by different application programs. This means that each application has to use agreed meanings for the aspects modelled.

This paper describes Personis, a user model server. In section 2, we discuss one of the core design requirements on Personis: that it enable the user to scrutinise and control their user model. We call this scrutability. Section 3 gives an overview of the server and its use in a simple recommender application. Section 4 describes the API to the server and Section 5 describes how Personis relates to other work.

2. Scrutability

We introduce the notion of *scrutability* of a user model. This means that the user can scrutinise the model to see what information the system holds about them. In addition, it means that the user can scrutinise the *processes* underlying the user modelling. These include the processes used to collect data about the user. It also includes the processes that made inferences based on that data.

We characterise the issues for scrutability of user models with the following scenario.

Sarah starts Mynews, a personalised electronic 'newspaper'. First she gets the headlines. Today, these include:

- Politician caught out by wife
- New movie release: Flipper saves the day
- New music release: Sydney Symphony Orchestra - Carr-Boyd's Prelude

Mynews is supposed to select just news items most likely to interest this user. It models Sarah's interests so that it can collect reports on issues she will want to know about. Where there are several items about an issue, the user model is supposed to ensure selection of reports Sarah will prefer. For example, she may like a particular critic's music reviews or one syndicate's reports for European news. Now consider how Sarah might scrutinise the operation of Mynews. Some questions she might ask are:

- How did it decide I would be interested in a Flipper movie?
- Or a Carr-Boyd release?
- What or who is Carr-Boyd?
- How can I let it know that I am pleased to hear about the Flipper movie?

- Why did it give me that article about the politician?
- And especially at this time, during a national election campaign?
- Why hasn't it reported the new unemployment statistics that were due for release today?

The first two questions relate to particular parts of the user model representing Sarah's preference for 'Flipper movies' and 'Carr-Boyd music'. Sarah should be able to find information to help answer the questions if she has access to the *value* of that part of the model and a record of the *processes* which contributed to that value. It may be that Sarah has rated several Flipper movies and other animal-centred children's movies, usually giving very positive ratings. She may have also ordered several such movies from an on-line service that provides such information to the user model. If such information affected the values in the user model, Sarah should be able to scrutinise the model to see that this was so.

The third question might be answered within the articles of the newspaper. In general, such questions require an explanation of part of the user model ontology, in this case, a music ontology with 'Carr-Boyd' or 'Australian contemporary art-music composer'. The fourth question relates to user control of the model.

The last three questions are likely to involve a combination of the user model and the mechanisms controlling Mynews. These questions relate to issues of media control.

Perhaps the most compelling reason for scrutable user models is the right of the individual to know what information a system maintains about them. It seems likely that legislation will develop in relation to user models for customisation. A hint of the direction that this might take comes from the 1995 European Privacy Directive (Union, 1995) which makes quite strong demands on information such as a user model data. Amongst its requirements are the need to make available to the user:

- 'the data ... for checking and correction'
- knowledge of 'the recipients or categories of recipients of the data',
- 'the purposes ... for which the data are intended',
- 'right of access to and the right to rectify the data' and,

- ‘an intelligible form of the data undergoing processing and of any available information as to their source’.

This set of requirements matches the sorts of questions we have listed for the scenario above. It is this goal for scrutability that differentiates our approach to building user models and the Personis user model server.

3. Overview of user model server

In this section, we give an overview of the operation of Personis in terms of one example application that we have developed. This is a "Personal Jazz Channel": it is intended to provide the user with their own personalised jazz music programs. So the user who particularly likes certain styles of jazz can ensure that they are offered music that is predominantly within that scope. To do this, the system needs to maintain a model of the user's preferences and dislikes.

To prime the system with a small amount of information about the user's preferences, the start up screen asks the user to rate a small list of key styles, artists and CDs. This screen is shown in Figure 1. It has been widely noted that users are rather impatient about providing information about themselves, as noted especially by researchers in collaborative filtering, such as (Konstan et al, 1997). So we carefully designed this application to request just a small amount of the user's time. They are asked to rate just seven CDs, ten styles and eleven artists. We have chosen these so that they represent a broad range of Jazz music. As can be seen in Figure 1, the user need not rate all of them. It is useful if they simply rate a few that they recognise or feel strongly about. We have used a very simple three-level rating scheme, again to reduce the load on the user.

Once the user has completed as much (or little) of the rating screen as they choose to do, the jazz channel application creates a user model for this user with the preferences recorded. This *evidence* about preferences has been *given* by the user and is accorded a high reliability.

