An Architecture For Adaptive Interfaces

Anthony Lennard and Alan Parkes

1995

AAI/AI-ED Technical Report No.110

Computing Department
Lancaster University
Lancaster
LA1 4YR
UK

aai@comp.lancs.ac.uk
http://www.lancs.ac.uk/computing/research/aai-aied/
ftp.comp.lancs.ac.uk (148.88.8.9) /pub/aai/

An Architecture For Adaptive Interfaces

Anthony Lennard and Alan Parkes
Computing Department
Lancaster University
Lancaster, UK
LA1 4YR
Tel: +44 (0)1524 65201 extn 4899
E-mail: anl@comp.lancs.ac.uk

ABSTRACT

If the interface of a system is to be made completely adaptable to the current user, then the interface and application have to be completely separate. Furthermore, if the model of the interface is independent of both the application and the operating system, then the user will be able to have their own model of the interface, which when added to the application, creates the working system. In this paper, we put forward an architecture to facilitate this goal.

INTRODUCTION

An adaptive system is a system "which automatically acquires knowledge about its users, updates this knowledge over time, and uses the knowledge to adapt to the users' requirements" [11]. The key part of this definition is that adaption is an automatic process, instigated by the system as opposed to the user. If a system is to change its interface in fundamental ways (for example, more than simply changing window sizes or colours), then the system will have to be based on an architecture where the model of the interface is completely separate from the application. Furthermore, if the model of the interface is independent of both application and operating system, then the user should be able to use their model of the interface in all applications they encounter. In this paper we define such an architecture.

Rationale for Adaptive Interfaces

The standard argument for adaptive interfaces is that end users are heterogeneous, and would thus be best served if given an interface that matched their capabilities. For example, many studies have concluded that users with a poor spatial ability would be best served with a highly structured dialogue based interface (where the system always takes the initiative), rather than, say, a command based interface [6, 10, 13, 15].

The main disadvantage of adaptive interfaces that has been put forward is that the maxim of consistency is not being upheld if an interface is constantly changing to meet the demands of the user. In the worst case, the problem of `hunting' occurs [9], where as soon as the user becomes proficient in using an interface, the interface changes and the whole learning process starts again.

However, we would argue that an adaptive interface should *increase* consistency between applications. This is because, for even the simplest of tasks, different interfaces provide different procedures to carry out the same task. For example, consider the selection of an item from a menu, using the MS Windows and Macintosh operating systems. The exact procedures are given in Figure 1. There is no problem if a Macintosh user uses a Windows program, as the procedure acquired for the Macintosh will work in Windows. But if a Windows user selects an item from a menu using Method B, then tries to use the same method on a Macintosh, no item would be selectable (as the menu would disappear when the mouse button was released at stage 2). The user would then, through trial and error, asking someone, or

looking through a user guide (if one could be found), have to learn a new procedure to select a menu item.

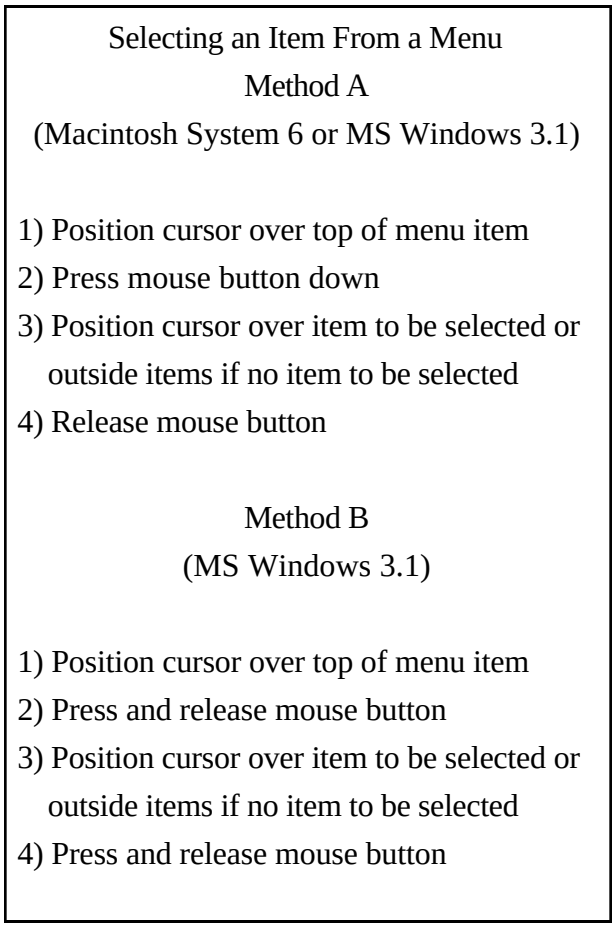

Figure 1 - Method For Selecting An Item From A Menu Using the Macintosh And MS Windows Operating Systems

