ACCESSING NETWORKED SERVICES: A USER INTERFACE DESIGN PROBLEM

ABDUL HANAN ABDULLAH

Doctor of Philosophy

The University of Aston in Birmingham

October 1994

The University of Aston in Birmingham

ACCESSING NETWORKED SERVICES: A USER INTERFACE DESIGN PROBLEM

ABDUL HANAN ABDULLAH

Doctor of Philosophy

1994

Summary

This research investigates the general user interface problems in using networked services. Some of the problems are: users have to recall machine names and procedures to invoke networked services; interactions with some of the services are by means of menu-based interfaces which are quite cumbersome to use; inconsistencies exist between the interfaces for different services because they were developed independently. These problems have to be removed so that users can use the services effectively.

A prototype system has been developed to help users interact with networked services. This consists of software which gives the user an easy and consistent interface with the various services. The prototype is based on a graphical user interface and it includes the following applications: Bath Information & Data Services; electronic mail; file editor. The prototype incorporates an online help facility to assist users using the system.

The prototype can be divided into two parts: the user interface part that manages interaction with the user; the communication part that enables the communication with networked services to take place. The implementation is carried out using an object-oriented approach where both the user interface part and communication part are objects. The essential characteristics of object-orientation - abstraction, encapsulation, inheritance and polymorphism - can all contribute to the better design and implementation of the prototype. The Smalltalk Model-View-Controller (MVC) methodology has been the framework for the construction of the prototype user interface.

The purpose of the development was to study the effectiveness of users interaction to networked services. Having completed the prototype, tests users were requested to use the system to evaluate its effectiveness. The evaluation of the prototype is based on observation, i.e. observing the way users use the system and the opinion rating given by the users. Recommendations to improve further the prototype are given based on the results of the evaluation.

Keywords: user interface design, object-oriented programming, usability, networked services, electronic mail

PUBLICATION

A paper based on this research has been presented at a conference. The details of the paper and conference are:

Title	: Implementing an Interface to Networked Services.
Authors	: Abdul Hanan Abdullah & Brian Gay.
Conference Name	: ACM 12th SIGDOC Conference.
Conference Place	: Banff, Canada.
Conference Date	: 2 - 5 th October 1994.

A copy of the conference paper is placed in appendix D

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Brian Gay for his invaluable advice and assistance throughout the course of this research.

My thanks also go to Mary Finean, Arif Jawaid and Baldev Singh Pawar for their time reading, proposing modifications and making comments to improve the presentation of this thesis.

I also wish to acknowledge the University of Technology of Malaysia and the Public Services Department of Malaysia for providing financial support of this research.

My special gratitude to both my parents, my wife Rohana and my children Umar, Hafidz, Sakinah, and Mohammad for their patience and understanding during the period of this research and writing.

TABLE OF CONTENTS

APPENDICES

FIGURES

TABLES

CHAPTER ONE

INTRODUCTION

1.0 BACKGROUND

The last ten years have seen a tremendous growth in the field of computer networking. As a result, users who are familiar with the network may easily access various networked services from their own departments. Basically, the networked services are the applications that use the network facility and can be accessed through the computer network. These include the services provided by the different departments in the organisation such as the online library search, electronic mail facilities and other services provided by other external organisations such as the ISI Bath Information and Data Services (BIDS).

Accessing networked services can be quite complicated even for computer experts. Users need to know where the services are located and then different procedures have to be employed to access different network services. It is often the case that many users are discouraged from using the services and for some the mention of it is a frightening experience.

Network-based services are used by university students, researchers and professionals from various disciplines. The majority of users have some kind of experience with popular software tools such as word processor, spreadsheet and graphical packages which are widely available on personal computers. For them, the computer is only a tool to enhance their productivity and they expect the same kind of convenience and flexibility from the networked services as they experience when using their software tools (Colomb, 1991).

An aspect of networked services that requires research attention is the user interface. Sadowsky (1993) points out that the current state of networked services is relatively primitive and tools to guide users to find items of significant interest are still inadequate. A well designed user interface is required if an effective use of the service is to be achieved. The interface not only enables users to access various networked services, but also enables them to navigate the services.

This chapter discusses the networked services which are available at the Aston University. A brief introduction to some of the services is given. General problems which are related to the networked services are also identified.

1.1 NETWORKED SERVICES AT ASTON UNIVERSITY

Networked services are essential for any academic environment. The services (Rosner, 1991) would help the creation and distribution of information, which are the essence of teaching and research processes. Increasingly in the future, the ability to manage a diversity of information-based services is required by any academic institution in order to excel in research and teaching.

A campus wide network has been developed at Aston University in the late 1980s (Brindley, 1990; Krivine, 1989). The four million pounds project was funded by both the Department of Trade and Industry and the University Grants Committee. As well as expensive infrastructure, close attention has also been given to the provision of networked services. Examples of networked services which have been installed are as follows:

1. Electronic Mail Facilities.
2. Library Search and Information Services.
3. Bath Information Data Services.
4. USENET News.
5. Directory Services.

6. File Transfer Facilities.

7. National Information Services and Systems.

The above services will be discussed in greater details in the sections that follow.

1.1.1 ELECTRONIC MAIL

Electronic mail is the most successful network application service. A market-research firm (Reinhardt, 1993) reveals that the number of electronic mail users in the U.S. rose 60 percent in the year 1992 and will rise another 60 percent the following year. By the year 1995, the number of users in the country could reach 38 million.

Plattner and Lubich (1988) list a number of advantages of electronic mail that contributes to its popularity. Firstly, electronic mail does not require the recipients to be present at the time the message arrives. This is especially practical when the communicating parties are on different continents, where the working hours are not the same. Secondly, electronic mail systems support bilateral and multilateral modes of communications. Users may send a message to multiple recipients at a time. Thirdly, electronic mail is usually an integral component of office automation systems. Thus a mail message can be printed, copied into another document, filed away or forwarded.

Figure 1.1 shows a general model of an electronic mail system. It consists of two main components, i.e. message transfer agent and user agent. The user agent is a program that provides the interface to the mail system. Basically, it allows users to compose, send and receive mail. An advanced user agent may include facilities such as searching the folders for messages and management of personal mailing lists.

The message transfer agent is concerned with the delivery of the mail message from the originator to the recipient. It is analogous to the electronic post office. With the postal system a letter may visit a few post offices before being delivered. Similarly a mail message may have to pass through several message transfer agents to reach the destination.

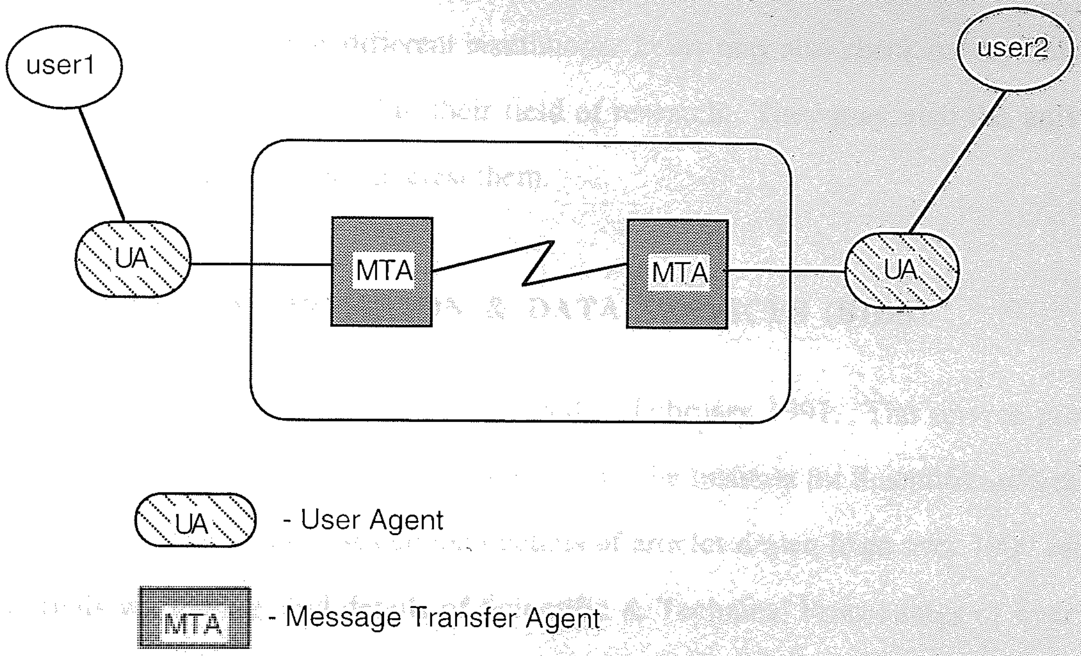


Figure 1.1 : General Model of Electronic Mail

1.1.2 LIBRARY SEARCH AND INFORMATION SERVICES

The Library is an important service in any academic institution. Library users may interrogate the library catalogue, search for books or journals by title, ISBN number or author and reserve them if they wish. Users also may wish to find out the number of books borrowed from the library, the respective date due for each book and check whether there are any fines outstanding. The GEAC online computer search was introduced in 1983 to help users find such information (GEAC, 1991).

The Library and Information Services are the first service to be placed on the campus network. With the service on the network, users may request many of the services from their desk, such as accessing the catalogue, placing books on reserve and renewing loans. They are no longer required to visit the library building every time they wish to access such services.

The campus network is connected to most Higher Education Institutions in the United Kingdom via JANET (Joint Academic Network). Thus, users can call directly to the library catalogues of other institutions (Library & Information Services, 1993). Calling other library catalogues may be required by users to look for thesis and research publications produced by different institutions. Users may also check nearby university libraries for journals related to their field of research. They may visit the university which holds journals which interest them.

1.1.3 BATH INFORMATION & DATA SERVICES (BIDS)

The BIDS ISI Data Service was launched in February 1991. The service provides access to four databases supplied and owned by the Institute for Scientific Information, USA (ISI). These databases contain details of articles drawn from over 7000 selected journals worldwide, and details of Scientific & Technical Proceedings of over 4000 conferences per year (Morrow, 1992).

The GEAC online service only allows users to search for books and journals managed by the library. It does not contain details of the articles in a journal. BIDS offers a complementary service to the GEAC online service. Useful facilities provided by BIDS include a service for users to retrieve articles in all scientific journals by title, keyword or authors or optionally retrieve all articles found in any selected journal.

There are other commercial services that offer access to the same data supplied by the ISI incorporation, where payment to the service is according to the usage of the service. These tend to be used by librarians who act as search facilitators. Users describe the information they wish to retrieve and the librarian conducts a search on their behalf. It is assumed that the librarian is prepared to spend time gaining the experience to conduct a quick and effective search.

BIDS on the other hand was designed from the outset as an end-user service (Morrow, 1992). BIDS offers a direct service to users' desks, offices and research laboratories. To use BIDS, users must be a member of a subscribing institution. Since, the payment is a flat-rate annual fee, connect time is less important, but a simple and easily understood interface is crucial to its success.

1.1.4 USENET NEWS

The USENET news (Erickson,1993; Tanenbaum,1989) holds hundreds of newsgroups on a variety of topics. There are groups for most areas of computer science, such as programming languages, databases, graphics and operating systems, as well as for many sciences, hobbies, political and social interests. The subscribers to any newsgroup may read any of the messages posted to that group and they may also post messages of their own.

The newsgroups are divided into a few broad categories according to their subjects. Examples of these categories include:

1. **Comp:** Computers and their software. Currently there are more than 500 topics under this category. Examples are comp.sys.apollo, comp.ibm.pc.software, comp.ai.genetic and comp.windows.x.
2. **Alt:** Alternative groups that do not fit into any other category. This group is further classified under various topics such as sport, music, religion, politics and rock-n-roll.
3. **News:** The news system itself and its operation. This includes a discussion on policy issues and technical aspects of USENET as well as a question and answer session regarding USENET.

There are thousands of groups and each group maintain a list of articles, summarised by subject and author. Users need news reader software to participate in USENET. A news reader offers users a variety of options such as browsing through lists of newsgroups and articles, searching articles by title or author, or posting a message to any group.

1.1.5 DIRECTORY SERVICES

In telecommunication networks, telephone books and directory services are provided that contain a list of subscribers' names and their respective numbers. In the future, these services will be replaced by online directory service (Tanenbaum, 1989). The service allows users to look for detailed information on a named person or institution. The information may include address, telephone number and electronic mail address.

