



The University of Sydney

User Model Agents for Distributed Filtering

Technical Report Number 499

July, 1995

Judy Kay and R J Kummerfeld

ISBN 0 86758 982 5

**Basser Department of Computer Science
University of Sydney NSW 2006**

User model agents for distributed filtering

J. Kay and R. J. Kummerfeld
Department of Computer Science
University of Sydney

Abstract

This paper describes an architecture for intelligent agents derived from user models. The work extends our previous work on user modelling as a basis for various forms of customised systems. We make use of the um toolkit for representation, management and acquisition of the user models.

We have used um-models as the basis for customisation of a coaching system, a personalised hypertext for teaching C and a movie advisor. The system described here extends the application of such user models by constructing intelligent agents from domain-dependent parts of user models and distributing these agents to publication centres where they filter newly published objects.

Keywords

user model, agent, filtering, objects, message delivery

Author's Address

Judy Kay & Bob Kummerfeld
Dept of Computer Science
University of Sydney 2006
AUSTRALIA

judy@cs.su.oz.au
bob@cs.su.oz.au

+61-2-351-3838 (fax)
+61-2-351-3423 (phone)

Introduction

In(Kay 1994) we describe an architecture for the customisation and delivery of multimedia information to user systems. Part of that architecture involves filtering information objects as they are *published* at a publication centre and only forwarding information about “interesting” objects to the user. The filtering of object descriptions is carried out using information about the user from a user model.

Figure 1 shows the context in which our agents operate. Information from the user model, *UM*, is used to create filtering agents. These agents are then distributed to one or more publication centres where they are passed data objects for filtering. The data objects, such as movies or news stories, are created by a *producer* and published at the publication centre. The data objects include descriptors that are examined by the agent. If a data object is judged to be of interest to the user, the agent forwards a description of the object to the user system where the process may be repeated.

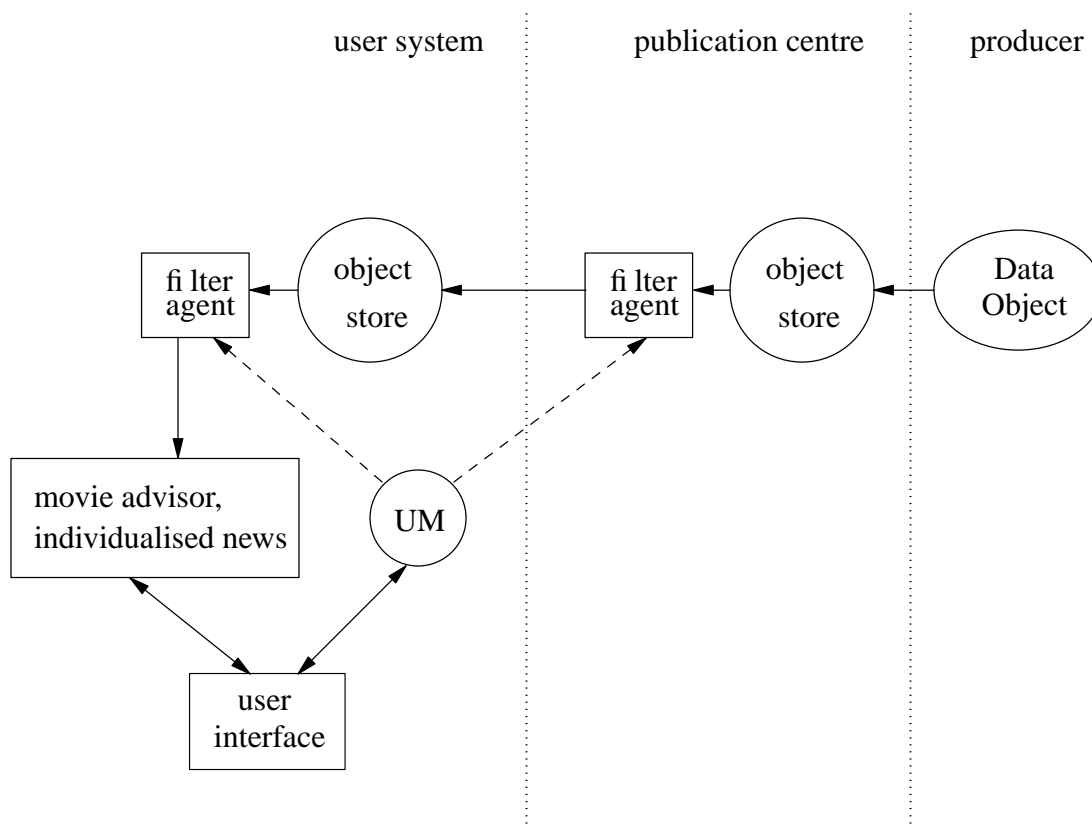


Figure 1. Context for user model based filtering agents

We illustrate this discussion with examples from two domains: an advisor for movies and an individualised newscast system. The goal in the first example is to provide users with movies they are likely to enjoy, using a filter for new movies as they appear. In the second example, news items that are likely to interest the user are used to construct a customised news programme.