The notion of consistency between interfaces can be taken further. If, for example, the user can go to the previous word in a document, say by pressing `control p', in the word processor that they normally use, it would surely be advantageous if the same command worked when typing some text in a drawing package. Similar points can be made for how icons are presented, the style of dialogue used, the layout of the screen, and so on. For the user to presented with a consistent interface, an application must be completely separate from its interface, and the user must have their own model of the interface, which is then added to the application, to create the working system.

Current Adaptive Systems

Adaptive systems generally fall into two categories. Firstly, some systems have a choice of interfaces, with the system choosing the most appropriate interface based on what it knows of the user. For example, in the MONITOR system [1, 2], either a menu- or command- based interface is presented to the user, based on their computing experience and the number of errors they have made.

In the second category of adaptive systems, the interface essentially stays the same, although changes in the level of functionality, the appearance, the level of help, etc., are possible. For example, in the object-oriented drawing package AIDA [5], extra icons are created if the system detects that the user repeatedly draws a shape for which no icon is available. In Berthome-Montoy's menu based system [3], the system adapts to the:

1) *User's Habits and Experiences.* By remembering the layout of the screen, and giving an appropriate dialogue to enter parameters based on the user's experience with a command;
2) *User's Task.* By detecting sequences of commands (then offering a new command to replace the sequence), and deducing appropriate default values for the commands;
3) *User's Knowledge.* So appropriate help can be offered, based on the users' experience, current plans and goals.

CURRENT ARCHITECTURES FOR ADAPTIVE INTERFACES

The majority of adaptive interfaces are based on an architecture consisting of three components, namely:

1) *User Model.* Describing the user of the system.
2) *Domain model.* Describing the parts of the interface which can be adapted, with alternative options being given. This is usually in the form of a list of available options. For example, in the MONITOR system, the domain model contains a flag indicating whether a command or menu based interface is being used.
3) *Adaptive Mechanism.* The interface adapts using adaptive mechanisms based on the user and domain models.

The fundamental problem with this approach to adaptive systems is that the system, or more specifically the *interface* to the system, can only adapt in ways that are explicitly specified and described in the domain model, thus limiting the range of adaptions that the system can perform. If the interface to the system is to adapt in a more significant way, then the entire interface to the system will have to be specified in the equivalent of the domain model. If the interface was described in an operating system and interface independent way, then the user's preferred interface could simply be added to the application to create the working system.

Underlying Rationale for the Architecture

The overall objective of our proposed system, is that it presents a user with a consistent interface for each and every application that they use regardless of the type of application and the operating system on which the application runs. The underlying principle is that the majority of applications share most of their interfaces, if you define the interfaces in terms of sufficiently detailed primitive operations, such as selecting an item from a menu, typing a piece of text, or selecting an icon. If each part of the interface is specified through a dialogue model in a device and application independent way, then by porting the dialogue models between applications, the user will always be presented with a consistent interface.

The most well established model for describing an interface is the Seeheim Model [7], which has been developed from research into User Interface Management Systems.

The Seeheim Model

The Seeheim Model (Figure 2), divides the interface into three parts, namely:

1) *Presentation Component.* Responsible for screen management, information display, interaction techniques and lexical feedback from the interface. It is the

only component of the architecture which is device specific, so if an application was moved to a different platform, only the presentation component would need to be changed.

2) *Dialogue Control Component.* Manages the dialogue between the user and the applications. This is normally represented by a state transition diagram, context free grammar or an event-based technique.

3) *Application Interface Model.* Describes the user interfaces' view of the application. It has two main parts:

   i) A description of the application's routines and data structures.
   ii) A description of how the user interface communicates with the system. The three possible interaction modes are user initiated, system initiated and mixed initiative.

