



The University of Sydney

**The justified user model:
A viewable explained user model**

Technical Report Number 483

June 1994

Ronny Cook and Judy Kay

ISBN 0 86758 922 1

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

The justified user model: a viewable, explained user model*

R Cook and J Kay †

Basser Department of Computer Science
University of Sydney
ronny@tmx.mhs.oz.au,judy@cs.su.oz.au

Abstract

This paper describes our experiences with giving users access to the system's model of them. This access is supported by a number of tools that enable a user to see various forms of information: an overview display serves as a navigation tool as well as a helpful summary; the main model viewer program enables the user to see the detailed model the system holds; the *justification* part explains the system's reasoning about the user model; the *explanation* subsystem provides a tailorable glossary of the terms used in the user model description; and the *change* facility allows the user to add their view of the correct value of components of the model.

We outline the series of studies we have conducted to learn how users respond to our visual displays of a user model. These indicate a new purpose for a user model as a useful communication tool that enables users to find information that interests them as well as to communicate the way that they would like to be modelled.

We also discuss some issues that arise from making the user model accessible. These affect the way that the programmer sees their relationship to the user. They also highlight the need to address different interpretations of a user model.

Introduction

The user model is becoming increasingly important in a number of classes of systems that are currently of considerable interest: customised documentation, teaching systems, information filtering and other tailored interfaces.

There are many potential benefits from making a

user model accessible to the user it describes. This is on the grounds of the user's right to access information about themselves, the accountability it enforces on the programmer creating and using the user model and the benefit of having the user verify or correct the information in the user model. In addition, there appear to be educational benefits (Crawford and Kay 1992, Crawford and Kay 1993). Others have argued the importance of making complex systems more comprehensible (for example, Fisher 1991; Maass 1983).

This paper describes viewer programs that enable the user to see their user model, expressed in the um representation (Kay 1990). From the very first, um was designed to make the user model accessible to the user. The viewers described here enhance *practical* accessibility because they make it far easier to browse through the model and to appreciate its overall structure and values.

Our development of the accessible user model and associated viewer programs has been in the context of a number of projects. Our most extensive work has involved systems that support learning of the sam text editor (Pike, 1987). The primary goal in this work has been to enable users to become more effective sam users. We have created several coaching systems, each based on a different approach to teaching about sam (Parandeh-Gheibi and Kay 1993). Another series of coaching experiments have taught users about Unix (Butler, 1992). We have also been studying various models of usage and learning of sam (Benyon, Kay and Thomas 1992).

We have become particularly interested in the way that a viewable user model can, itself, be a useful tool. So, for example, one of our sam coaching systems allowed users access to the viewer programs. We expected that such access would enable these users to approach their learning about sam differently.

* This work is supported by Telecom Australia Grant Reference Y05/04/34 and BLO/02/02/89

† R Cook is currently at Message Handling Systems, Newtown, Australia
To appear UM94, 4th User Modeling Conference, Cape Code, USA>

This paper begins with a description of the viewer programs. Then we report on our evaluation experiments and discuss our findings.

Overview of the user model

Figure 1 shows a user's model as displayed by `quick_view` (for **quick viewer**). Parts of the model are "folded" into single nodes to reduce window clutter. `quick_view` can also be used to view selected parts of a user model by invoking it with a specification of the parts to be viewed.

The um form of a user model is a directed acyclic graph. This is presented to the user as a tree. Non-leaf nodes are called *partial models*. They are shown as circles. These represent a meaningful grouping of related aspects of the user model. The leaves of the `quick_view` tree are the *components* of the user model and these are shown as shapes other than a circle.

Different shapes are used to indicate the type of the component. Squares are used to show *knowledge* components. Diamonds are *beliefs*, aspects that the user may believe true but the system designer (and hence the system) considers untrue. These may well be misconceptions. They may, equally, represent alternate views which conflict with those of the main model being displayed. The crosses are *characteristics* and other properties of the user for which true/false values may not be appropriate.

Most of the model shown in Figure 1 is devoted to knowledge-components. One belief-component represents the view, held by many users, that killing the `sam`-window is a good way to quit. (This means the user misses potentially helpful warnings.) The bottom of Figure 1 has several characteristics-components such as the representation of the user's apparent typing speed (`wpm_info`).

In addition, `quick_view` displays the truth value of the nodes and leaves. For all but belief-components, true is indicated by black, false by white. A circle in a circle (and corresponding nested shapes) indicates situations where the system could not determine the truth of the component. We describe the way that the truth is determined later.