The filtering agents are charged with knowledge of the user in a particular domain. So, if announcements of new movies are being filtered, the agent has information about the user's preferences for various genres and subjects. In the case of news stories information about the user's preferences for sport news, local news, financial news, geographical source of news and others are held by the agent.

As the figure shows, there may be several agents involved in the filtering process. One critical division between such agents concerns the location of the agent and the knowledge of the user it holds. Sensitive information, that the users do not want available outside their local environment, is available only to local agents. So, for example, the user who enjoys violent, sexually explicit movies may not want this information to be available to agents that are exported.

Figure 1 also illustrates the process of *publication*. As well as the actual data (or a pointer to it) for the movie or news story, movie and news story objects include a set of descriptors giving information *about* the object. Publication involves a producer of information sending it to a publication centre in the form of an object where it is stored in a database and the descriptors passed to filtering agents.

Other objects may be published that contain further information about a previously published object. For example, they may be movie reviews, ratings by critics or stereotypes of popular preferences.

It is expected that many descriptors will be provided by the creators of the primary objects (movies or news items). For example, the cast, directors, subject and genre classifications, awards and the like are readily available.

However, there are many other descriptors that are not readily available but which could be very useful for filtering and selection. For example, if many people who enjoyed *movie₁* also enjoyed *movie₂* this could be useful for selecting movies where a user is known to have seen and enjoyed *movie₁*. Or, it may be that married men in their thirties tend to like the movies that are regarded as good by a certain critic. Another goal of our system is to gather this type of information from the agents as they do their work.

The actual filtering operation in our initial system is a simple pattern match of the descriptor names and their values.

User access and control with agents

The user modelling component of the system makes use of the um toolkit(Kay to appear). We want our agents to maintain the um philosophy of user control and access. As part of this, they need to be able to justify their actions.

This means that they should be able to explain why some *movie_i* was or was *not* offered to the user. The answer involves three main elements: the descriptors for the movie, the mechanism that uses these for filtering and the user model, as it was at the time the

filtering was done.

For example, if the agent's record of preferred actors was the only factor in making the selection, a record of this and the relevant descriptors of this movie can be provided. This can be done by turning on such logging or by asking the agent via a query to justify its selection, as performed at the time of the actual selection.

This information can be presented to the user in a meaningful way. Our standard viewer (Cook 1994) for the user model can be used to display the relevant parts of it.

A separate display is needed for the movie descriptors: the same interface is appropriate for browsing the descriptions for the recommended movies.

The third part, the explanation for the filtering process is one of the elements that we require when the agent is constructed. For simple mechanisms, like our current pattern matching, it is adequate to provide text describing the process.

So far we have described the way we provide the user with the information needed to understand how a particular filtering action was done. If the user is to be in control of the process they must also be able to alter any of its three elements.

The user model can be amended via its viewer interface. The other two parts are less straightforward. Altering the filtering mechanism currently requires amending the code that is carried by the agent. This is clearly infeasible for most users. In the future we see it as desirable to build interfaces that support some customisation of the filtering mechanism.

The last element in the process, the movie descriptors, is more problematical. Users may disagree with these in several ways: they may consider that a descriptor does not apply or that its weighting is incorrect or they may consider that an important descriptor is missing. In the current implementation, the user can only indicate this type of problem by some informal mechanism, like mailing the producer of the information. In the longer term, we want to ensure that a producer can be automatically collect this type of feedback. It is a potentially invaluable source of improved information about the movies but it needs to be managed with care to ensure that malicious and frivolous users do not corrupt the movie descriptors.

Architecture

The architecture of the system is illustrated in Figure 2, which is a refinement of Figure 1.