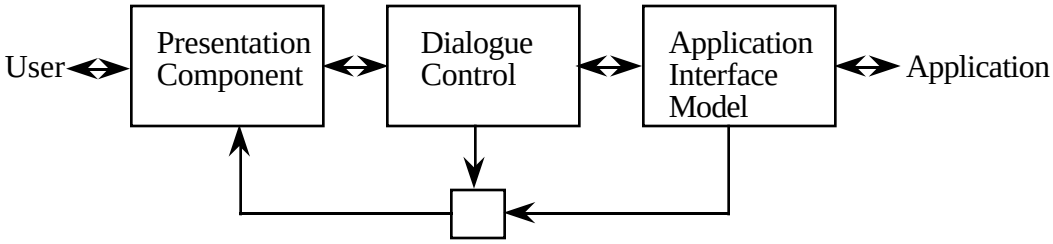

Figure 2 - The Seeheim Model

The main problem with the Seeheim Model is that control of the dialogue is handled by two components, with the dialogue control component describing the overall dialogue, while the presentation component (which is device-dependent and hard-coded), is responsible for the interaction techniques (such as menus, icons, windows, etc.). Thus, the Seeheim Model allows the general dialogue of the interface to be specified, but not the dialogue of the actual interaction techniques. However, in most modern event-based interfaces, it is these interaction techniques that dominate the user's interaction with the interface and should therefore be at the heart of any dialogue model of an adaptive interface. Furthermore, the role of the presentation and application model components should be minimised, as these parts of the system are device dependent. A modified version of the Seeheim Model is thus required, if it is to form the basis of an architecture for an adaptive interface.

AN ARCHITECTURE FOR ADAPTIVE INTERFACES

To help explain the architecture of the adaptive interface, we will introduce a simple `Note Pad' application. The relevant features of the application are as follows:

1) *Entering Text.* The user can enter lines of text, which are stored by the application.
2) *File Handling.* The text can be saved to, or read from, the local file system.

One possible interface for such an application is to allow the user to enter the text in a window, and let the file handling capabilities be accessed through a menu, with three options being available, namely: Open, Save, and Save As. While describing the architecture, we will concentrate on explaining how the menu of this application can be specified.

The architecture for the adaptive interface is shown in Figure 3, and consists of three distinct levels:

1) *Interaction Level.* Concerned with the interaction between the application and the output facilities of the current operating system.
2) *Adaption Level.* Contains the mechanisms which allow the interface to be adapted.
3) *Storage Level.* Holds all of the data required by the adaptive interface.