One role of `quick_view` is to give the user an overview of a large part of the user model structure 'relevant' at this point. Now the definition of 'relevance' has two parts. First, the root of the displayed tree defines the 'relevant' context. In the case of Figure 1, this is the whole model for this user (starting at the 'root'). We could have displayed just the part of the model starting, for example, at `sam`.

The second part of 'relevance' concerns the selection of the actual nodes to be displayed. One option

is to display the whole tree so that the user can, indeed, get an overview of the part of the model being viewed. An alternative is to try to determine the most useful, and so relevant, parts.

There are many reasons for avoiding the display of the whole partial model subtree. If the starting node given to `quick_view` has a large subtree with many components, it may be infeasible to display all of them as the screen would be too cluttered or the print too small to be readable.

Even if all the components can be displayed so that they are clear, it may be that many have the same value. Then the `quick_view` display can be clearer and more meaningful if we collapse any partial model that has all its components of the same value. Where all the components are true, this constitutes showing the user that the more general notion is true.

In designing `quick_view`, we were even more concerned about the opposite situation where the user knew none of the components in various partial models. This is the case when a beginner has only made small use of `sam` and knows little about it. We were concerned that a screen showing large numbers of unknown nodes would be discouraging. Moreover, a user who knows only the simpler aspects of `sam` is unlikely to be interested in sophisticated features.

We also allow the user to override the default 'relevance' definition. When the user clicks on a node whose subnodes are not displayed, the next level down the tree is displayed. Conversely, the user can collapse the subnodes by clicking on a node.

A second role of `quick_view` is as a navigation aid when viewing a user model in detail. Normally the user will investigate the details of the model using a viewer like `um_view`, described below. Then `quick_view` keeps the context available.

Detailed model viewer

The main viewer interface, `um_view`, allows the user to traverse their user model. A startup screen displays the root and immediate child nodes of Figure 1. The user can select nodes that are leaves in the visible tree, expanding that part of the tree until the leaf components are available as is the case in the example of Figure 2. (As one would expect, the converse navigation applies and leaves can be collapsed by clicking on their parent node.) Rather like the `quick_view` display, `um_view` gives the user the value of each node displayed. However, it offers much more at the component's nodes. Here it provides a pop up menu from which the user can select three main actions: *justify* the value of the component; *alter* the truth value of the component and *explain* the meaning and purpose of this part of the model.