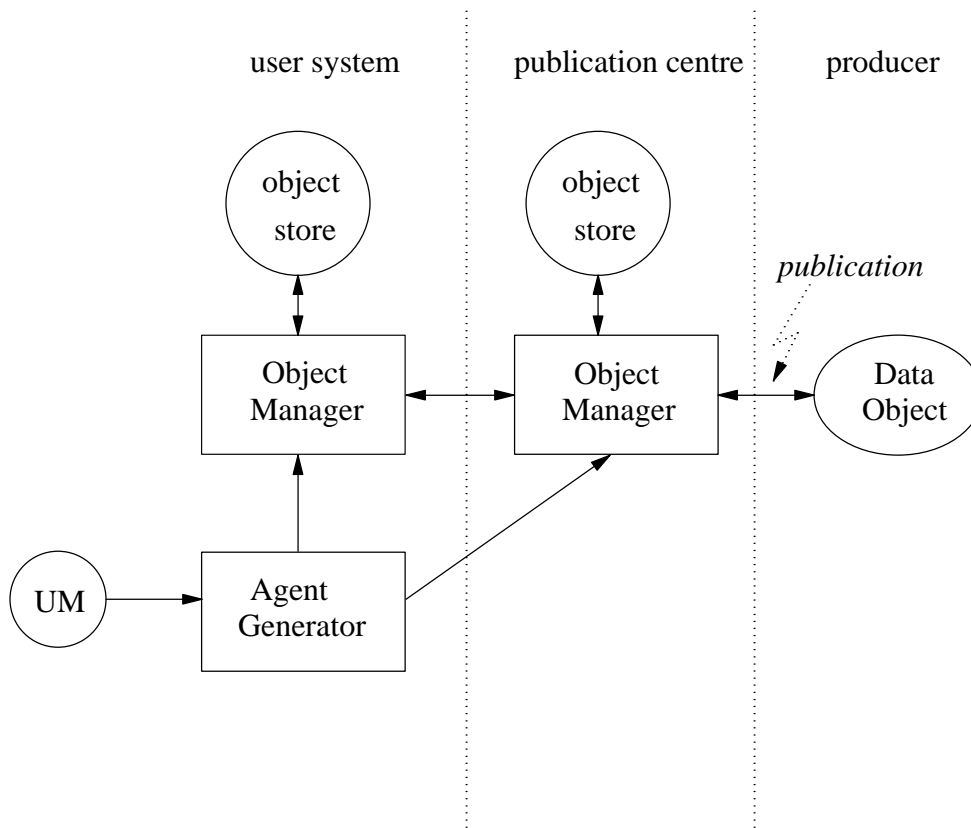


Figure 2. Data and agent object architecture

Objects

Data objects and agents are represented in the system as a set of descriptors for the object followed by the data for the object.

The descriptors are stored as a set of attribute/value pairs in plain ASCII text. All objects have ID and TYPE attributes with the ID value being a globally unique identifier for the object and the TYPE value giving the type of the object. The TYPE value uses the MIME (Borenstein 1993 and Postel 1994) structure of *type/sub-type* and registered MIME types are used where possible (for example 'audio/basic' for a radio program). The object data can follow the description in the same data stream or the description may contain a pointer to the data. This format for objects is very similar to the SOIF (Summary Object Interchange Form) used by Harvest (Bowman 1994) with the exception that we allow the object data to be carried with the description.

For example, the following object is the movie 'Alien':

ID: M0000010
TYPE: MOVIE/MPEG
TI: ALIEN
AB: Three crew members of the space-freighter Nostromo are sent to investigate ...
...etc...
DR: Ridley Scott
CA: Ripley: Sigourney Weaver
CA: Dallas: Tom Skerritt
DE: Monsters 8
DE: Science Fiction 8
DE: Action 9
RD: 1979
RT: 105 minutes

...MPEG movie data at this point...

In this example, the actual data for the movie follows the descriptors. Alternatively, the descriptors may contain a pointer, in the form of a Uniform Resource Identifier (URI), to the actual data.

Operation

An information producer creates an object containing the raw data and descriptors and sends to a publication centre where it is added to the database with the publish operation.

Requests may be sent to the publication centre to publish, replace, delete or retrieve a copy of an object.

An agent generator program examines the user model (UM) and generates an agent object which it publishes at a publication centre.

An agent is written in the Python scripting language (Rossum 1991). A "safe" version of the Python execution environment is being developed to provide some measure of security by restricting the operations that can be performed by the Python script. Most file operations have been removed and a message sending primitive added.

Newly published objects are passed to registered agents which can examine the descriptors and data. An agent may then forward a copy of the complete object, or the descriptors alone to the user. Agents may also send email to any user or a message, in the form of an object, to any other agent.

An example of an agent object:

```
ID: agent86
TYPE: agent/python

#!/usr/local/bin/python
import sys
import posix
import regrab
import agent

f = open(sys.argv[1], 'r')
m = agent.Message(f)

if m.has_key('Type') and m['TYPE'] == 'radio':

    ....etc....
```

The code of the agent script, written in the python language, is contained in the data part of the object. The above example shows a fragment of the script for the object 'agent86'.

Object Transport

Objects are moved from machine to machine using the TCPmsg (Kummerfeld 1992) message transfer system. This uses a simple protocol above a TCP/IP connection to carry arbitrary binary messages of any length between hosts.