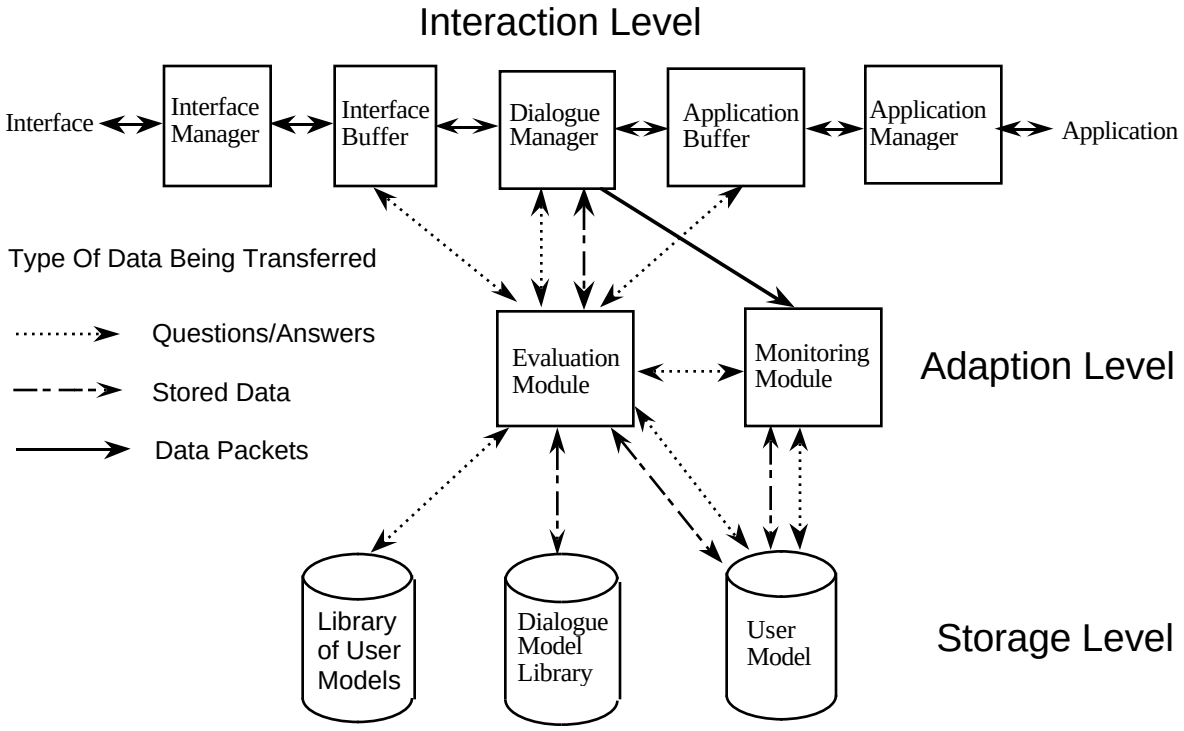

Figure 3 - The Architecture of the Adaptive Interface

The architecture will now be described in detail.

Interaction Level

From Figure 3, it can be seen that the Interaction Level is made up of five separate modules, with the Dialogue Manager being at the heart of the interaction.

Dialogue Manager

The role of the Dialogue Manager is to control the interaction between the user and the application, in an application and operating system independent way. This is achieved by using a variant of state transition diagrams, called a Memory-Based State Transition Diagram, to control the dialogue, with pseudo-code that can be interpreted by the Interface Buffer being used to control the appearance of the interface.

When specifying a dialogue with state transition diagram techniques, the complete dialogue is usually specified in a single diagram (possibly breaking the diagram up using sub diagrams). However, this means that multi-threaded dialogues cannot be specified, and these form the basis of most modern interfaces. Therefore, in the dialogue manager, each individual interaction technique, such as menus, icons, text tools, drawing tools, and so on, are specified independently as separate dialogues, in the form of a Memory-Based State Transition Diagram.

Then, at any one time, any number of dialogues can be active simultaneously, which means that any interface can be specified [8].

For example, in the Note Pad application, there are two interaction techniques, namely the text window and menu. Considering the menu, we will describe a menu that behaves equivalently to a Macintosh menu, i.e. press mouse button down when cursor over menu name, move cursor to desired menu option, release mouse button. This is given in Figure 4.