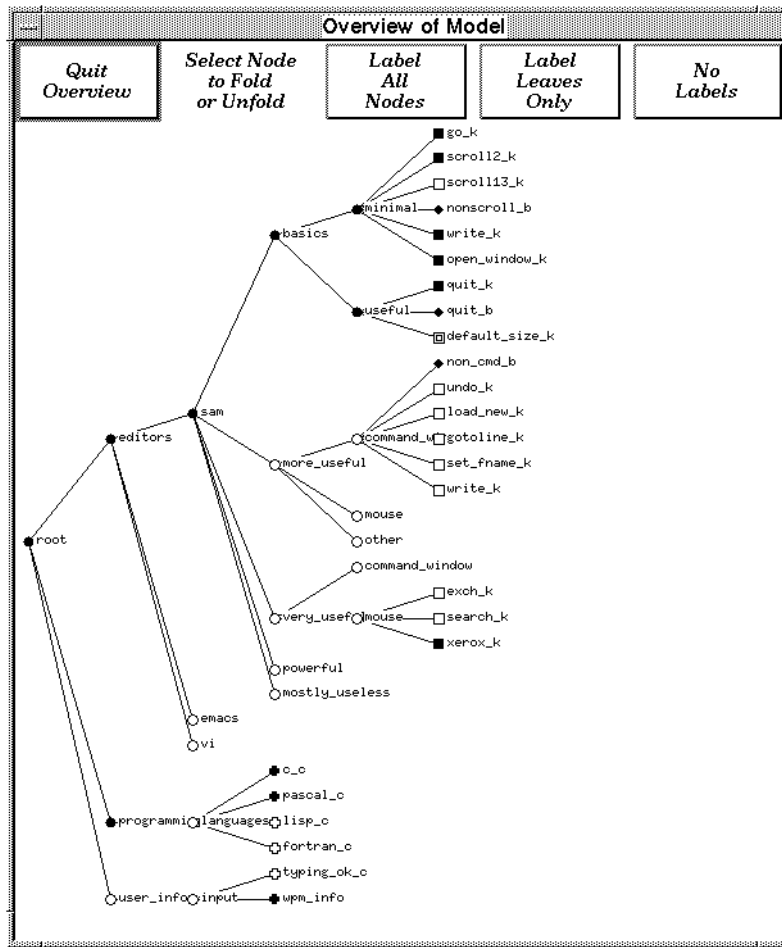


Figure 1. Example quick_view display

Justifications Selecting 'justify' gives a display of evidence recorded in the user model for this component. The *supporting* evidence is that which suggests the component is true and the *negating* evidence suggests it is false. This is a direct presentation of the contents of this part of the user model.

Each line of the justification display summarises one piece of evidence: the evidence type (for example, Supporting when it indicates the truth of the component); information about the source of the evidence; and the time that the evidence was added to the model.

For example, an item of evidence may support the truth of the user knowing how to use a particular command and the user may themselves be the source of that evidence via an interview program.

When this list of evidence is displayed, the user can select an evidence item from the list. Then *um_view* displays a description of the program that created the evidence item. For

example, much of the evidence about user's knowledge of *sam* comes from analysis of the *sam-monitor* log. The various methods used to form conclusions about the user's knowledge are explained.

A rather more interesting example of such evidence is where *um_view* explains a Rule that was the source of evidence. For example, we have a complex rule for determining the user's knowledge of a command, like *xerox*, which creates multiple windows on a file. The rule has to deal with the fact that it is easy to make an accidental selection from a mouse menu so there are other factors like the user's experience and the context of its invocation.

Altering a value This action allows the user to set the value of a component. The actual effect of this is to add a new piece of evidence.