The implementation of an international directory service is not that easy. Different countries have different ways of implementing their directory systems. As an example, not all countries sort their directories by the last names. Even among the countries that sort their directories by the last name, what constitute a last name may differ from one country to another. Furthermore, in the U.S, all subscribers with the same last name are sorted using their first names as the secondary key, whereas in Holland, the secondary key is the street address and in some Scandinavian Countries, the secondary key is supplied by subscribers. The key could be an occupation, a first name or even an academic degree.

An international standard directory service has been introduced to resolve some of the differences in naming and addressing convention. The service allows users to look up names based on attributes. For example, users may request details of a person named 'William' working in 'ParcPlace Sales Office in California'.

1.1.6 FILE TRANSFER

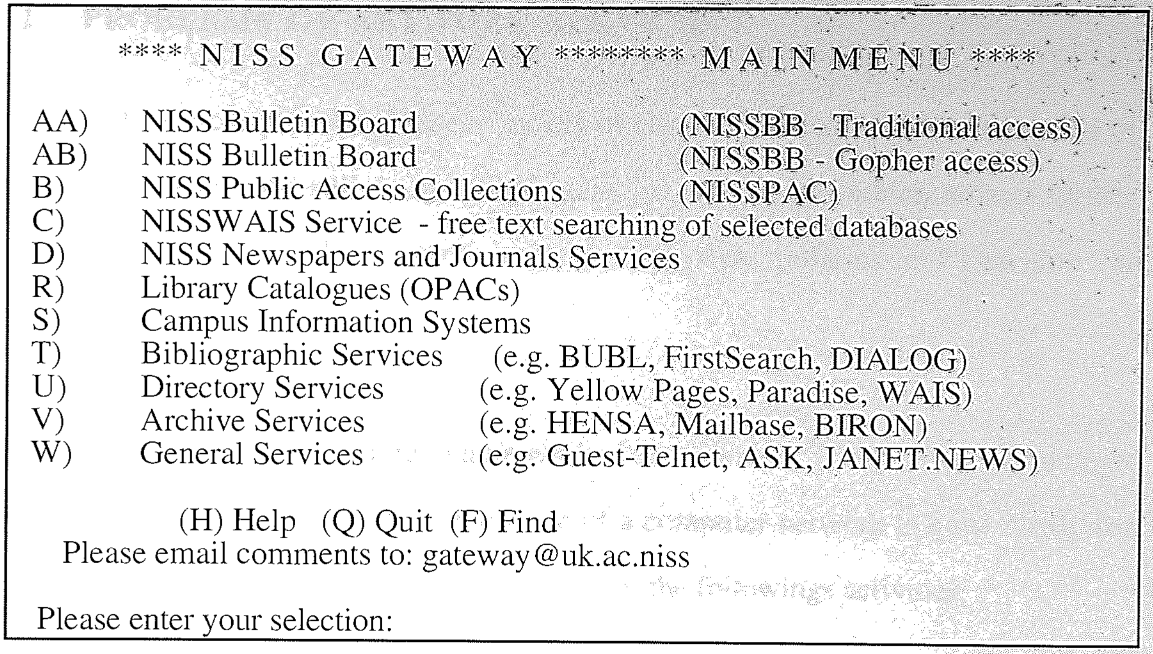
The term file transfer (Comer, 1988) means copying a file from one machine to another. This service allows users to log on to remote machine, list remote directories, initiate the transfer of files to or from the remote machine, and may execute a few simple commands remotely. Examples of such commands include delete files and create folders. The service prevents users from accessing other portions of the computer's file system.

Most archive sites contain text files that provide additional descriptive information about the contents of the site, a particular directory or files. These information files are often named 'readme' and 'index' (Hahn & Stout, 1994). The value of these files depends on the completeness, clarity and currency of the information provided.

There are many reasons why users transfer files among a variety of machines. Researchers may retrieve remote programs from any archive site and execute them locally for testing purposes. Some users may download files so that they can be printed using a departmental laser printer. Others may upload files so that they may run a program on a faster processor.

1.1.7 NATIONAL INFORMATION SERVICES AND SYSTEMS (NISS)

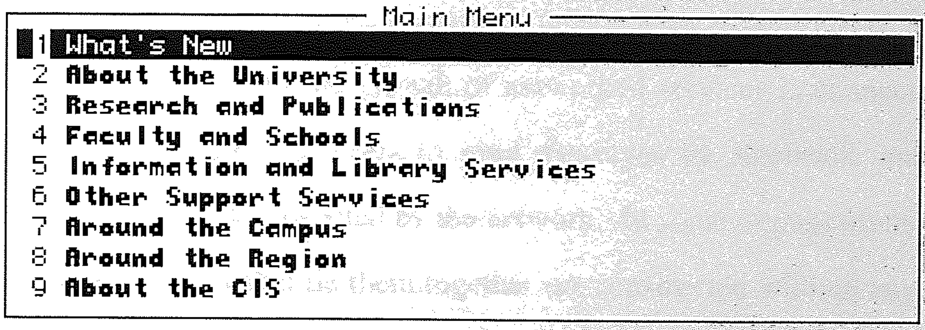
National Information Services and Systems (NISS) was established in 1984. NISS provides computer disseminated information for the UK academic community. NISS services are freely accessible for JANET users. The NISS gateway provides menu-driven access to a large number of online information services world-wide. By using the services users are able to roam through the global network unrestricted by local or national barriers. The NISS main menu is shown in figure 1.2.



```
**** NISS GATEWAY ******** MAIN MENU ****

AA)   NISS Bulletin Board                (NISSBB - Traditional access)
AB)   NISS Bulletin Board                (NISSBB - Gopher access)
B)    NISS Public Access Collections     (NISSPAC)
C)    NISSWAIS Service - free text searching of selected databases
D)    NISS Newspapers and Journals Services
R)    Library Catalogues (OPACs)
S)    Campus Information Systems
T)    Bibliographic Services   (e.g. BUBL, FirstSearch, DIALOG)
U)    Directory Services       (e.g. Yellow Pages, Paradise, WAIS)
V)    Archive Services         (e.g. HENSA, Mailbase, BIRON)
W)    General Services         (e.g. Guest-Telnet, ASK, JANET.NEWS)

        (H) Help  (Q) Quit  (F) Find
  Please email comments to: gateway@uk.ac.niss

Please enter your selection:
```

Figure 1.2 : NISS Main Menu

The NISS Campus Information System allows users to access information about different universities in the United Kingdom and United States which are connected to the service. Figure 1.3 shows the main menu of the Birmingham University Campus Information Services. Users may access various campus activities and services by using the service.



```
Main Menu
1 What's New
2 About the University
3 Research and Publications
4 Faculty and Schools
5 Information and Library Services
6 Other Support Services
7 Around the Campus
8 Around the Region
9 About the CIS
```

Figure 1.3 : Birmingham University Campus Information Service

Some NISS services such as NISS Newspapers and Journals Services are only limited for demonstration. The Newspapers Services allow users to search for newspapers articles by specifying selected topics. The Journals Services contain reports on feasibility studies of electronic journals. When the services are available, users may access research journals through the network.

1.2 PROBLEMS OF NETWORK SERVICES

Networked services provide effective means of communication and dissemination of information. There are still many issues related to the services which require further research. Problems such as access controls, copyright policies and charging for services are still unresolved.

Networked services only operate in a networked environment. Despite the advantages of networking, installation and maintenance of a computer network is quite costly for some organisations. Costs have to be allocated for the followings activities:

- Wiring of a complete building or several buildings.
- Special networking equipment and dedicated computers to manage the communications within the organisation and with the outside world.
- Personnel to look after the development and maintenance of the network.
- Integrating existing equipment from different vendors.
- Subscribing to the communication line and some of the networked services such as Bath Information Data Services (BIDS).

Another factor that impedes the growth of networked services is connectivity. For example, it is impossible for users to send messages by electronic mail to their colleagues if they are not connected by the network. In some organisations, computer systems and the network that tie them together are constructed without any long term planning. Each organisation purchases equipment and software that satisfy their local computing needs. As a result, different organisations have their own set of services and different ways of sending and receiving services. Users may communicate with each other within the same organisation, but it is impossible to communicate with staff members from other organisations.

Users find it convenient to use other popular services such as the fax machine because greater number of users are connected to the services. They can easily send messages to their colleagues in almost all parts of the world by simply dialling the fax number. Any user with a telephone line can have his own fax machine at home or office.

The computer network is evolving from a purely research prototype (Comer, 1988). It has now developed into a mesh of interconnected networks which connect researchers, industry and government agencies across the world. The exponential growth of the network creates many problems. Dixon (1993) points out the following problems:

- Current address space is insufficient.
- Routing equipment can no longer cope with the growing number of networks.
- There is no automatic means of updating all the routers in the world when a new network is connected.
- The rate of development of new applications is slow.

Comer (1988) also states other problems with the computer network. There is no automatic mechanism to carry out the following services:

- Storing and locating information about individual users.
- Storing and locating information services offered by different hosts in the network.

Dixon (1993) points out that "increasingly, network users are outside the computer science community and they are not interested in the technology - they just want something that they can plug-in and turn on."

The ease of interaction becomes increasingly important as networked services penetrate markets of novices and non-technical users. Many existing networked services do not provide services which are easy to learn and convenient to use. Users have to recall machine names and commands in order to invoke some of the services. Some users are not aware that networked services are available and others are put off by difficulties in using the services.

1.3 STRUCTURE OF THE THESIS

This thesis consists of eight chapters. Chapter 2 reviews the literature on the user interface which is related to this research. The chapter mentions the different types of user interfaces in brief. A detailed discussion of issues related to the graphical user interface which include general design principles is given. The chapter also overviews the roles of online help facilities and discusses the procedures to evaluate user interfaces.

Chapter 3 reviews the user interfaces of some of the networked services. It starts by describing the general problems and difficulties in using services. Discussion on some of the networked services at the Aston University, i.e. the library services, Usenet news, Bath Information Data Services and electronic mail are also given. The problems in using these services are highlighted.

Chapter 4 discusses general design issues for the user interfaces and networked services. The issues include the ways users interact with the system and how commands and information are presented to users. The design is based on a graphical user interface and the user interface design principles that were reviewed in chapter 2 are taken as guidelines.

Chapter 5 discusses the design of the prototype. The general design issues have already been discussed in chapter 4. This chapter discusses a refinement to the design which is specific to the application being developed. The prototype consists of user interfaces to two networked services, i.e. electronic mail and Bath Information Data Services. The prototype also includes a file editor which functions as a support service to the networked services.

Chapter 6 discusses the implementation issues of the prototype. Discussion include the advantages of implementing the prototype using object-oriented approach. The reasons for adopting the Smalltalk language to implement the prototype are also discussed. The Smalltalk MVC framework and its support mechanisms are described in detail. Finally, the approaches taken in implementing the prototype are described.

Chapter 7 discusses the result of the evaluation. This includes comments given by test users and observations made by the facilitator when users execute the given test tasks. Recommendations to improve further the prototype are given. The chapter also discusses the rating given by the users.

Chapter 8 concludes the thesis by summarising the author's work. Suggestions are also given for future development.

CHAPTER TWO

LITERATURE REVIEW

2.0 INTRODUCTION

The literature shows a widespread growth of research in many aspects of the user interface in recent years. It is beyond the scope of this thesis to give a complete review of research and findings which are related to the whole of the field. Only those aspects which have laid the groundwork for the development of this research are presented in this chapter.

This chapter discusses the general aspects of user interface design. It starts with the definition and a brief overview of the history of user interfaces. It is followed then by various discussions which are related to the graphical user interface. This includes discussion on direct manipulation, tools involved in the design of user interface, general design principles and online help facilities.

2.1 DEFINITION OF USER INTERFACE

Apple (1992) defines the user interface as "the rules and conventions by which a computer system communicates with the person operating it."

The term 'user interface' will be used in this thesis to describe a computer's interface to the user. According to Grudin (1993), the term assumes that the computer is the point of reference. The same assumption applies when different categories of users are discussed in the literature. For example, novice or naive users are often expert in their field, but as far as the use of computer is concerned, they are beginners.

Dialogue is another term which is used extensively in the literature that discusses the user interface. The terms interface and dialogue carry a different meaning. The dialogue is defined as the observable exchange of symbols and actions between human and computer, whereas an interface is the supporting software and hardware through which the dialogue takes place (Hartson & Hix,1989; Chen,1990).

2.2 REVIEW OF USER INTERFACES

In the early days of computing, physical switches on the front panel of a computer were manipulated to load program instructions into the machine. Then, the operator would press a 'run' switch to execute the program. This was a primitive user interface and it was quite cumbersome to use.