Then the application makes additional inferences about further CDs, styles and artists. These are of lower reliability than the *given evidence* and they make use of the metadata available for each CD. For example, in Figure 1, the user indicated that they like *Sunday at the Village*

Vanguard by the Bill Evans Trio. The system creates several pieces of new *evidence* about preferences: it *infers* the user likes the style that our metadata records against this Bill Evans Trio CD. Since the CD encodes metadata about the style and artists, the Personal Jazz Channel can make inferences based on both of these. From a quite small amount of information from the user, this gives a larger number of pieces of evidence about more aspects in the user model. This set of inferred preferences and dislikes is added to the Personis user model.

This very simple process enables the system to have a quite rich set of preference data from a small set of questions. Each inference has produced a separate piece of evidence which is lodged with the Personis server.



Figure 1. Example rating screen for a personalised jazz channel

At this point, a personal streaming audio channel is created with a mix of tracks conforming to the user's modelled preferences.

At any time, the user is able to examine and change the personal information held by the system by selecting the *Profile* button at the top of the screen. This brings up a screen like that shown in Figure 2. At the left is a list of the two main categories of user model information used by the Personal Jazz Channel. The first of these is simple personal data. The second is the jazz preferences. In this screen, the user has selected the *styles*. So the right part of the screen shows the elements modelled in that category. To keep the screen uncluttered, only elements that have evidence are displayed. The user can see how the

system assesses their preference of this aspect. For example, the figure shows the user does not like the *Cool* style. The user can alter this rating with the arrow buttons, just to the right of the rating *Don't like*. If the user selects the up arrow, the display will increase the rating and create a piece of evidence that the user has *given* the revised positive rating for this style. The question mark to the right of this provides a list of the evidence that is currently held about the user model component for *Cool* style.

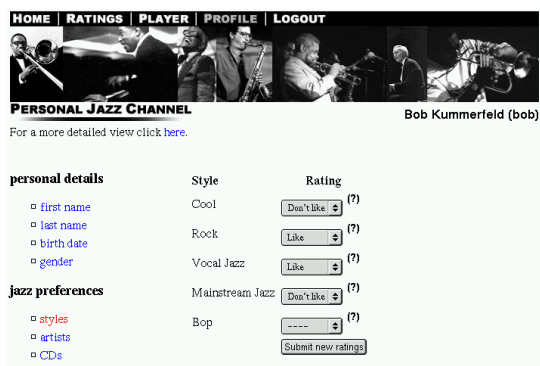


Figure 2. Profile information

In the case of the Personal Jazz Channel application, the profile information presented in Figure 2 is only a part of a larger collection of information stored in the user model. The display is restricted to information relevant to the context of the application.

4. Architecture and Implementation

The Personis user model server is implemented using an object database with remote access provided with remote method calls using the XML-RPC[†] protocol.

The database contains objects representing contexts, components (the *atoms* of a model), evidence for component values and views (a collection of components). The contexts are arranged in a hierarchy for administrative convenience but components from any context can be grouped according to the application. Internally, objects in the database are referenced using an Object Identifier that is resolved to a server address. This allows parts of user models to

[†] www.xmlrpc.org

reside on different servers. It is envisaged that users may keep most of their personal model on a local system under their control but can, if they choose to do so, allow selected information to be stored on servers outside their direct control.

The user model server is designed to be scalable to a large number of users, each with large models. Access to model objects by an application is extremely simple with only one line of program code typically required to acquire a complete set of component values corresponding to a view.

The basic API for the user model server is simple and elegant. Remote method calls using the XML-RPC protocol (and possible future use of SOAP) provides user model access to any application with a client side XML-RPC implementation.

A key function of the user model server is the *resolution* of evidence to determine a value for a component. This process is carried out using user-provided *resolver* program code written in the Python^{††} programming language. Resolvers are executed by the server in a secure execution environment. They are provided with a component and its evidence and are expected to return a resolved value. A default resolver is provided at all times and applications are not required to provide one.

5. Programmer Interface

The client program interface consists of an *access* user model class that provides four methods: *ask*, *tell*, *save* and *close*. When an access object is created the client specifies the user name, password and server. The access object is then used to *ask* for component values from contexts or views, to *tell* evidence for components and to commit the evidence to the model with *save*. The session is concluded with the *close* method.

For example, the following code written in the Python language accesses the user model for "bob" at server "um" and retrieves the "full-name" view that consists of the components "first-name" and "lastname".

```
from PUMS import *

um = access(server="um",user="bob",passwd="*")
name = um.ask(context=["personal"], view="fullname")
```

^{††} www.python.org

```
print "First name is ", name["firstname"]
um.close()
```

Other classes and methods are provided that allow administrative functions to be carried out by appropriately privileged users.