Explanations Explanations of each component

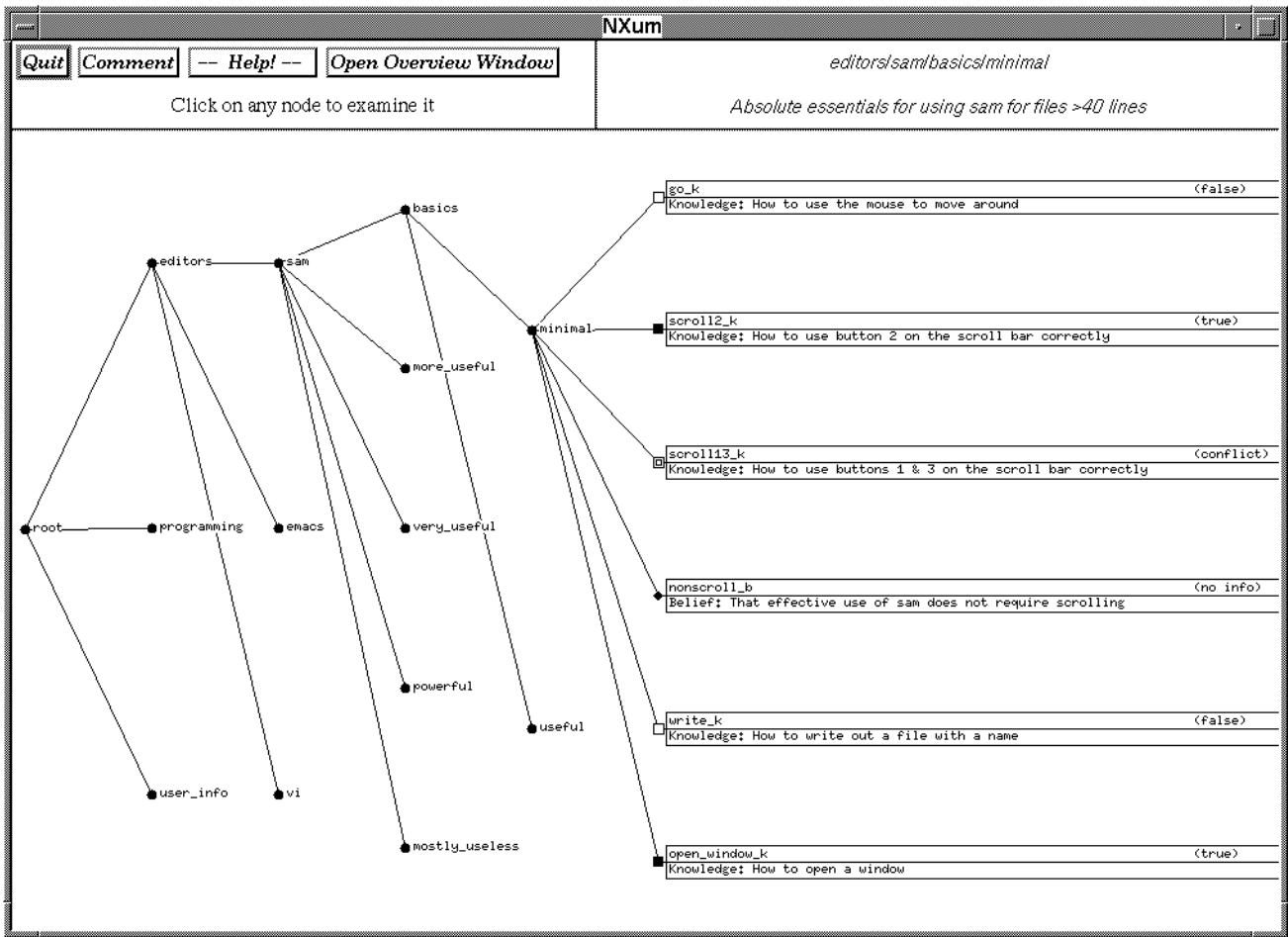


Figure 2. Leaf level screen dump of um_view

in the model are available from the viewer. The text displayed is varied on the basis of the user's *expertise*, as stored in this part of the user's model. (This is viewable like any other component and the user can alter it as they wish.)

If the model does not include this expertise indicator, the explanation generator uses defaults. These make the explanation match the difficulty classification of the component: for example, basic aspects are described in terms suited for the novice and very sophisticated features are in terms appropriate to an expert reader.

Summary of changes to user model during the viewing process In the process of viewing a user model, several changes may be made to the model. The most obvious is when the user specifies the value for a model component. Any such change adds a *given* evidence item to the list of justifications for the component. As such additions are weighted very highly, this effectively sets the value for a component.

Another kind of change occurs when the user requests an explanation. In this case, a *told* evidence item is added, which has a very low weighting. The weighting is sufficient to push a model with no evidence (tentatively *false*) to the *uncertain* state.

Evaluations

We have done two main forms of evaluations. Earliest was a series of in-depth interviews and observations of users of the viewer. The second form involved using the viewer programs as part of experiments in coaching users about *sam* and *unix*.

The interviews involved eight regular users of *sam* at the end of their first year of using it. Seven of them were selected from the first year computer science class and one was a member of administrative staff.

All were quite able to use the viewer for

the purpose of assessing the accuracy of the **sam** model. This gives us confidence in the usability of the interface for regular computer users.

Several users showed considerable interest in and concern about the details of the models.

After these interviews had helped us refine the interface, we used **um_view** as part of a series of coaching systems, for **sam** and Unix. Those studies (Butler 1992) (Parandeh-Gheibi and Kay 1993) used several different approaches to coaching students. Some of the groups were given access to their user model and invited to explore it. They had no assistance other than email telling them how to start the viewer.

This represents a study of users in a real use situation rather than a carefully controlled environment as in our earlier series of observations and interviews.

In all, forty-six users made any use of the viewer at all. Of these, fourteen users made substantial use of the viewer.

For these heavier users, the picture that emerges is of substantial use of all the main facilities of the viewer. This is encouraging from the perspective of usability of the interface.

Also encouraging is that fourteen users, 30% of the whole group, made substantial use of the viewer. To put this in perspective, students were invited to use the viewer in unsolicited email. Use was completely discretionary and competed against a heavy load of work for credit. This made the students highly 'discretionary' users.

Discussion

The primary concern of this work is to make user models more accessible to the users they model. The underlying **um** models were designed to be available to users. The two viewer programs make them easier to understand. Firstly, **quick_view** gives an overview that would be difficult for a user to develop by simply studying each part of the user model. Our users' comments made it clear that they were impressed with summary powers of this style of display.

At the same time, the information provided by **um_view** makes the details of the model more meaningful. This is partly because the user defines the focus as well as the logical organisation and the provision of tailored explanations. In particular, the explanations enable the user to appreciate what is being modelled and how the values came to be there.

The substantial exploration of these by our users indicates that at least this population has many users who are really interested in exploring this level of detail in their user model.