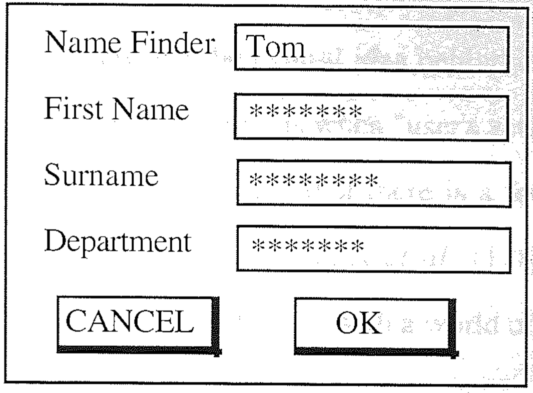
In the 1970's, the TTY-type terminal was introduced along with the emergence of time-sharing systems. The terminal allowed the development of prompt and command language style interfaces (Morse & Reynolds, 1993). In the prompt style interface, the application asks questions that must be answered by users. This style of interface is suitable for an application that requires extensive branching, i.e. every question has a large number of replies and the question asked is dependent on the previous reply (Coats & Vlaeminke, 1987). Thus, this type of interface is widely used in the expert system application. Among the limitations of prompt style interface are the interaction is controlled by the application and answering a series of questions one at a time is quite tedious for the frequent users.

The command language interface is quite popular, primarily because it has been a common style for computer operating systems and most of the programming languages. It was gradually developed with the development of computer hardwares and the diversity of computer applications. Early programming languages such as FORTRAN, COBOL and ALGOL were designed for a non-interactive computer environment. Programmers would write hundreds or thousands lines of codes and then compile and run to get the result. This was followed by incremental programming where programmers would create a smaller pieces of code and test them interactively. In the late 1980's, scripting languages such as HyperCard were introduced to allow users to manipulate the screen presentation and control the mouse (Kaehler, 1988).

It is beyond the scope of this thesis to discuss the diversity of command languages. The style of the interface is very powerful and flexible, but due to its complexity, this style of interface is confined to expert users only. Large amounts of time and efforts are required to understand the system. Learning the full capabilities of the system is normally acquired through external instruction, such as training and manuals rather than by using the system itself (Coats & Vlaeminke, 1987). Once understood, users have to recall the most appropriate commands and their respective syntax. Error messages and online help are hard to provide because there is no way for the system to know what users wish to do. Furthermore, since this type of interface may require extensive keyboard entry, it could be tiresome and error-prone.

Menu-based interfaces were introduced to eliminate many of the limitations found in the command-based interface. This interface requires CRT technology, which allows full screen display capability. The list of the items is displayed on the screen and a selection can be made by a mouse click or few keystrokes. This style of interface is attractive since it requires little learning and eliminates the need to memorise complex commands. Menu-based interfaces can be easily supported by online help and error handling facilities. Different types of menus, such as pull down and pop-up menu will be discussed in the topic of graphical user interface. Many aspects of menu design have been discussed in detail in the literature. A comprehensive discussion on this issue is given by Paap (1988) and Shneiderman (1992).

Menu-based selection is only appropriate when users need to select from a list of commands or items. In the case where data entry is required, the form fill-in style of user interface is recommended (Shneiderman, 1992). Fig 2.1 is an example of form fill-in. This style of user interface is appealing because it resembles the familiar paper form. Most of the information is visible, thus giving users a feeling of being in control of the dialogue. Users have to fill in the blank and move the cursor from one field to another by certain key, such as TAB. Form fill-in is appropriate when the pattern of input is predictable. Multiple choice menu are a type of form fill-in.



Name Finder	Tom
First Name	*******
Surname	********
Department	*******

Figure 2.1 : Form Fill-in

Direct manipulation user interfaces are becoming popular due to the availability of bit-mapped screen displays, powerful processors and input devices such as the mouse (Verplank, 1988). "The main advantage of direct manipulation is that the computer system attempts to model everyday operations more directly than older style of interface, thus making them easier to learn. This makes them appealing to novice users and fairly intuitive for infrequent tasks" (Maguire, 1990).

Due to the importance of direct manipulation in the development of modern user interface, a detailed discussion on this topic is given in the next section.

2.3 DIRECT MANIPULATION

Shneiderman (1983) first coined the term 'direct manipulation interface' to refer to an interface having the following characteristics: continuous representation of the objects of interest; rapid incremental reversible operations whose impact on the object of interest is immediately visible; and physical actions such as labelled button pressing instead of complex syntax and command names. Shneiderman describes a number of systems as examples of direct manipulation. The systems include screen editors with the 'What You See Is What You Get' principle, spreadsheet programs, CAD/CAM applications and video games.

Several other authors attempt to describe the central idea behind direct manipulation. According to Booth (1990), direct manipulation is when "user's actions should directly affect what happens on the screen to the extent that there is a feeling of physically manipulating the objects on the screen". Hutchins *et al.* (1986) describe direct manipulation as "the feeling of involvement directly with a world of objects rather than of communicating with an intermediary".

In addition to Shneiderman's description, Hutchins *et al.* (1986) give a thorough discussion on the concept of direct manipulation. They relate direct manipulation to the concept of directness which comprises of two factors, i.e. distance and engagement. Distance refers to the gap between a user's thoughts and physical requirements to accomplish the tasks and engagement refers to the degree of involvement the user experiences with the system.

Berry (1992) classifies the different interaction techniques according to the degree of direct manipulation. The typed command is at one end of the scale, providing the least feeling of direct manipulation. Drag and drop technique is at the opposite end, providing the most direct manipulation feel. Pop-up and pull-down menus are in between the two. Figure 2.2 shows the degree of direct manipulations among the interface techniques. Berry considers pop-up menu provides a higher degree of direct manipulation than pull-down menu. This is due to the fact that the former dynamically appears beside the object of interest and offers commands related to the particular object in its current context.

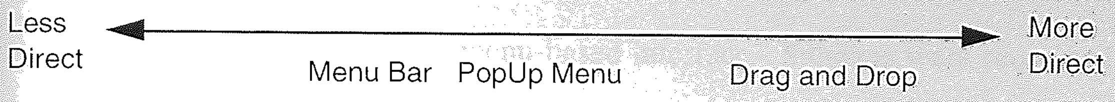


Figure 2.2 : Degree of Direct Manipulation

The summary of advantages of direct manipulation are: (Shneiderman,1983)

- Novices can learn basic functionality quickly.
- Experts can work extremely rapidly.
- Casual users can retain operational concepts.
- Error messages are rarely needed.
- Users can immediately see if their action are furthering their goals.
- Users experience less anxiety because the system is comprehensible and actions are easily reversible.

Hutchins *et al.* (1986) and Benbasat & Todd (1993) argue that none of these advantages is unique to direct manipulation systems. What is to be determined is the effectiveness of direct manipulation approach as compared to other style of interfaces.

Even though, there are so many papers which discuss the topic of direct manipulation, only few of these were empirical studies. Several experimental studies compared a direct manipulation interface to alternative ways of interacting with the computer. Shneiderman & Margono (1987) compared the use of the MS-DOS command language to the Macintosh direct manipulation by novices for performing simple file manipulation tasks. The result showed that users of direct manipulation perform the task faster, with less errors and they were generally more satisfied with the interface. According to the authors, this was due to the facts that users do not have to recall commands and the efficient handling of the mouse and the pull-down menu in the direct manipulation interface.

Benbasat & Todd (1993) conducted an experiment to study the effectiveness of direct manipulation compared with the menu-based interface. Subjects were requested to carry out number of simple tasks using an electronic mail system. The study reveals that subjects working with direct manipulation interfaces completed the task faster than those with menu-based interface. However, this difference in time was not significant when the task was repeated for the third time. This indicates that the advantage of direct manipulation diminishes after a learning period. There was no difference in the numbers of errors made by using direct manipulation as compared to menu-based interface.

The problem of direct manipulation interface is that users may have to spend more time with the objects (Morse & Reynolds, 1993). Users have to select objects repeatedly and then select the operations to perform on those objects. Even though, rubber-band grouping of objects is supported, it may be quicker for an experienced users to use the 'wildcard' feature provided by many operating systems. In general, direct manipulation interface is not that powerful and flexible in comparison with command languages (Myers, 1992; Buxton, 1993).

Direct manipulation interfaces are mostly implemented in a Graphical User Interface (GUI) environment (Bourne, 1992; Nielsen, 1993a). A continuous representation of objects is manifested in a GUI as information or icons in a window. Physical actions are normally implemented by point and click metaphor and reversal of actions are easily effected in the typical GUI format.

2.4 GRAPHICAL USER INTERFACE

The Graphical User Interface (GUI) originated from the research carried out at Xerox's Palo Alto Research Center (PARC) in the 1970s. The Apple Macintosh was the first computer system that introduced GUI to the market in 1984. Most of the ideas of the GUI were adopted from the development of Smalltalk, Star and Tajo at the Xerox Research Center (Gosling *et al.*, 1989). The introduction of the Macintosh was a commercial success. The Macintosh GUI has caused a widespread acceptance among nontechnical users who were traditionally unreceptive toward computing. Because of its success, the Macintosh has became a model by which GUIs are judged (Marcus, 1992). The definition (Hayes & Baran, 1989) of the Macintosh GUI includes:

- Pointing device, typically a mouse.
- Menus that can appear and disappear under the control of pointing device and cause execution of program code.
- Windows that graphically display what the computer is doing.
- Icons that represents applications files, directories and so on.
- Graphic metaphors such as dialogue boxes and buttons that select the next computer action.

Today's GUIs come in many varieties. For example, some GUIs do not use icons, while on others, icons are optional or available only sometimes.

The primary purpose of the GUI is to facilitate user-computer communication by insulating the complexities of computer hardware and software from the user. The GUI causes the application program to look more like real world functions and much less like a computer program. As a result, the novice enjoys immediate ease of use and the experienced user moves more quickly to the advanced features of the application.

The GUI is still in its infancy (Mandelkern, 1993). At present the GUI can be broadly classified into three major camps. The first camp is the IBM's System Application Architecture (SAA), which includes Microsoft's Windows and Presentation Manager.

The second camp is the UNIX systems, usually built upon X-Window. The UNIX based NeXT is one major exception, since it does not use X. The third camp is the Macintosh.

Developing a GUI software is not an easy task. This is due to the fact that such software is concerned with handling asynchronous events and its primary intention is to conceal the complexity from users (Gay, 1993; Huan-Chao, 1991). Furthermore, developers have to adopt object-oriented methods rather than traditional functional methods (Powell, 1990; Nielsen, 1993a). In one recent study, Nielsen (1993a) found that it is quite difficult for interface designers to change from a function-oriented interface to object-oriented one.

2.4.1 WINDOWS

A Windowing system allows the display of information about multiple applications at the same time. The system includes the programming tools and commands for building windows, menus, dialogue boxes and other items on display. It manages how windows are created, resized and moved on-screen, and how the user moves from one window to another, among other functions (Hayes & Baran, 1989).

Version 11 of the X Window System is accepted as an informal standard by many vendors and is being developed as a formal standard. Originally developed by the Massachusetts Institute of Technology (MIT), X Window implements the client-server model for distributed computing environments. The networked based windowing protocol allows the application to be separated from display process. In the X Window System, the term client refers to a potentially remote application program that wishes to display information on one or more workstations. A number of clients may share resources and display services provided by the workstations. The server is a process running on a display device, whose function is to display client output and return messages from the user interaction to the client (Gosling *et al.*, 1989).

X window is not a complete user interface since it does not define the look and feel of the application. It is just a windowing system shared by a group of different GUIs.

2.4.2 ICONS

The introduction of the icon-oriented user interface was the result of research in object-oriented programming, where computer scientists believed that typical human communication is object-centered (Blankenberger & Hahn, 1991). The effectiveness of this form of interfacing, however, has been a controversial issue. There are those who regard the use of icons as an efficient means of interaction and there are those who argue that in many applications the use of icons causes more confusion since they fail to represent their intended meaning.

In support of their use, it has been claimed by Lodding (1983) that icons can reduce the complexity of the system, thus, make it easier to learn. The merits of icons in user interface are also discussed by Marcus (1992). Icons are visually appealing, easy to understand, use less space than the equivalent in words and more importantly, they can replace written languages and contribute to a form of communication which has the potential of being globally meaningful.

The use of icons as a form of communication does not really live up to its expectations (Roger, 1989). On some occasions, the intended meaning of icons are easily understood, and for others icon representations may be misleading. Different meanings can be attributed to a single icon. An icon used by MacDraw to represent zoom in and zoom out is shown in figure 2.3a. Even though, the representation takes less space, its functions are not easy to guess. On the other hand, the Superpaint program uses a pull-down menu with the entries zoom in and zoom out to represent the same function.