6. Related Work

There has been some research exploring ways to ensure that users can maintain a sense of understanding and control of the personalised systems they use. For example, there has been work to support information seeking within a large documentation system (Hook et al, 1996) where the design aimed to give users a sense of control by making the internal workings of the system ‘transparent’ to the user.

There have been several systems where an inspectable model has been built. For example, HYPADAPTER teaches programming via adaptive ‘documents’ about LISP (Hohl, Bocker, and Gunzenhauser, 1996). Orwant worked in the domain of creating a customised newspaper. He explored several interesting ideas in the area of user modelling that ensures the user is able to determine what is modelled about them (Orwant, 1994). He experimented with mailing a summary of the information in the user model to the user at regular intervals. In one case, he built a user model to represent the user’s activity at the computer and used a graph to display both the user’s recent activity levels as well as their projected activities with time on the x-axis and the amount of activity on the y-axis. He faded the colour of the display to indicate the confidence levels of predictions. He also built models of user behaviour as Markov models as they moved around a building and their movements were tracked with an active badge system. He made displays of these available to the user. A particularly interesting display of user preferences enabled the user to see both their own user model of news preferences as well as the collective models of other *communities*, where these were groups of users defined by some characteristic such as gender.

For teaching documents and systems, there seem to be some additional reasons for making the user model available to the user as it might facilitate user learning, both by creating a good opportunity and framework within which to reflect on their learning and by giving the learner

foundations for planning and managing their learning. In this area, there has been considerable work in making user models visible if not scrutable, such as (Paiva, Self, and Hartley, 1995).

6.1. User model servers

For the most part, work on personalisation has placed the user model within an application. There has been some work on general user modelling software. One early project in this area was GUMS, the Generalised User Modelling System (Finin, 1989, Kass, 1991) managed a collection of databases, each holding the modelling information for an application. Its successor, GUMAC, Generalised User Modelling Architecture (Kass, 1991) was designed to support generation of natural language financial advice. Also built to support natural language was BGP-MS (Belief, Goal and Plan Maintenance System) (Kobsa and Pohl, 1995) and UMT (Brajnik and Tasso, 1994). One of the few user model servers was built by Orwant for the experiments described above (Orwant, 1995). The main work in building generalised tools for modelling user’s knowledge has been in teaching systems. Paiva built TAGUS (Paiva and Self, 1995) for representing students and simulating their reasoning. The problems of dealing with inconsistency in such modelling information were explored in THEMIS (Kono, Ikeda, and Mizoguchi, 1994) and also in SMMS, Student Modelling Maintenance System, (Huang, McCalla, Greer, and Neufeld, 1991).

Several of these systems, including BGP-MS, UMT and TAGUS, had interfaces for the use of the developer to scrutinise the models. However, these were not designed for the user. We believe that if users are to be able to really scrutinise their models, the foundation design of user modelling system must be based on this goal.

In recent reviews of generalised support for user modelling, (Kobsa, 2001, Fink and Kobsa, 2000) it was noted that we have yet to see a user model server that addresses the needs for ensuring the user’s privacy, control and ability to scrutinise their user model and the processes for personalisation. In fact, most personalisation to date has been limited to a single application or web site.

Microsoft has recently announced a system called "Hailstorm"[†] that provides a personal data store

for registered users. The system allows a user to store personal data in a standard form with standard access methods on a central server provided by Microsoft.

Access to the information is provided through a standard set of services using the Simple Object Access Protocol (SOAP) and is based on the earlier "Passport" system for storing user names and passwords.

The data that can be stored in the Hailstorm system is much more general than normally found in a User Model. For example, the Microsoft whitepaper on the system mentions (among others) the following types of data: addressbook, email, diary, documents, device settings.

The system purports to give users control over the data while retaining privacy. However, the architecture presents both a single point of failure and a single point of security vulnerability. Also, the system does not provide any resolution or interfering ability. It is a simple data store.

7. Conclusion

The underlying design of the Personis user model server is based upon the primary requirement that users have access to their user model and control over it. In addition, it has been designed to provide efficient support for user modelling with an elegant but powerful programmer interface. It is novel in its design being explicitly focussed on user control and scrutability.