Unsolicited comments offered by one of these users indicates that they used the viewer beyond the time of the study because they found it a helpful way to learn about **sam** by reading the component explanations. The user model enabled this person to identify components at the frontier of their knowledge and to explore the domain deciding what to learn, then learning it.

An **um_view** display currently represents just one conceptualisation of a domain. For example, the **sam** part of the model of Figure 1 is organised in terms of the utility of each aspect of **sam** for typical editing tasks. Many other organisations are possible.

The philosophy of **um** is that it decouples the user model from the reasoning processes that constructed it as well as those that make use of it. This means that it is the responsibility of the programs which create the user model to combine components to form a partial model that is appropriate to the needs of that consumer program. So, for example, the model may be a pure *overlay* model that relates the user's knowledge to that of an expert. Equally, it can be another arbitrary form, with misconceptions or alternate frameworks being modelled.

There is then a critical interaction between this philosophy and that of the viewer to make the model truly accessible to the user. On the one hand, the user model is decoupled from its creators and interpreters. On the other hand, a user can only gain true understanding of the model if they know about these programs. The viewer programs use a default structuring of the models.

Although the viewer was intended primarily to display user models, it also imposes some constraints on them. When one designs a model knowing that the user can see it, one is obliged to think rather differently from the programmer who encapsulates assumptions about the user in their code. This imposes a level of accountability on the programmer. It also makes the programmer think about in terms that the user will find understandable and acceptable. More than this, we find ourselves trying to see the model from the user's point of view.

Conclusion

From our current experience, the viewer interface appears to be successful. Users found it easy to use and intuitive. A substantial minority of users also found the user model interesting enough to explore quite thoroughly.

The viewer appears to serve an important role in helping users decide what they want to learn and the overall size of the domain. This suggests that the viewer has the potential to serve a useful role as a support for documentation and other information.

In effect, the viewer enables the user model to become a learning tool. This constitutes a new purpose for the model. It no longer serves as a mere back-end to programs that try to customise their interaction with the user: it takes on a first order role of its own.

Most critically, the current experience gives us growing confidence in the practicality of making user models accessible.

Our current work is to extend the explanation facilities so that they can be tailored further. Another critical area of development involves the extension of the viewer to operate within the context of particular consumers. In a similar vein, we see considerable promise in exploring the use of the viewer with a variety of partial model structures that enable the user to take differing views of the domain and to see their user model projected onto those views.

Acknowledgements

Richard Thomas set up the `sam` monitoring software. Kathryn Crawford took most of the interviews. Both contributed in formative discussions. David Benyon contributed to the early design of the `sam` project.

References

Benyon, D. Kay, J. and Thomas, R. 1992. Building user models of editor usage: In Proc. UM92 - Third Intl Workshop on User Modeling, Andre E., Cohen, R., Graf, W., Kass, B., Paris, C., and Wahlster, W. eds. Schloss Dagstuhl, Wadern, Germany: IBFI (Intl Conf and Research Center for Computer Science). 113-132.

Butler, G. 1992. Unix coach. Hons thesis, Basser Dept of Computer Science, University of Sydney.

Cook, R., and Kay, J. 1993. Tools for viewing um user models, SSRG 93/3/50.1. Dept of Computer Science, University of Sydney.

Crawford, K. and Kay, J. 1992. Shaping learning approaches with intelligent learning systems: In Proc Intl Conf for Technology in Education, France, 1472-6.

Crawford, K. and Kay, J. 1993. Metacognitive processes and learning with intelligent educational systems: In *Cognitive Science Down Under*, Slezak, P. ed. Ablex. 63-77.

Fisher, G. 1991. The importance of models in making complex systems comprehensible In *Mental models and Human-computer Interaction 2*, Tauber, M. J. and Ackerman, D. eds. Elsevier. 23-33.

Kay, J. 1991. um: a user modelling toolkit: In Proc Second Intl User Modeling Workshop, Hawaii, 11pp.

Maass, S. 1983. Why systems transparency?: In *The Psychology of Computer Use*, Green, T. R. G., Payne, S. J. and Veer, G. C. V. eds. Academic Press, 19-28.

Parandeh Gheibi, N. and Kay, J. 1993. Supporting a coaching system with viewable learner models, In Proc. Intl Conf for Computers Computer Technologies in Education, Petrushin, V and Dovgiallo, A. eds. Kiev, Ukraine, 140-141.

Pike, R. 1987. The text editor, sam. *Software Practice and Experience* 17(11):813-845.