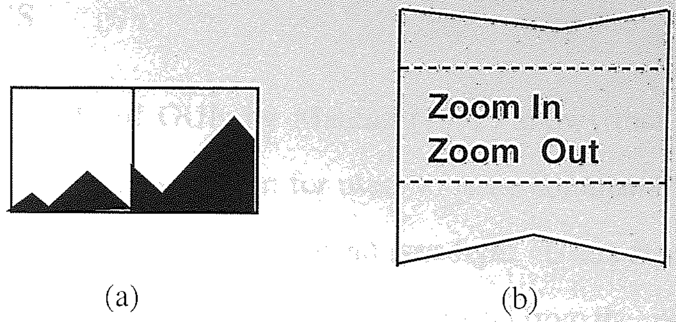


Figure 2.3
(a) MacDraw Icon Representation
(b) Superpaint Pull-Down Menu

Figure 2.4 is a mail box that has been widely used in the United States. An icon of this is been adopted by xmh (Peek, 1991) to represent an electronic mail application. The icon should be appropriate for the users in the United States, but may not be practical for users from different countries where the mail box used in their countries is not the same as in the United States. The function of the mail box is not the same from one country to another. In the United States, people send and receive letters through the mail box, but in England, mail box is only used to send letters. Therefore, to design icons for an international recognition is not an easy task.

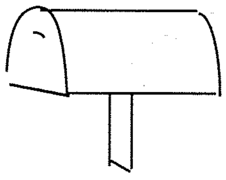


Figure 2.4 : Example of a Mail Box

Recent research by Benbasat & Todd (1933) on the performance of casual users on a electronic mail system reveals that there were no advantage associated with iconic representation as compared to text-based representation. The authors claim that the finding are consistent with a number of earlier studies carried out by different researchers.

2.4.3 MENUS

A comparative study of GUIs by Marcus (1992) shows that popular windowing systems provide a menu as a mean for users to issue command to the system. They mostly adopt select-and-operate command paradigm. Users first select the object to which operation is to apply, then choose the command from the menu.

Different windowing systems provide different types of menu which can be classified according to their appearance and behaviour.

Pull-Down Menus

Pull-down menus appear as an extension of a horizontal menu bar (Ziegler & Fahnrich, 1988). The menu bar constantly holds the titles of the pull-down menu. A pull-down menu appears below its title when the title is selected. A pull-down menu is more useful than pop-up menu especially when there are many functions represented. An example of a pull-down menu are as shown in figure 2.5. The figure shows that the title edit is selected and the edit menu is displayed.

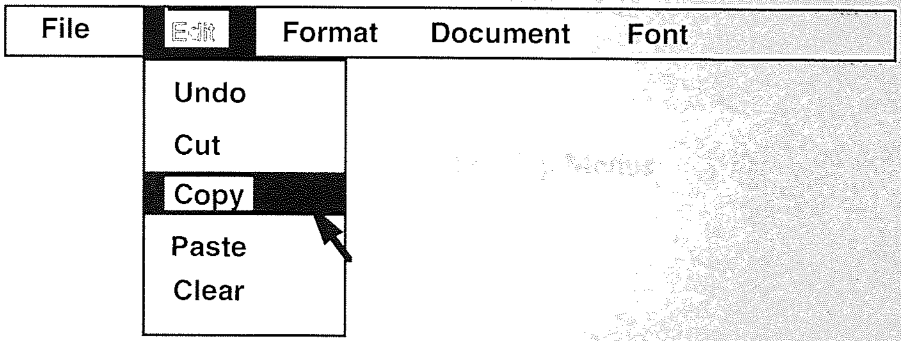


Figure 2.5 : Pull-Down Menu

Pop-Up Menus

A pop-up menu appears in response to a click with a pointing device such as a mouse within a particular region of the display. The region could be an entire screen or some well-defined area such as a window or a view of a window.

Interactions in a Smalltalk system is mainly through pop-up menus (Goldberg & Robson, 1983). Different pop-up menus are provided by pressing the different buttons on a mouse. Figure 2.6 are two examples of pop-up menus implemented in Smalltalk-80 which are obtained by pressing the middle and right mouse buttons respectively in a workspace window. The left mouse button is reserved for text selection. The menu displayed by pressing the middle button holds commands related to the selected view. Since the workspace window contains editable text, the commands provided are related to text manipulation. The menu obtained by pressing the right button hold commands related to the window itself.

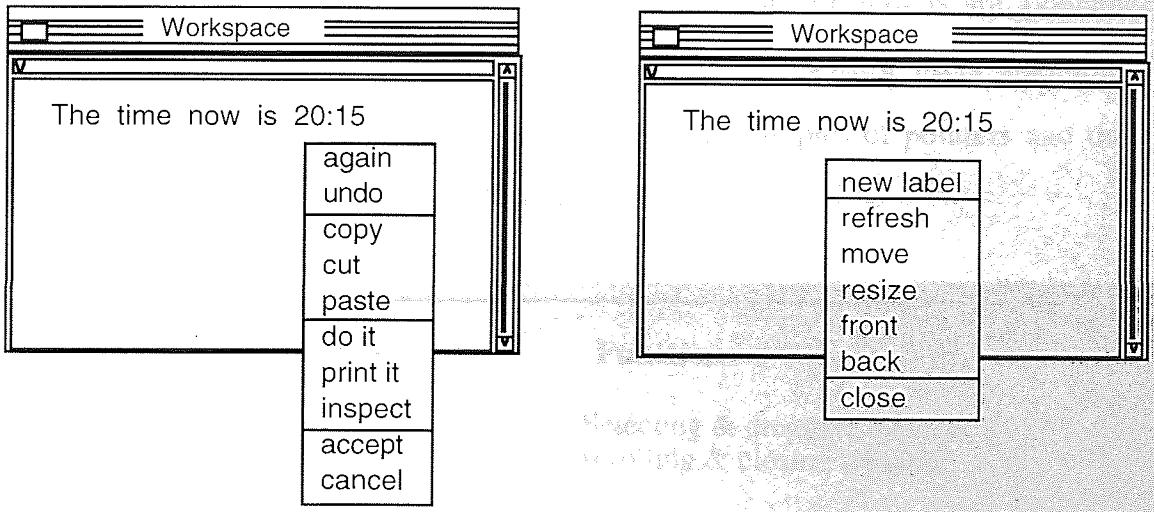


Figure 2.6 : Pop-Up Menus

Permanent Menus

Both menus, the pull-down and pop-up menus are displayed only when users make a selection. Users usually have to click and drag the pointer to make their selection on these menus. Permanent menus are the menus that are continuously displayed on the screen. Selection on the permanent menus can be made simply by clicking on the right choice. Application programs that use permanent menus include paint programs such as PC paintbrush and MacDraw. A combination of permanent and pull-down menus are stay-up menus. Users have the choice of having the stay-up menus continuously displayed on the screen or just behave as a pull-down menu.

2.4.4 POINTERS

A pointer is associated with a pointing device such as a mouse, track ball, or a joy stick (Tullis, 1988). When users move the pointing device, the corresponding pointer makes the same movement on the screen. Users' actions such as mouse clicks causes an object over which the pointer is positioned to be selected.

The different images of pointers provide users with a meaningful visual feedback to indicate the current state of the system or which modes of actions are available. Changing the shape of the pointer image is very effective because users' attention is focused near the cursor. Figure 2.7 shows typical examples of pointers and their effects in some of the Macintosh applications.

Pointer	Name	Functions
↖	Arrow	Selecting & dragging objects, scrolling & closing window
+	Crosshair	Drawing or stretching graphic objects
I	I-Beam	Selecting and inserting text
✚	Plus sign	Selecting field in array
⌚	Wristwatch	Indicating that computation is in progress

Figure 2.7 : Macintosh Pointer Images

2.4.5 CONTROLS

Well-designed controls allow the GUI to produce a sense of directly manipulating physical objects (Marcus, 1992). "Controls provide users with familiar tools and formats for responding to computer's need for information" (Apple, 1992). Examples of controls include buttons, checkboxes, radio buttons, sliders and scrolling lists.

Buttons

A button is usually an image which resembles a pushbutton. Figure 2.8 shows three typical buttons in a dialogue box. The title of the buttons represent commands that can be performed in a particular system. Unlike other controls, buttons have the ability to produce an immediate system response. Unlike menu system, users are not required to navigate through the menu entries to make a selection (Marcus, 1992).

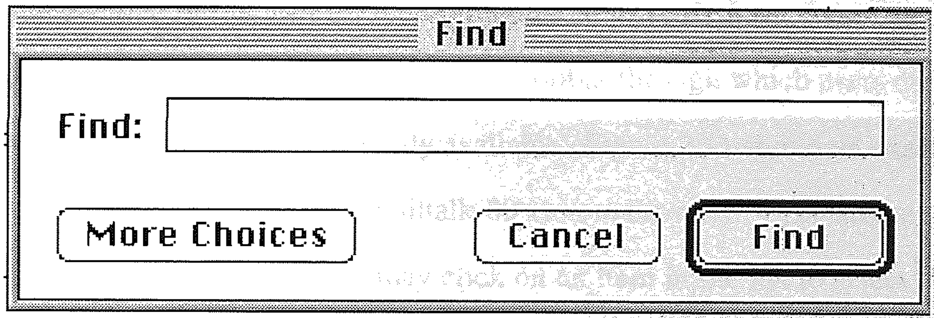


Figure 2.8 : Buttons in a Dialogue Box

Radio Buttons

Radio buttons act like the buttons on a car radio, whereby only one button can be on at one time (Apple, 1992). Figure 2.9 shows a typical set of radio buttons. The active setting has a dot in the middle of the button. Figure 2.9 shows that *A4 Letter* size paper is selected. Clicking one button in a group activates that particular button and turns off whichever button was on before.

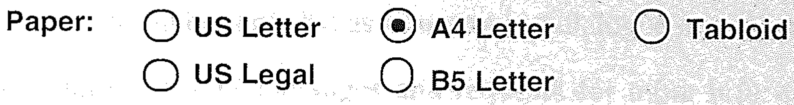

Paper: ○ US Letter ● A4 Letter ○ Tabloid ○ US Legal ○ B5 Letter

Figure 2.9 : Set of Radio Buttons

Checkboxes

Unlike radio buttons, checkboxes allow users to make multiple selections from the choices offered. Users can select all, some or none of the choices by toggling the checkboxes to display or suppress the check mark. In the example shown in figure 2.10, characters and words are selected.

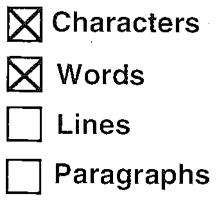


Figure 2.10 : Set of Checkboxes

Scrolling Lists

Scrolling lists are used to present an unbounded list of items in a small well-defined region (Sun Microsystems, 1990). A scrolling list in its basic form comprises of a list items such as a list of customer names and a scrollbar through which users can look at more items in the list that aren't currently available. Figure 2.11 shows an example of a scrolling list which is used by the Smalltalk-80 system browser. Typically, a scrolling list is a read-only control and users may click on an item in the list to select it or they may scroll through the list to browse through its content without selecting anything.

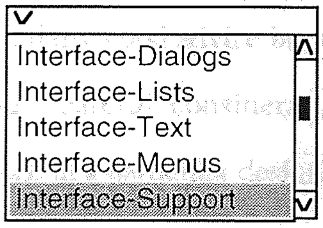


Figure 2.11 : Scrolling List

Dialogue Boxes

A special window which can be regarded as a control is a dialogue box (Marcus, 1992). A dialogue box usually displays messages and requests for more information from users or allows users to select options (Apple, 1991). An example of a dialogue box is the alert box which is designed to warn users from accidentally losing some of their work or quitting an application. Figure 2.12 shows an alert box which is displayed when users try to quit Smalltalk-80 from the Macintosh Menu bar.

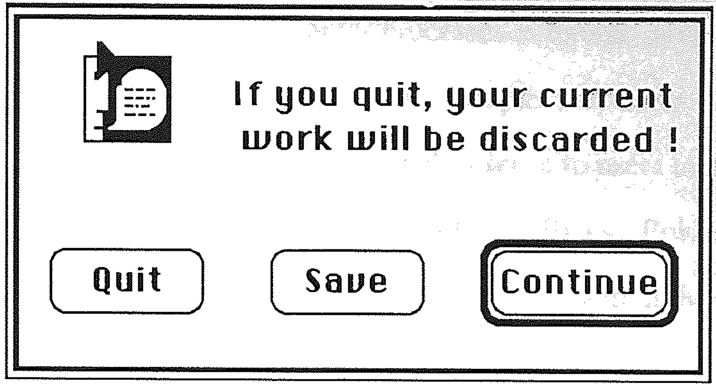

If you quit, your current work will be discarded !