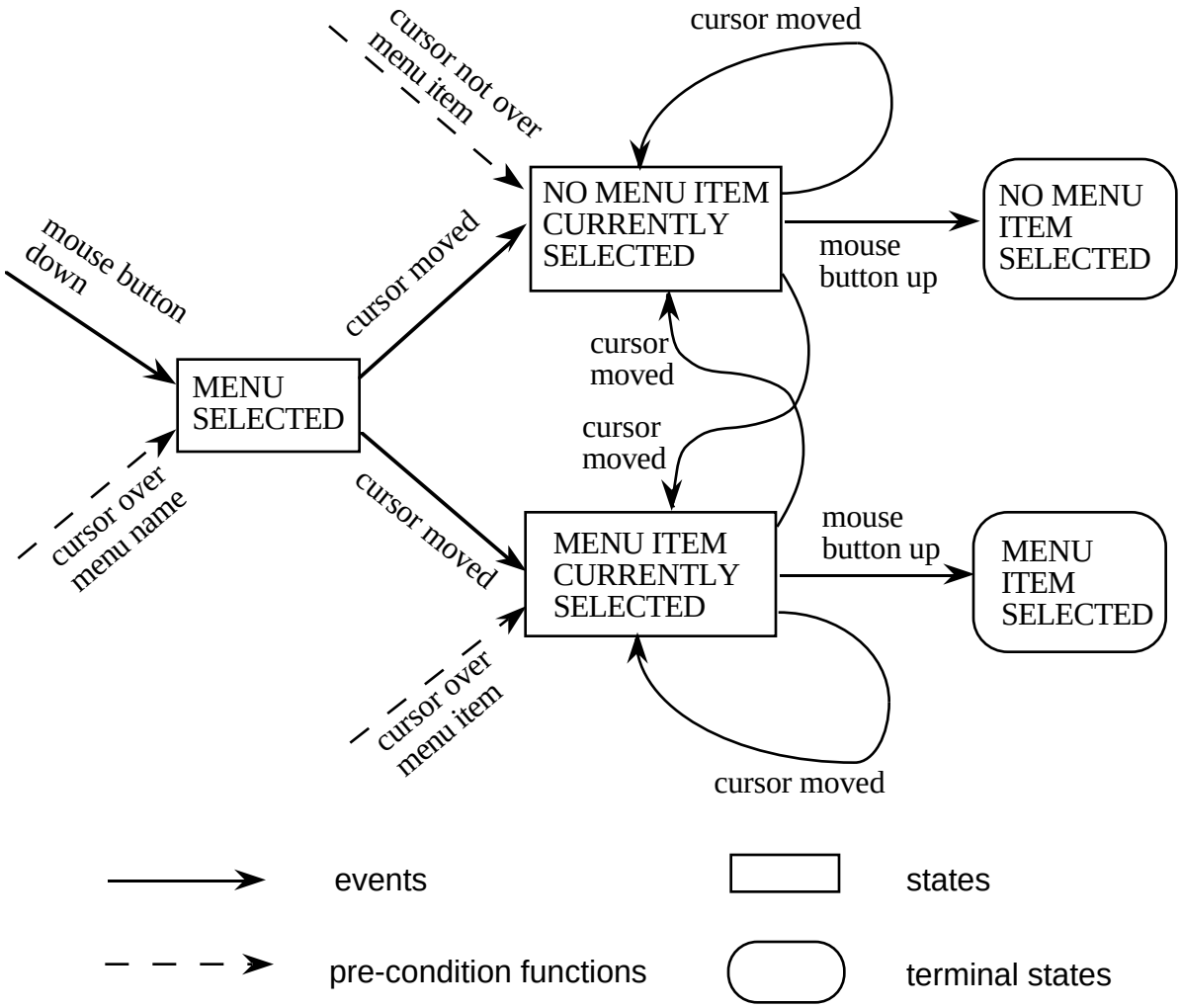

Figure 4 - Dialogue for Selecting a Menu Item Using the `Macintosh Technique' (Method `A' of Figure 1)

At each stage of the dialogue, the Dialogue Manager gives instructions to the Interface and Application Buffers through "packets". For example, the following packet would be sent if the state `menu item selected' was reached, and the user had selected the `Save' menu item :

alert `Interface Buffer' `menu item selected' `Save'

Application and Interface Buffers

The Interface and Application Buffers perform two distinct functions, namely:

1) *The Buffering of Packets*. The Interface Manager, Dialogue Manager and Application will generate packets that are not required by any parts of the

system, and therefore need to be discarded. For example, if the state `NO MENU ITEM SELECTED' was reached (Figure 4), the Dialogue Manager tells the Application Buffer that no menu item has been selected. However this information is not needed by the Note Pad application and so the packet should be discarded.

2) *Providing Information.* The Evaluation Module (to be described later), uses the Interface and Application Buffers to query the interface (to discover the capabilities of the operating system's graphical interface), and the application (to discover the application's functions, type, etc.). For example, in the Note Pad application, the evaluation module needs to know what editing and file handling facilities are supported. More specifically, for the `Save' command, the Evaluation Module needs to know that a save function is available, which, if called with no arguments, will save the current Note Pad, to the current file system, using the default file name.

Interface Manager

The objective of the Interface Manager is to provide a device independent interface to the operating systems graphical capabilities, converting packets from the Interface Buffer into commands for the operating system and vice-versa. Unlike the Presentation Component in the Seeheim Architecture, discussed earlier, it has no managerial responsibilities, but is rather used only to transfer information between the Interface Buffer and the Operating System.

The Application Manager

The Application Manager is very similar to the Interface Manager, except that it is concerned with the application rather than the interface, and is thus the sole component of the architecture that has to be changed for each application. The Application Manager contains a complete description of the available functionality of the application, with sufficient information to enable data packets from the Dialogue Manager to be converted into the required commands.

For example, in the case of the Note Pad application, the Application Manager needs to know the name of the function which saves the current Note Pad, and the facts that the function requires no parameters and does not return anything to be displayed. The Application Manager will also know how to obtain the help files from the application, concerned with the functionality of the command.

Adaption Level

The Adaption Level is concerned with adapting the interface itself. To enable this to be completed, there are two modules which analyse and record the users' interaction with the system, and then use all available information to adapt the interface to the user's requirements.

Monitoring Module

The role of the Monitoring Module is to record the interaction that the user has with the system, so that the Evaluation Module can analyse this interaction. The user's interaction is not stored as a list of low-level events (such as key presses) but in terms of the paths traversed through the dialogue models representing the interaction.