† <http://www.microsoft.com/net/hailstorm.asp> (visited Sept 2001)

References

- Boyle and Encarnacion, 1994.
C Boyle and A O Encarnacion, "Meta-Doc: an adaptive hypertext reading system," *User Models and User Adapted Interaction*, 4, 1, pp. 1-20 (1994).
- Brajnik and Tasso, 1994.
G Brajnik and C Tasso, "A shell for developing mon-monotonic user modeling systems," *International Journal of Human-Computer Studies*, 40, 1, pp. 36-62 (1994).
- Brusilovsky, 1996.
P Brusilovsky, "Methods and techniques of adaptive hypermedia," *User Modeling and User-Adapted Interaction*, 6, 2-3, pp. 87-129 (1996).
- Brusilovsky, Schwarz, and Weber, 1996.
P Brusilovsky, P Schwarz, and G Weber, "ELM-ART: an intelligent tutoring system on the World Wide Web" in *Proceedings of the Third international Conference on Intelligent Tutoring Systems, ITS-96*, ed. C Frasson, G Gauthier, and A Lesgold, pp. 261-269, Springer, Berlin (1996).
- Eklund and Brusilovsky, 1998.
J Eklund and P Brusilovsky, "The Value of adaptivity in hypermedia learning environments: a short review of empirical evidence," *Proceedings of the the Ninth ACM Conference on Hypertext and Hypermedia*, pp. 11-17, Pittsburgh (1998).
- Finin, 1989.
T W Finin, "GUMS - a general user modeling shell" in *User models in dialog systems*, ed. A Kobsa and W Wahlster, pp. 411-431, Springer-Verlag, Berlin (1989).
- Fink and Kobsa, 2000.
J Fink and A Kobsa, "A Review and Analysis of Commercial User Modeling Servers for Personalization on the World Wide Web," *User Modeling and User-Adapted Interaction - Special Issue on Deployed User Modeling*, 10, 3-4, pp. 209-249 (2000).
- Hohl, Bocker, and Gunzenhauser, 1996.
H Hohl, H-D Bocker, and R Gunzenhauser, "Hypadapter: An Adaptive Hypertext System for Exploratory Learning and Programming," *User Modeling and User-Adapted Interaction*, 6, 2-3 (1996).
- Hook et al, 1996.
K Hook, J Karlgren, A Waern, N Dahlbeck, C G Jansson, and B Lemaire, "A glass box approach to adaptive hypermedia," *User Modeling and User-Adapted Interaction*, 6, 2-3, pp. 157-184 (1996).
- Huang, McCalla, Greer, and Neufeld, 1991.
X Huang, G I McCalla, J E Greer, and E Neufeld, "Revising deductive knowledge and stereotypical knowledge in a student model," *User Modeling and User-Adapted Interaction*, 1, 1, pp. 87-116 (1991).
- Kass, 1991.
R Kass, "Building a user model implicitly from a cooperative advisory dialog," *User Modeling and User-Adapted Interaction*, 1, 3, pp. 203-258 (1991).
- Kobsa, 2001.
A Kobsa, "Generic User Modeling Systems," *User Modeling and User-Adapted Interaction - Ten Year Anniversary Issue*, 11, 1-2, pp. 49-63 (2001).
- Kobsa and Pohl, 1995.
A Kobsa and W Pohl, "The user modeling shell system BGP-MS," *User Modeling and User-Adapted Interaction*, 4, 2, pp. 59-106, Kluwer (1995).
- Kono, Ikeda, and Mizoguchi, 1994.
Y Kono, M Ikeda, and R Mizoguchi, "THEMIS: a nonmonotonic inductive student modeling system," *Journal of Artificial Intelligence in Education*, 5, 3, pp. 371-413 (1994).
- Konstan et al, 1997.
J A Konstan, B N Miller, D Maltz, J L Herlocker, L R Gordon, and J Riedl, "GroupLens: applying collaborative filtering to Usenet news," *Communications of the ACM*, 40, 3, pp. 77-87 (1997).
- Orwant, 1994.
J Orwant, "Apprising the user of user models: Doppelganger's interface," *Proceedings of the Fourth International Conference on User Modeling*, pp. 151

- 156, MITRE, User Modeling Inc, Hyannis, Massachusetts, USA (1994).

Orwant, 1995.

J Orwant, "Heterogenous learning in the Doppelganger user modeling system," *User Modeling and User-Adapted Interaction*, 4, 2, pp. 59-106, Kluwer (1995).

Paiva and Self, 1995.

A Paiva and J Self, "TAGUS - a user and learner modeling workbench," *User Modeling and User-Adapted Interaction*, 4, 3, pp. 197-228, Kluwer (1995).

Paiva, Self, and Hartley, 1995.

A Paiva, J Self, and R Hartley, "Externalising learner models," *Proceedings of World Conference on Artificial Intelligence in Education*, pp. 509 - 516, AACE, Washington DC (1995).

Union, 1995.

The European Parliament and the Council of the European Union, *European Community Directive on Data Protection*, <http://www.doc.gov/ecommerce/eudir.htm> (1995).

Weber and Specht, 1997.

G Weber and M Specht, "User modeling and adaptive navigation support in WWW-based tutoring systems," *User Modeling: Proceedings of the Sixth International Conference*, pp. 289-300, Springer Verlag, Wien, New York (1997).