Figure 2.12: Alert Box

2.5 USER INTERFACE DESIGN PRINCIPLES

Numerous authors discuss the general design principles of user interface. A comprehensive discussion of the principles is given by Shneiderman (1992). He suggests eight golden rules of dialogue design. Smith *et al.* (1990) discuss the principles used in the design of the Star User Interface. Apple (1987) also lays down its own principles in designing user interface.

Most of the principles are potentially good advice but they are too general to be directly applied (Smith, 1986). Thus, careful consideration is required to interpret the principles so that it is appropriate in a particular design.

2.5.1 APPLE MACINTOSH USER INTERFACE DESIGN PRINCIPLES

Apple (1987) describes ten fundamental principles of the Apple Desktop Interface. The Macintosh Human Interface Guidelines (Apple, 1992) includes one more principle, i.e. the principle of modelessness. The principles are relevant to a graphical user interface environment. The explanation of the principles may include opinions and suggestions from various sources. The principles are as follows:

1. Metaphors From the Real World

The use of real world metaphors for computer processes is to take advantage of users' direct experience with their immediate world. For example, users often use file folders to manage their paper documents. Therefore, it makes sense to users to store computer documents in folders that look similar to folders in their offices. Folders also can be renamed, moved around the desktop and deleted by throwing them in the trash.

The disadvantage of such a metaphor is that there is a limit to its capability. Many physical objects do not have enough power to manage the complexities of information technology (Smith *et al.,* 1990).

2. Direct Manipulation

Maguire (1990) states "direct manipulation is an interface style in which the user points at a visual representation of the task, manipulates it and immediately observes the results". The discussion on direct manipulation is already given in section 2.3.

3. See and Point

The traditional command-line interface requires users to recall the appropriate commands and type them into the computer with the right syntax. Users have to focus more on the complexity of the interface and this makes them distracted from their main task.

The see-and-point interface allows users to interact directly with the screen, selecting objects and performing actions by using pointing devices, typically a mouse. The Macintosh see-and-point interface can be categorised into two paradigms which are based on noun-then-verb user actions. In the first paradigm, the users select an object of interest (the noun) and then perform certain actions on the object (the verb). In the second paradigm, the users drag an object (the noun) onto some other object that has an action (the verb) associated with it (Apple, 1992).

4. Consistency

The very first principle suggested by Shneiderman (1992) is to 'strive for consistency'. The importance of consistency is stressed by Sun Microsystem (1989), by stating that "consistency enables users to apply previously gained knowledge to a new areas without having to learn from scratch. Users can figure out consistent products by detecting and learning the similar patterns."

Grudin (1989) challenges the principle of consistency. He forwards an example from everyday life of deciding where to keep different knives. Table knives, butter knives and steak knives should be placed in the same drawer. Such an arrangement of knives would make it consistent, easy to learn and easy to remember. But then, the most appropriate place to keep the putty knife is in a workbench drawer in the garage and the Swiss army knife packed away with the camping gear in the basement. By keeping the knives at different places, we have introduced inconsistency and increased the time to find them, but, common sense tells us to organise them according to the way they are used and the task they are involved. Grudin suggests that primary emphasis should be placed on the users' work environment rather than to strive for consistency.

Although the Macintosh desktop is one of the most consistent user interface, there are still instances of inconsistency. Obvious examples are the actions for deleting a file and for ejecting a floppy disk (Hix & Hartson, 1993). A way to delete a file is to select and drag the unwanted file to the trash icon. Users may carry out a similar action to eject a disk, i.e. select and drag the disk to the trash icon. Users are carrying out a similar action, but the results are different.

5. WYSIWYG (What You See is What You Get)

WYSIWYG is a phrase to describe what users can see from the display is indeed what he has got (Thimbleby, 1990). The result of users' choice is immediately displayed on the screen. In a WYSIWYG word processing environment, users can see directly the effect of making selected words bold or changing their font or font size. Therefore, users do not have to figure out themselves of how their documents shown on the screen will appear when they are printed on papers. This is due to the fact that there is no significant difference between what users see on the screen and what eventually gets printed (Apple, 1987).

6. User Control

"People learn best when they're actively engaged" (Apple, 1987). Thus, users have to be the initiators of actions rather than the responders (Gaines, 1981). In the case where users attempt to do something risky, the computer may interrupt them and warn them against the consequence of their action, but allow the action to proceed if users confirm that this is what they want. This strategy seeks to "protect users but allows them to remain in control" (Apple, 1992).

7. Feedback and Dialogue

Provision of immediate feedback is required in order to keep users aware of what is going on. Smith *et al.* (1990) state "it is disastrous to the user's model when you invoke an action and the system does nothing in response". Since no immediate feedback is provided by the system, some users of Openwin (Sun Microsystem, 1991) click an entry from the main menu several times. They keep on repeating their request because they are not sure whether the system has heard them or not. After some time users find that the screen is full with the same windows, equivalent to the number of requests they have made.

Feedback provided should be simple enough for users to understand (Apple, 1992). Most users would not know what to do when they encounter messages such as 'The application unexpectedly crashed. ID=13'. It would be more helpful if the message is well explained, for example 'Not enough memory was available for the computer to complete the task'.

Nielsen (1987) categorises different types of feedback according to their degrees of persistence in the interface. Some feedback is only relevant for a certain period of time. For example, transformation of the image pointer into a rolling ball is only relevant until processing is completed. Other feedback is relevant until the user explicitly acknowledges it. An example in this category would be a confirmer requesting whether the user really wants to delete a file. Finally, some feedback is so important that it has to remain as part of the interface. An example might be the remaining free space on a hard disk.

Feedback is very important especially for operations which require a longer time to complete. Card *et al.* (1991) provide the basic guidelines regarding response times and the user's feeling and behaviour. The time limit versus the user's behaviour is summarised in table 2.1.

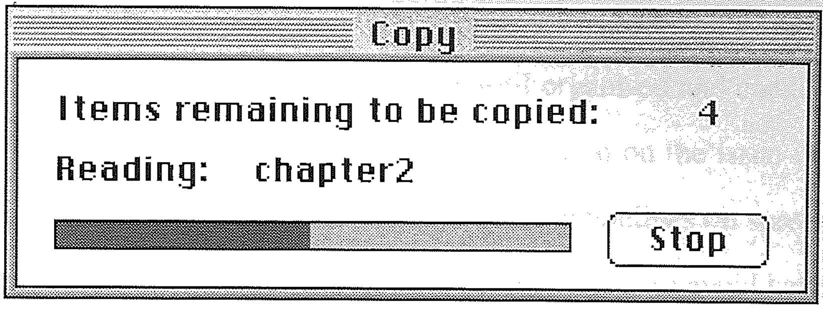
The time limit in second	**User's Feeling/Behaviour**
0.1	the system is reacting instantaneously
1.0	flow of thought stays uninterrupted
10	attention still focuses on the dialogue

Table 2.1 : Response Time vs. User's Behaviour

Myers (1985) suggests that percent-done progress indicators should be used for operations which are longer than 10 seconds. Figure 2.13 shows the Macintosh progress indicator. The advantages of using progress indicators are: (Nielsen, 1993b)

- They indicate that the system is currently working on the user's problem.

- They indicate approximately how long the user is expected to wait.
- They provide something for the user to look at, thus making the wait less painful.



Items remaining to be copied: 4
Reading: chapter2

Figure 2.13 : Progress Indicator

Using the percent-done progress indicator for operations which require between 2 and 10 seconds would violate the principle of display inertia (Nielsen, 1993b). The indicator displays itself for a short while and then disappears. The display happens so fast that it causes irritation to the user.

8. Forgiveness

Apple (1987) defines forgiveness as ".. letting users do anything reasonable, letting them know they won't break anything, always warning them when they are entering risky territory, then allow them either to back away gracefully or plunge ahead, knowing exactly what the consequences are". This principle encourages users learning to use the application by exploration. They can test various functions supported by the application without damaging the system.

9. Perceived Stability

The computer environment changes as users interact with it, but users should feel that there are some stable reference points to count on (Apple, 1987). The Apple Desktop interface uses a two-dimensional space on which objects are placed and define a number of graphic elements such as menu bar and window border to achieve visual sense of stability. To achieve a conceptual sense of stability, the interface provides a finite set of objects and a finite set of actions to perform on those objects. When actions are not available, they are not eliminated from a display, but are merely dimmed (Apple, 1992).

10. Aesthetic Integrity

Aesthetic integrity means that "information is well organised and consistent with the principles of visual design" (Apple, 1992). A discussion on the issue of good visual design has been given by Tullis (1988). Some of the guidelines on screen design are: eliminate unnecessary information; a balanced layout - this is to avoid having too much information on any side of the screen; use plenty of empty spaces especially around block of text; group related information logically.

The guideline is not only for aesthetics reason. An empirical study by Tullis (1981) shows that users complete tasks 40% faster when a screen format is improved. Therefore the way information is presented on the screen partly determines the success of the user interface.

Some users have to spend quite a long time at the computer screen. Hence, the pleasantness of the interface is quite important. Users also gain some impression about any application from the way it looks. Generally, they are likely to be afraid of an application which is visually overwhelming (Sun Microsystems, 1990).

11. Modelessness

The idea of modelessness is to allow users to do whatever they want when they want to on the computer and in an application (Apple, 1992). Thimbleby (1990) states that the user find modes difficult because much important information is hidden. Therefore it can be easily forgotten or not noticed by the users. Furthermore a mode may lock the user into one operation until that operation is completed (Apple, 1992).

This does not means that modes should be completely avoided. Thimbleby (1990) also states that modes do have advantages for the user. Modes protect the user from making mistakes and help the user avoid interface clutter.

2.5.2 OPEN LOOK USER INTERFACE DESIGN PRINCIPLES

OPEN LOOK (Sun Microsystems, 1990) suggests three basic user interface principles. These principles are simplicity, consistency and efficiency.

1. Simplicity

Users want a quick success which, in turn give them the impression that the applications are easy and intuitive (Sun Microsystems, 1990). This encourages further exploration of the capabilities of the system.

One of the factors that contribute to simplicity is clearly labelled controls and the term used should be simple enough for users to understand. Berry (1992) points out that the term 'composed' is a technical term and it is not appropriate for users. This term is widely used by many electronic mail applications.

Simplicity is one of the principles pursued in the designed of the Star user interface (Smith *et al.* 1990). Smith *et al.* (1990) state that "typically, a trade-off exists between easy novice use and efficient expert use. The two goals are not always compatible". They suggest the rule stated by Alan Kay to approach this problem, that is: 'Simple things should be simple; complex things should be possible'. Smith *et al.* further state that "simplicity, like consistency, is not a clear-cut principle".

2. Consistency

This principle has already been discussed in section 2.5.1.

3. Efficiency

"An efficient application minimises the number of steps required to perform an operation and provides users with short-cuts" (Sun Microsystems, 1990). Short-cuts allow users to do more things with minimum number of steps: less mouse travel, fewer keystrokes, and less switching between the use of keyboard and the mouse. In some cases users may use both the keyboard and the mouse simultaneously

Recent word processing applications provide short-cuts by having buttons which can be turned on and off on the screen so that users can have a fast access to the most frequently used functions. Figure 2.14 shows the ClarisWork 2.0 short-cuts facility. Expert users can access commonly used functions such as copy, cut and paste simply by clicking on the right buttons.

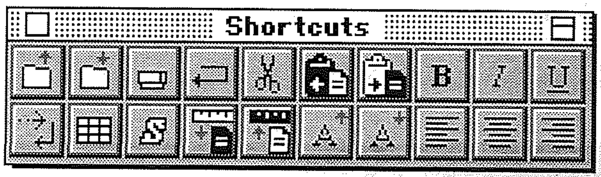


Figure 2.14 : ClarisWork Shortcuts

Macintosh System 7 provides a short-cut by allowing users to access any program, document or disk by putting it or its alias in the Apple menu (Apple Computer, 1991). Figure 2.15 shows that the Microsoft Word application can be invoked by selecting the entry from the menu. Usually users have to carry out a number of steps to invoke the application: open the hard disk; find and open the folder that contains the application; invoke the application; close the opened windows to prevent screen clutter.