For example, in the Note Pad application, the states that a first time user entered (referring back to Figure 4) when attempting to select the menu item to save the current Note Pad could have been as shown in Figure 5. This clearly shows that the user has not grasped the fact that the cursor must be over the desired menu item when the mouse button is released.

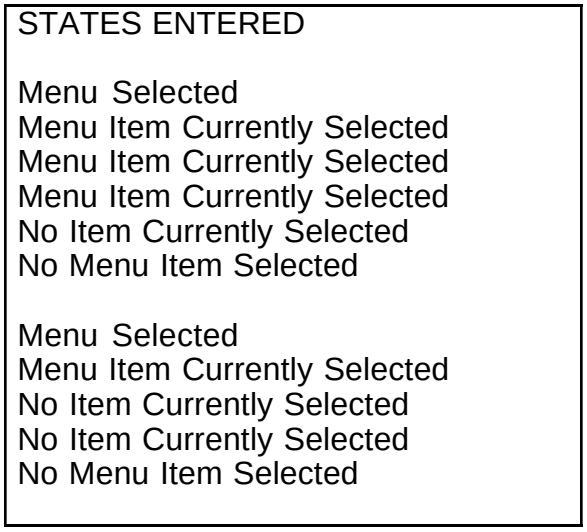

Figure 5 - States Entered when Novice User Attempts to Select an Item from a Menu

The Evaluation Module

The Evaluation Module is at the heart of the adaptive interface, and has a number of distinct roles depending on the current state of the system.

1) *Set Up.* The first time a user uses a new application, an interface to that application has to be created. Assuming that this is not the first adaptive interface that the user has used, the Evaluation Module, by querying the Application and Interface Buffers, will construct the required dialogue models for the system, based on the User Model and the bank of dialogue models that are available.

2) *Running Mode.* Whilst the user is using the system, the Evaluation module runs in the background analysing the users interactions with the system (which are recorded by the Monitoring Module). In order to do this, the Evaluation Module communicates with most parts of the architecture. Then, if the evaluation module believes that the user would benefit from the interface being changed in some way, it decides which adaptions are required. Then the adaption itself can begin.

3) *Adapting Mode.* During the adaption process, the Evaluation Module adopts a coordinating role, ensuring that the user is kept informed about the adaptions taking place, and changing the dialogue models which actually perform the adaption.

In order to perform its function, the Evaluation Module needs intelligence, as it has to decide on an initial interface for the user, decide which parts of the interface are unsuitable and thus the most appropriate time to adapt the interface. As a basis for making these decisions, the Evaluation Module has available to it:

1) Knowledge on how the user is performing with each part of the current interface (from the User Model).
2) The user's general level of experience with the current type of application (from the User Model).

3) Other users' experiences with similar interfaces, who have similar computing experience, knowledge, and so on (from the Library of User Models).

Assuming that there are a wide variety of users with different characteristics, the Evaluation Module can match the current user characteristics with the available libraries of dialogue models to select the optimum interface for the user.

For example, suppose a user tried to select a menu item in the Note Pad application using Method B from Figure 1, due to the fact that they had previously used an MS Windows application, which the system was not aware of (otherwise they would have been given the correct menu style to begin with). The interface could then be adapted, to give the user the correct menu style, simply by replacing the menu dialogues appropriately. If required, a more fundamental adaption of the interface, say to a completely command line based Note Pad, could be achieved, by replacing all of the current dialogue models with dialogue models appropriate to a command line based interface.

Storage Level

The storage level holds all of the data required by the adaptive interface. This data is of two distinct types, namely that concerned with the user and that concerned with the dialogue models that make up the interface.

Dialogue Model Library

The Dialogue Model Library is simply made up of the the contents of the Dialogue Manager, for every user and for every system that each user uses. However, this is not allowed to grow excessively large, since in reality there is only a comparably small set of possible dialogues. For example it would become difficult to find ten different ways to select an item from a menu using a mouse.

User Model

The problems of building a model of the user have been the subject of much research, particularly in the field of Artificial Intelligence in Education [14]. However, the problems concerned with building a user model for adaptive interfaces are less severe, especially if our architecture is used, as a user's ability and experience can be represented by the dialogue models that they have previously used and are currently using, together with additional information on the efficiency of their interaction with the interface.