Object Management

When an object is received at a machine, TCPmsg will pass it to an object manager program along with a set of operations and parameters.

The object manager recognises five operations: publish, replace, delete, retrieve and sendto.

The publish operation involves the object manager storing the object in a database, the replace operation replaces the object and the delete operation deletes an object from the database.

The publish and replace operations have an owner identifier as the single parameter. The object to be published or replaced carries its globally unique ID in the description part. The delete operation requires two parameters: the ID of the object to be deleted and an owner identifier. The retrieve operation has the ID of the required object as a single parameter.

The sendto operation requires two parameters: a regular expression describing what objects are to be sent the incoming object and the method (function) to be invoked within the agent script. The object manager searches the database for all agent objects that match the regular expression and have the required method. For each of the agent objects that match, the nominated function is then invoked and is passed the incoming object.

Movie advisor and individualised news system

In our prototype movie advisor and the individualised news service, descriptors for newly released items are created by hand and combined with the data for the item to form an object which is then published at a publication centre. We also use the system to publish news, current affairs and infotainment radio programmes. In other work (Kay 1994) we discuss a multicast technique for distribution of objects to publication centres.

Publication involves invoking the *publish* operation and the *sendto* operation to send the movie object to all agent objects that have the *filter* method. In our first implementation the filter method does a simple pattern match on descriptors to determine if the user is interested in the object. If the user is found to be interested, the agent invokes TCPmsg to publish a copy of the object at the user's home system. This further distribution of the object may involve additional filtering by agents registered at the user's home system where more information about the user may be available.

Conclusion and future work

The principal concern of this work is to extend the previous use of a user model to the creation of agents that are substantially driven by user modelling information.

We have constructed an object distribution and management system and represented agents and data items as objects. The data objects to be filtered contain a description of the data as well as the data itself or a pointer to the data. Filter agent objects contain executable scripts and a list of methods provided by the agent. As they are published, data objects are passed to filter agents which may choose to pass on the description of the data object to a user system.

One benefit of the agent approach to the task of remote filtering is that the user system determines how the filtering is carried out since it has complete control over the programming of the agent. It also allows filtering to be carried out remotely and doesn't require all items or item descriptions to be forwarded to the user system. Filtering can be carried out in several stages allowing users to keep details of their personal preferences at the home system.

An important issue for user model based agents that we are now exploring is the question of how to split responsibilities between agents, especially if that interacts with the environment in complex ways. In particular, where parts of the user model are to leave the user's own machine, it is important that the user appreciates just what information about them is being exported and is helped to understand potential implications of its use externally.

We consider that it is very difficult to guarantee protection of this information. This is precisely why we have designed an architecture that uses multi-level filtering with one layer being on the user's machine. We consider that sensitive information should be kept there.

We are also investigating another class of agents that analyse filtering agents to build up useful stereotype information for use by filtering agents. This should be able to identify patterns of user preferences, or user *communities* (Orwant, 1994 user modeling).

Our approach to filtering combines a number of elements: user models and associated tools; a flexible and simple object representation of agents that uses a secure scripting language to define its functionality; and transport based on TCPmsg. The whole system maintains the philosophy that users should be able to understand and control agents that act on their behalf.

References

Borenstein 1993

N Borenstein and N Freed, *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, RFC 1521 1993 <<ftp://ds.internic.net/rfc/rfc1521.txt>>

Bowman 1994

C M Bowman, P B Danzig, D R Hardy, U Manber, and M F Schwartz, *Harvest: a scalable customizable discovery and access system*, University of Colorado-Boulder Tech Report CU-CS-732-94 1994

Cook 1994

R Cook and J Kay, *The justified user model: a viewable, explained user model*, UM94: User Modeling Conference 1994 also Sydney Univ. CS Dept. Technical Report 483 (ISBN 0 86758 922 1)

Kay 1994

J Kay and R J Kummerfeld, *Customization and Delivery of Multimedia Information*, Proc Multicomm 94, Vancouver 1994
<<http://www.cs.su.oz.au/~bob/CandD.html>>

Kay to appear

J Kay, *The um toolkit for cooperative user modelling*, User Modeling and User Adapted Interaction Kluwer 3 to appear and as SSRG Report 94/3/36.3

Kummerfeld 1992

R J Kummerfeld and P R D Lauder, *The IPSendfile Project*, Networkshop 92, Brisbane 1992 <<http://www.cs.su.oz.au/~bob/tcpmsg.html>>

Postel 1994

J Postel, *Media Type Registration Procedure*, RFC 1590 1994
<<ftp://ds.internic.net/rfc/rfc1590.txt>>

Rossum 1991

Guido van Rossum and Jelke de Boer, *Interactively Testing Remote Servers Using the Python Programming Language*, CWI Quarterly 4 1991