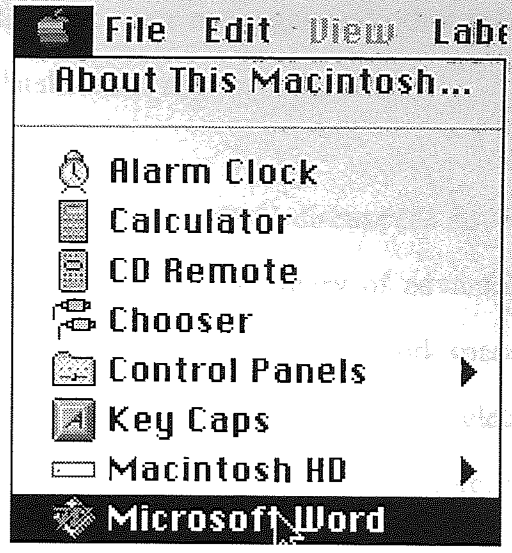


Figure 2.15 : Invoking Microsoft Word from the Apple Menu

Sun Microsystems also suggests the concept of 'progressive disclosure' to increase the efficiency of applications. The concept is derived from the design of consumer appliances and stereo systems. For example, a stereo system has its complex controls placed behind panels. The idea of such arrangement is to make the system looks simple but provide the advanced features desired by the advanced users. Pop-up menu and pop-up windows are similar to the controls behind the panel of a stereo system. Only those commands that are frequently used are visible on the screen.

2.6 ONLINE HELP FACILITIES

Shneiderman (1992) points out that "even though increasing attention is being paid to improving user interface design, there will always be a need for supplemental materials that aid users, in both paper and online form".

Duffy *et al.* (1992) define online help as the online delivery of performance-oriented information. It is designed to answer the question 'how do I?'. The user of online help is trying to complete some task in an application. They also distinguish the online help system from other types of online assistance. The following examples are not regarded as online help: online tutorial; error messages and other information retrieved from online services such as databases of newspaper articles.

Duffy *et al.* (1992) and Shneiderman (1992) discuss the advantages of having online help available on the computer. The summary of advantages are: information is available whenever the computer is available; extra work space is not required to open up manuals; information can be easily updated at low cost; electronic indexing and text searching facility can be incorporated; less expensive to store, reproduce and distribute; graphic, sound and animation may be used to explain complex actions.

A study by Elkerton (1988) reveals that many online help systems fail to help users solve their current problems. He identifies a few reasons why such failure occurs. One of the most important reasons is that designers regard online assistance as nothing more than an electronic version of a printed manual. These types of online manual may be useful for skilled users, but completely ineffective for novice and intermittent users.

Sellen and Nicol (1990) list five main reasons why users avoid using help facilities: difficulty in finding information; failure to obtain relevant information; difficulty of switching between the help and the working context; complexity of the help interface and; the quality and layout of help information. They recommend different help interfaces for different kinds of help, from the basic introduction to the application to procedural and navigational questions.

When users need help, they must switch from a problem-solving mode to a learning mode. Clark (1981) found that the mode switch sometimes causes users to forget why help was requested in the first place. Houghton (1984) suggests that, one way of making help messages less disruptive is to place them on the screen simultaneously with the problem.

The solution to the problem is to develop a context-sensitive help, whereby help messages are displayed within the users' working context (Apple, 1992; Shneiderman, 1992). This type of help allows users to see problem and solution simultaneously, and user's application remains active or in control. Therefore, users are not bothered by the mode change and are not required to memorise help messages before returning to their application.

Macintosh System 7.0 (Apple, 1991) is the first system to provide a context-sensitive online-help on the Macintosh. The help is displayed in a small balloon next to the object pointed by the mouse pointer. The help describes menu commands, dialogue boxes, icons and windows. Users may turn on and off Balloon Help from the menu bar on the top right of the screen. Once Balloon Help is turned on, the balloon for an items appears when the user move the pointer to an item and stays on the screen until the user move the pointer away from the item. Figure 2.16 shows an example of a help balloon.

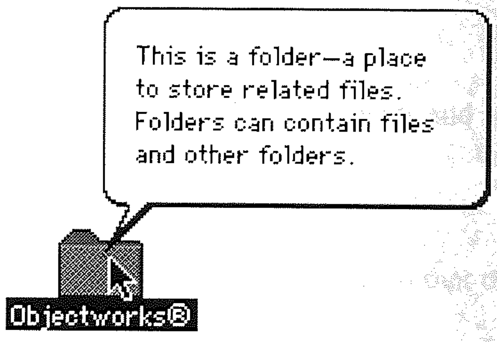
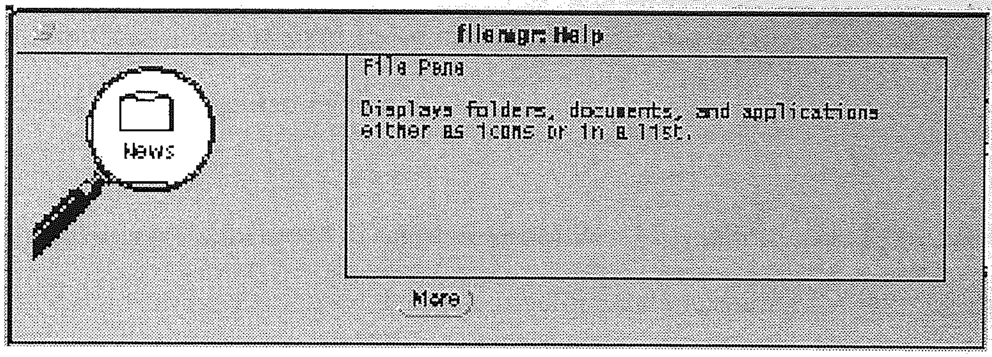

This is a folder—a place to store related files. Folders can contain files and other folders.

Figure 2.16: Help Balloon

A context-sensitive help is also supplied by OPEN LOOK (Sun Microsystems, 1989). The way to obtain the help messages is slightly different from the System 7 context-sensitive help. Users have to move the pointer to the object for which they require help and then, press HELP key on the keyboard. A pop-up window will then display information relevant to the specific object pointed by the mouse pointer. The window is shown in figure 2.17. Users may retrieve additional, application-specific information by pressing the button labelled 'More' in the help window.


filemgr: Help

File Pane

Displays folders, documents, and applications either as icons or in a list.

Figure 2.17 : Help Window on OPEN LOOK

2.7 MEASURING USABILITY

The purpose of the evaluation is to measure the usability of the prototype by different categories of users. Visvalingam (1988) defines usability as "the features that a system possesses, over and above its basic functions, which promote ease of use".

Nielsen (1993b) states "usability has multiple components and is traditionally associated with these five usability attributes:

- **Learnability** - the system should be easy to learn so that the user can rapidly start getting some work done with the system.
- **Efficiency** - the system should be efficient to use, so that once the user has learned the system, a high level of productivity is possible.
- **Memorability** - the system should be easy to remember, so that the casual user is able to return to the system after some period of not having used it, without having to learn everything all over again.

- **Errors** - The system should have a low error rate, so that users make few errors during the use of the system, and so that if they do make errors they can easily recover from them. Further, catastrophic errors must not occur.

- **Satisfactions** - The system should be pleasant to use, so that users are relatively satisfied when using it; they like it".

Nielsen also suggests methods to measure usability. Learnability can be measured by selecting users who have not used the system before and measuring the time for them to reach a specified level of proficiency in using it. The system's error rate can be measured by counting the number of mistakes made by users while accomplishing some tasks. Subjective satisfaction can be measured by asking the users their own opinion about the system.

To measure the usability of the prototype, users are requested to execute a sequence of tasks given to them and then, answer questionnaires based on their experience using the prototype. Two types of data are to be generated from the evaluation. They are:

- **Qualitative** - These are non numeric data and results, such as problems users had while using the system. The data are acquired by observing users while executing tasks and from users' verbal or written opinions.

- **Quantitative** - These are numeric data which are acquired from users' opinion rating based on their experience using the system.

Even though qualitative data are not used in the measurement of usability, they are quite important in determining the problems with the interface. These data are secondary data which may complement the measurement of usability. Users may wish to express reasons why they are satisfied or dissatisfied with the interface.

2.8 DEVELOPING A USABILITY EXPERIMENT

Developing an experiment for the evaluation involves the four main activities (Hix & Hartson, 1993):

- Selecting appropriate test users.
- Developing tasks for users to perform.
- Preparing questionnaires.
- Determining procedures for the evaluation session.

1. SELECTING TEST USERS

One of the first activities in the evaluation of the prototype is to select users for the evaluation. Recently, terms such as test users, participants and usability evaluators are used in human factors literature to indicate representative users taking part in evaluating user interface (Nielsen, 1993b; Hix & Hartson, 1993). This is to emphasis that it is the system that is being tested and not the users. The users are volunteers who help designers to evaluate the systems.

Test users and users will be used throughout this chapter to refer to users participating in the experiment. These test users should be chosen randomly among the expected users of the prototype. This means that they should also be representative of different categories of users within the targeted users population. Since the prototype is developed for academic communities, test users for the evaluation of the prototype include different categories of academic population such as the lecturers, computer officers, secretaries and post-graduate and undergraduate students.

Nielsen (1993b) found that users with knowledge of the problem domain of the interface usually provide more useful feedback than those who do not have this specific knowledge. In the case of the evaluation of the prototype, users should have some experiences using electronic mail or BIDS. This does not mean that users who do not have any knowledge of the problem domain are to be excluded from the evaluation. They are still potential users of the system and their opinions about the system and the way they react to it can be compared with the group who is familiar with the problem domain.

The different backgrounds and experiences of users are regarded as important information for the evaluation. They are required when analysing the results of the evaluation. Due to its importance, the information is recorded before the evaluation starts and is placed at the beginning of the questionnaire form.

Hix & Hartson (1993) point out that "it is always nice and sometimes necessary to offer bribes to get users". Bribes could be any appropriate and inexpensive token for the time, ideas and cooperation given by users. These could be just refreshments such as free coffee and biscuits.

2. SELECTING TEST TASKS

Test tasks should be selected to represent functions supported by the prototype. The tasks should be able to demonstrate general capability of the system. They should also provide a reasonable coverage of the most important parts of the user interface (Nielsen, 1993b). They are the ones that users are expected to perform often, and therefore should be easy for users to accomplish (Hix & Hartson, 1993).

The instruction to carry out these tasks are listed on a paper in the order in which users will be asked to perform them. The instructions are provided in brief and should mention what the users should do rather than how the users should do it (Hix & Hartson, 1993). Hence, users have to explore the system in order to accomplish the task successfully. A well-designed user interface should provide enough feedback and guidance to help users using the system.

Ravden & Johnson (1989) state six advantages of having test users performing tasks during evaluation. The summary of those advantages are:

- Tasks represent the work for which the system is designed. Hence, they provide the most effective way of demonstrating the system's functionality.
- Enables users to see the application as a whole and not just a series of screens and actions.
- Users can be exposed to different aspects of the user interface.
- Many significant problems and difficulties can only be realised after carrying out tasks.
- In some cases, aspects of usability can only be measured by using the system.
- Important information can be gathered by observing and recording problems and difficulties experienced by users when interacting with the system.

3. DEVELOPING QUESTIONNAIRE

Questionnaires provide the most effective and the least expensive method for evaluating a system. Karat (1988) points out that usability can be evaluated simply by asking users about their experience in using the system. Since the selected test users are among the potential users of the system, their opinions about the system being evaluated have to be considered.

The effectiveness of the questionnaire is quite obvious for some aspects which are hard to measure objectively such as the users' subjective satisfaction (Nielsen, 1993b). The replies given by users to such questions are subjective, but when replies from multiple users are averaged together, the result is an objective measure of the system's pleasantness. Nielsen (1993b) further states that the questionnaire is the most acceptable method of assessing the usability of a system.

Several researchers have laid down some guidelines regarding the questionnaire (Nielsen, 1993b; Hix & Hartson, 1993):

- It is recommended to use a short questionnaire. If possible, limit the questionnaire to a single page. A short questionnaire stands a better chance of receiving attention from the users.
- Ask questions to which you really want to know the answer.
- Rating scale should be the same throughout the questionnaire.
- At least 30 users should be involved in the evaluation.
- Pilot tests are required to prevent misunderstanding.

4. DETERMINE PROCEDURES FOR EVALUATION SESSIONS

The success of an evaluation depends on the feedback of the users. Ravden & Johnson (1989) point out that users must be given a clear explanation of the system, the purpose of the evaluation, the importance of their participation and what are expected from them at the different stages of the evaluation.