Library Of User Models

If a user agrees that their model could be made available anonymously to the system, this would be extremely useful, as users could be compared for similar characteristics which may predict their preferred interface.

CONCLUDING REMARKS

In this paper we have proposed an architecture for an adaptive interface, where the application has no control over the interface that the user will use. The main advantage of such an architecture is that the user would be offered a consistent interface across all of the applications which they use, which should improve their productivity with the application [12].

However, before the goal of adaptive interfaces can be achieved, a number of obstacles, both technical and psychological, have to be overcome. Psychologically, users could be concerned that the machine is taking over, or constantly monitoring them. This might be overcome by allowing users to view their user model, and allowing them to edit it if they believe that it is

wrong, but this means that the model must be understandable to the user and therefore relatively simple.

Technically, there are a number of challenges, which overlap into all areas of computer science research. For example, the user will have to be provided with help during the interaction. The standard problem with intelligent help systems is to tailor the help to the expertise of the user. For example, the Unix Consultant [4] tailors its advice on Unix depending on the expertise of the user. However, in a completely adaptive system, the help will have to be derived both from the Dialogue Manager (for `how to do it' knowledge), and from the application (to discover what a command actually does). Only then can help be tailored to the needs of the user and output in an appropriate form.

These are real challenges, but if a fully adaptive system could become reality, with software tailored to the actual user rather than the `average' user, then the usefulness of computers could be dramatically improved.

ACKNOWLEDGEMENTS

This research is supported by the Engineering and Physical Science Research Council (EPSRC) and by the Computing Department, Lancaster University.

REFERENCES

1. Benyon, D. Adaptive Systems: A Solution to Usability Problems, User Modeling and User-Adapted Interaction, 3 (1), 1993, pp. 65-87.

2. Benyon, D. and Murray, D. Experience With Adaptive Interfaces, The Computer Journal, 31 (5), 1988, pp. 465-473.

3. Berthome-Montoy, A. Building a User Model for Self-Adaptive Menu-Based Interfaces, in Proceedings Fourth International Conference on User Modeling (Hyannis, Massachusetts, 1994), The MITRE Corporation, Bedford, Massachusetts, pp. 51-57

4. Chin, D. N. KNOME: Modeling What the User Knows in UC. In Kobsa, A. and Wahlster, W. (ed.), User Models in Dialog Systems, Springer-Verlag, Berlin, 1989

5. Cote-Muñoz, J. A. AIDA - An Adaptive System For Interactive Drafting. In Schneider-Hufschmidt, M., Kühnm, T. and Malinowski, U. (ed.), Adaptive User Interfaces: principles and practice, Elsevier Science Publisher B.V. (North-Holand), Amsterdam, 1993

6. Egan, D. E. Individual Differences in Human-Computer Interaction. In Helander, M. (ed.), Handbook of Human-Computer Interaction, Elsevier Science Publisher B.V. (North-Holand), Amsterdam, 1988

7. Green, M. The University of Alberta User Interface Management System, Computer Graphics, 19 (3), 1985, pp. 205-213.

8. Green, M. A Survey of Three Dialogue Models, ACM Transactions on Graphics, 5 (3), 1986, pp. 244-275.

9. Innocent, P. R. Towards self-adaptive interface systems, International Journal of Man-Machine Studies, 16 (3), 1982, pp. 287-299.

10. Jennings, F., Benyon, D. and Murray, D. Adapting systems to differences between individuals, Acta Psychologica, 78 (1-3), 1991, pp. 243-256.

11. McTear, M. F. User modelling for adaptive computer systems: a survey of recent developments, Artificial Intelligence Review, 7 (3-4), 1993, pp. 157-184.

12. Payne, S. J. and Green, T. R. G. Task-Action Grammars: A Model of the Mental Representation of Task Languages, Human-Computer Interaction, 2 1986, pp. 93-133.

13. Sein, M. K., Olfman, L., Bostrom, R. P. and Davis, S. A. Visualization ability as a predictor of user learning success, International Journal of Man-Machine Studies, 39 (4), 1993, pp. 599-620.

14. Self, J. Bypassing the Intractable Problem of Student Modeling. In Frasson, C. and Gauthier, G. (ed.), Intelligent Tutoring Systems: At the Crossroad of Artificial Intelligence and Education, 1990

15. Vicente, K. J. and Williges, R. C. Accommodating Individual differences in searching a hierarchical file system, International Journal of Man-Machine Studies, 29 1988, pp. 647-668.