The evaluation of the prototype can be divided into two main stages:

- Executing tasks - During this stage, users are required to execute the test tasks which are assigned to them. These are the minimum number of tasks which allow them to have an insight into the system. They are not limited by the tasks assigned to them. They are encouraged to explore the system further in order to have a better understanding of the system.

- Answering questionnaire - Users are requested to answer the questionnaire. The answers to the questionnaire should be based on their experiences using the system. Root & Draper (1983) reveal that users give more useful answers if they are given questionnaire immediately after using the system.

It is also important to stress to the users that the purpose of the evaluation is to evaluate the interface and not to evaluate them. Some users may be afraid that participation in this kind of session reflects their own weaknesses in using computers.

Users are given written instructions about the evaluation and its procedures. This is the easiest way to make sure that all users are given similar instructions before they start their evaluation. This will ensure consistency and remove some of the variances from the test sessions (Hix & Hartson, 1993).

An evaluation form which comprises information regarding the evaluation, procedures of evaluation, selected test tasks and questionnaires are given in advance to the users. The form will be given when they have consented that they are ready to volunteer and they have made an appointment for that. They are expected to bring the form during the evaluation session.

Some authors like Hix and Hartson (1993) require users to sign an agreement before participating in evaluation. Nielsen (1993b) argues that signing an agreement will create negative results such as anxiety for the users participating in evaluation.

Information regarding the evaluation include the following:

- The description of the system.
- Why the system requires evaluation and the aim of evaluation.
- Why they are being asked to participate.
- The time required for the evaluation session.
- What are the evaluation procedures.
- What are they supposed to do.

Preparations for the evaluation such as reservation for the workstation that runs the prototype, paperwork and some refreshments for the evaluation should be ready by the time users come to the laboratory. Only one test user is allowed to attend an evaluation session at a time. It is assumed that the users have read all the instructions regarding the evaluation by the time they come to the evaluation laboratory. If they have not done so, then, they are given time to read the instructions. Users are then asked whether they have any questions regarding the evaluation.

The facilitator has to make sure that the session runs smoothly and efficiently. The room where the evaluation is conducted should be free from any form of noise or disruption. The users should settle comfortably in front of the prototype after they know the purpose of evaluation and what are they suppose to do.

The facilitator will observe how users perform tasks which have been instructed to them. In general, the facilitator should avoid giving any instruction on how to complete a task with which the participant is struggling. The most that the facilitator should do is to ask the participant, "do you need a hint for this task?". If the participant manages to complete the task after a hint is given, then the facilitator should note down that a hint is given to the users to complete the task.

The facilitator will take the following notes:

- The number of mistakes committed to complete each task.
- Questions asked as well as critics and suggestions pointed out by the users.
- Facilitator's observation.

The number of mistakes is the number of wrong actions taken by the users to accomplish the given task. The mistakes include clicking a wrong button or selecting a wrong command from a pop-up menu in application windows. They exclude clicking wrong mouse buttons or invoking pop-up menus to see available commands.

Questions, criticisms and suggestions collected during evaluation are qualitative data which could yield useful information in identifying trouble areas in an interface where users find difficulties using the system. The result could be used to suggest modification to the system.

Users are requested to answer the questionnaire after they have executed all the test tasks. They are allowed to access the system when answering the questionnaire, so that they can remind themselves of problems and explore other aspects of the system if they wish to do so. The facilitator will stay away from the users when they are answering the questionnaire, unless they have something to ask.

CHAPTER THREE

REVIEW OF USER INTERFACES TO NETWORKED SERVICES

3.0 INTRODUCTION

This chapter reviews user interfaces to networked services. It starts with a general review of existing user interfaces to some networked services. The difficulties involved when interacting with the services are highlighted. The user interfaces to the following services are reviewed:

- Geac Library System.
- Bath Information Data Services (BIDS).
- Usenet news reader (Tin and Xvnews).
- PP electronic mail.

3.1 REVIEW OF USER INTERFACE PROBLEMS

Some of the services were developed at the time when relatively little attention was paid to the design of the user interface. Most of the user interface designs were based on page screen display. Since the user interface part is embedded in the program codes, reconstruction of the user interface is almost impossible. Hence, users have to accept the difficulties in order to use the system.

Some networked services are old-fashioned menu-based interfaces even though they may be presented within a window of a graphical user interface. Menu systems are generally too modal. Users have to move from one menu to another in the hierarchy of menus in order to acquire the needed information. Users are not in control of the interface. Since most of the interactions use scroll mode, lengthy information may scroll up and pass the window.

To use the networked services, users must first know on which machine the service resides. Then they must log on to the machine by invoking its unique name. Once logged-on, different procedures need to be carried out to be connected to different services. BIDS users are usually required to log-on to a special host in their local institution that supports a connectivity service to BIDS. Having logged-on to the host, users have to log on to the service that provides the connectivity to BIDS. Finally users have to recall a special identification and password to be connected to BIDS.

Graphical user interfaces (GUI) have been developed for some networked services. The use of the modern GUI technology alone is not enough if consideration of other aspects of user interface is neglected. For example, users must be given some kind of notification such as a status indicator or transformation of the pointer to indicate that the system is currently busy and not ready to accept any input from users.

Most of the developments of user interface to the networked services are carried out independently by different developers. Thus, inconsistencies between the interfaces for different services are unavoidable. For example, most menu systems require the users to select their choice by typing an entry and then hit a *carriage return,* but in the case of NISS and Tin (a Usenet news reader), the users just need to enter an entry and it is automatically accepted by the system. As a result of inconsistencies, users have to keep on learning different styles of interaction every time they use a new service.

Inconsistencies among the user interfaces include the use of many different terms to refer to the same action. For example, different services use different terms to mean quit from the system. Examples of terms include exit, q (or quit), end, bye and logout. Thus, it is not surprising to see users leave the services without quitting after using them. They simply do not know how to quit.

Not all services support a help facility. The services which support the facility may not provide any indication that such facility is available and the ways to access the facility are not the same from one service to another. As a result not many users consult the help facility even though they are having difficulties in using the services. They would rather consult their colleagues or attempt to use the services by trial and error or abandon the services completely.

3.2 UNIVERSITY LIBRARY SERVICES

When the library computer was first connected to the network, an access to the service could be carried out by the command:

```
telnet library
```

The network would respond with the following messages:

```
Trying 134.151.16.73 ...
Connected to library.aston.ac.uk.
Escape character is '^]'.
```

A *carriage return* had to be entered immediately after the above messages, otherwise the connection would be closed. Since, no instruction was given by the system to enter a *carriage return*, users who were not familiar with the system were waiting without doing anything and timeout occurred. The request for a *carriage return* was not from the computer library, but from a gateway in the network. The gateway is used to solve incompatibility problem between the computer library and the network. To purchase a new gateway which can carry out an automatic connection required an investment of thousands of pounds.

The connection to the library was then rerouted through the Quipu Machine. Invoking the command **telnet library** will automatically establish a connection to the machine. An alternative is to specify the name of the machine with the command, i.e. **telnet**

quipu. The machine will respond with the menu of services as shown in figure 3.1. Users have to enter *library* at the login prompt to get connected to the library.

```
Welcome to Aston University

login as       de          to use the X.500 directory service
login as       library     to use the LIS OPAC system
login as       tx          to use the X29 (PAD) service
login as lynx to           browse WWW (set term=vt100)
(NO PASSWORDS are required)
login: library
```

Figure 3.1 : Quipu Machine Menu

Once the connection to the library is established, users are prompted with the library introductory screen. The screen is shown in figure 3.2. Users are instructed to press a *carriage return* to start the dialogue with the library.

```
067 ASTON UNIVERSITY  - GEAC LIBRARY SYSTEM -   ALL *INTRODUCTION


          ++++++++++++++++++++++++++++++++++++++++++++++
          + ++++++++++++++++++++++++++++++++++++++++++++++
          ++                                          ++
          ++  WELCOME TO ASTON UNIVERSITY LIBRARY     ++
          ++     & INFORMATION SERVICES               ++
          ++                                          ++
          ++    THIS IS THE GEAC 9000 SERIES          ++
          ++     ONLINE COMPUTER SYSTEM               ++
          ++                                          ++
          + ++++++++++++++++++++++++++++++++++++++++++++++
          ++++++++++++++++++++++++++++++++++++++++++++++

Now press CARRIAGE RETURN to begin
```

Figure 3.2 : Library Introductory Screen

User interaction to the library is through a hierarchical menu system. The interaction is too modal. This is due to the depth of the menu hierarchy. Users have to interact with several menus before they could successfully find their books or journals. Furthermore, some menus do not provide the required information to navigate through the menu system. A partial menu system of the library menu is shown in figure 3.3.

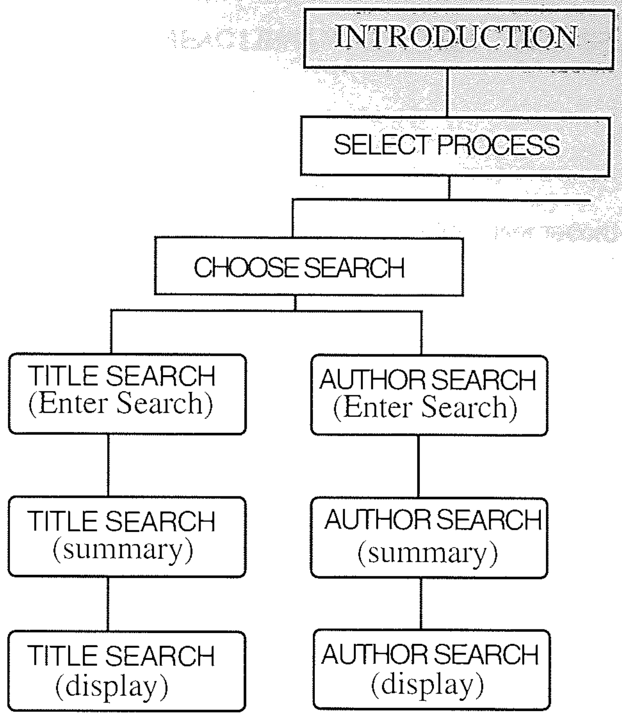
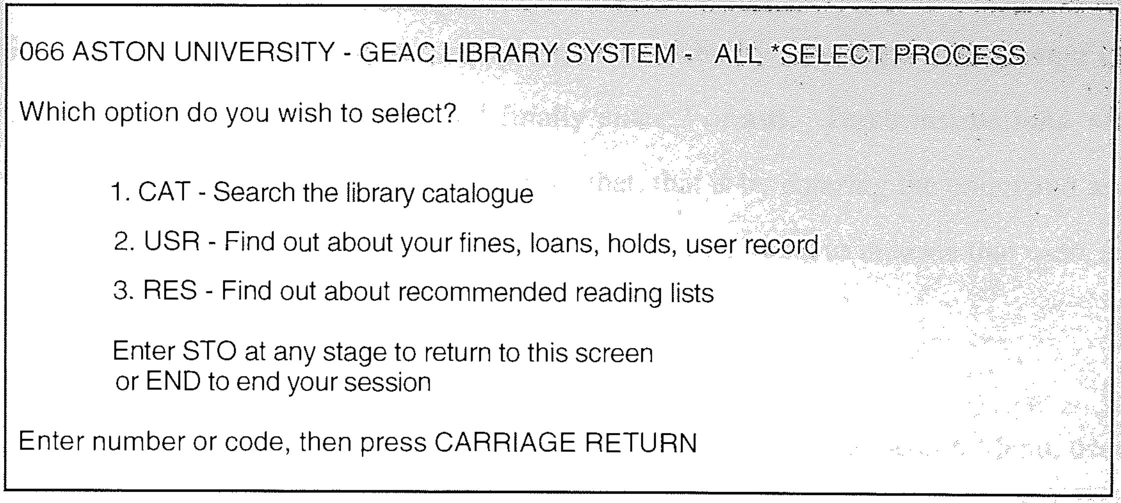


Figure 3.3 : Partial Menu System of Geac

The menus for Select Process, Choose Search and Title Search are shown in figure 3.4. Select Process is the main menu and to return to this menu at any stage can be made by entering **sto** as instructed on the screen. Users may also terminate the session at any stage by entering **end**. Users who choose the first choice, i.e. **cat** may not see the last, but an important instruction at the bottom of the menu, that is the instruction to use **sto** to return to the main menu and **end** to end the session at any stage of the dialogue. These instructions may not be available at different stages of the dialogue.

Assuming that users wish to search for books or journals by title from the Select Prosess menu. First of all, users have to enter **1** or **cat** to get the Choose Search menu and then enter **1** or **til** to get the Title Search menu. The Title Search menu allows users to input the title that they wish to search. This means that users have to go two stages down the hierarchy of menus before they can input the title.



066 ASTON UNIVERSITY - GEAC LIBRARY SYSTEM - ALL *SELECT PROCESS

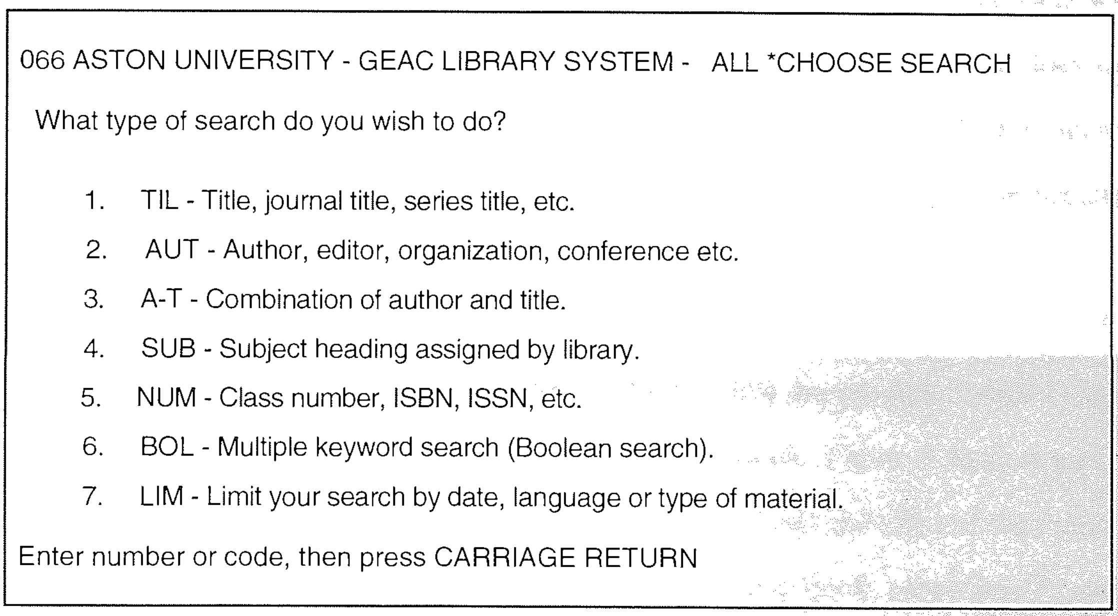
Which option do you wish to select?

1. CAT - Search the library catalogue
2. USR - Find out about your fines, loans, holds, user record
3. RES - Find out about recommended reading lists

Enter STO at any stage to return to this screen
or END to end your session

Enter number or code, then press CARRIAGE RETURN

a) Select Process Menu



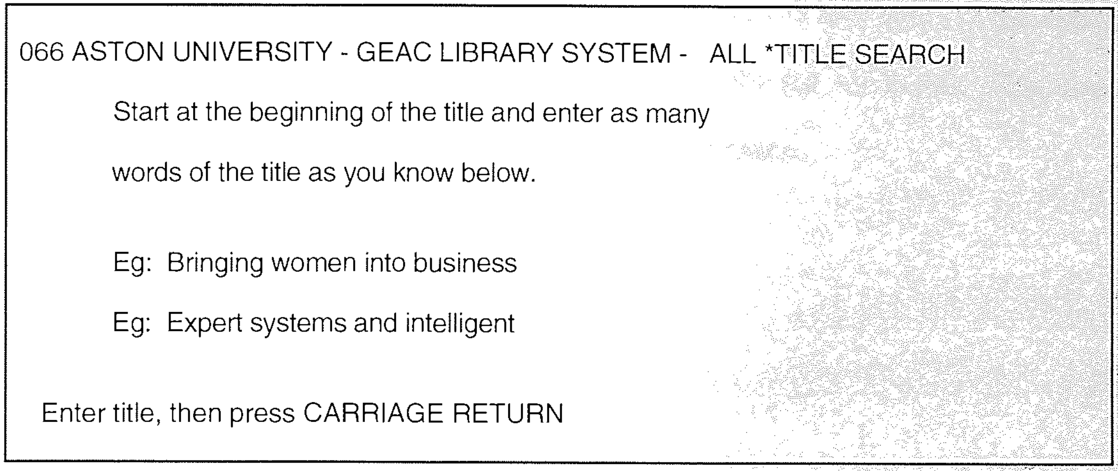
066 ASTON UNIVERSITY - GEAC LIBRARY SYSTEM - ALL *CHOOSE SEARCH

What type of search do you wish to do?

1. TIL - Title, journal title, series title, etc.
2. AUT - Author, editor, organization, conference etc.
3. A-T - Combination of author and title.
4. SUB - Subject heading assigned by library.
5. NUM - Class number, ISBN, ISSN, etc.
6. BOL - Multiple keyword search (Boolean search).
7. LIM - Limit your search by date, language or type of material.

Enter number or code, then press CARRIAGE RETURN

b) Choose Search Menu



066 ASTON UNIVERSITY - GEAC LIBRARY SYSTEM - ALL *TITLE SEARCH

Start at the beginning of the title and enter as many

words of the title as you know below.

Eg: Bringing women into business

Eg: Expert systems and intelligent

Enter title, then press CARRIAGE RETURN

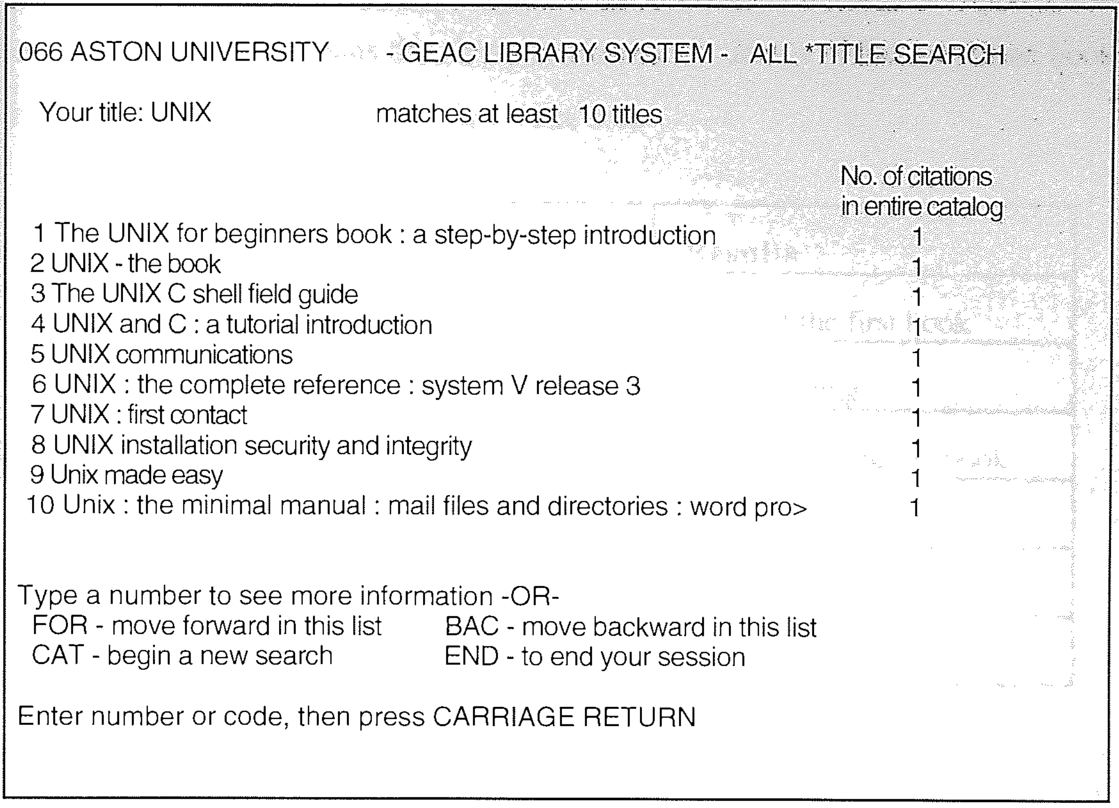
c) Title Search Menu

Figure 3.4 : Different Menus in the Library Menu System

If users are at the Title Search Menu and they wish to switch to the Author Search Menu, the most obvious way is to return to main menu by entering **sto**, then enter **cat** to get the Choose Search Menu and finally enter **2** or **aut**. Three instructions, **sto, cat, aut** are required. There is a shortcut for that, that is by entering the command **aut**. Unfortunately, there is no information on the Title Search Menu to indicate that users are allowed to enter **sto** or **aut** command.

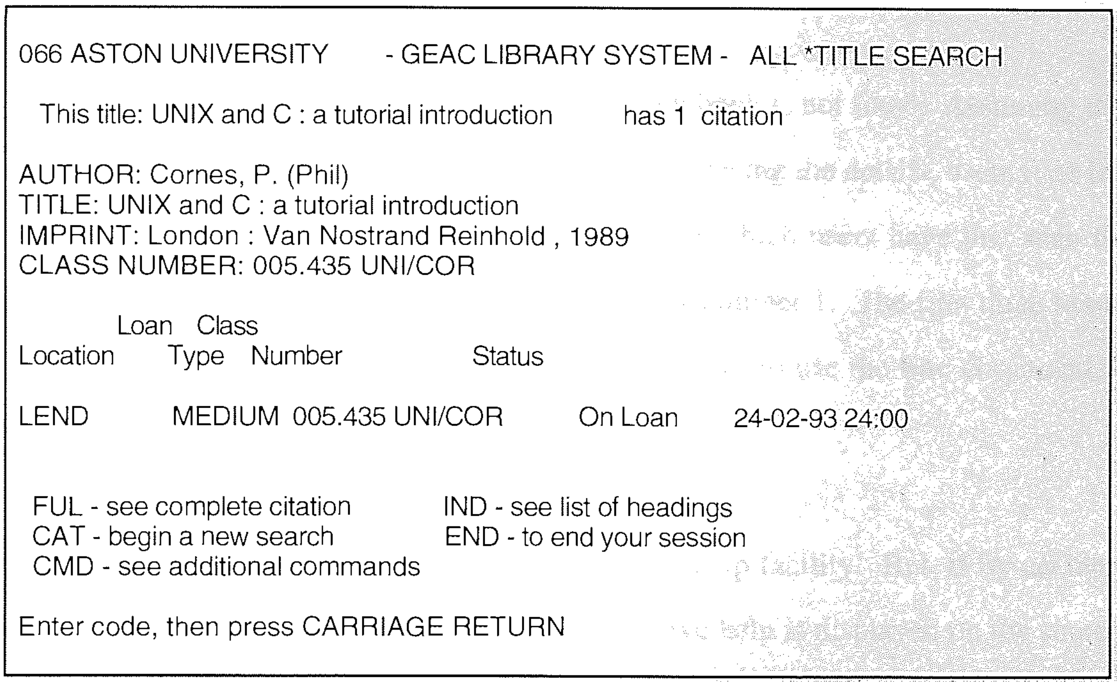
There is another flaw in the design of the dialogue. From the Title Search Menu, users may input *dog* and the word is taken as an input by the dialogue and a search for the word dog is made. But if users enter *cat, cat* is considered a command and users will be prompted with the Choose Search Menu. This means that the dialogue does not allow for a search of the word *cat.* Similarly, if users input *lam* from the Author Search Menu, a few successful matches are made, but users simply cannot input *lim* because *lim* is regarded as command.

When users enter their input at the Title Search Menu, they are prompted with a menu that displays a list of headings which are related to the search. Figure 3.5a shows a list of headings when a title search for the word *unix* is made. Only the title and the number of citations are listed on the screen. To see the detail of any book, users have to enter the book number. Figure 3.5b shows the details of the book number 4. It contains detailed information about the book such as author, publisher and status in the library. Users can see a complete citation of the book by using the **ful** command. This means that users have to step further down the hierarchy of the menu.



```
066 ASTON UNIVERSITY     - GEAC LIBRARY SYSTEM -   ALL *TITLE SEARCH

 Your title: UNIX                 matches at least  10 titles

                                                         No. of citations
                                                         in entire catalog
 1 The UNIX for beginners book : a step-by-step introduction      1
 2 UNIX - the book                                                1
 3 The UNIX C shell field guide                                   1
 4 UNIX and C : a tutorial introduction                           1
 5 UNIX communications                                            1
 6 UNIX : the complete reference : system V release 3             1
 7 UNIX : first contact                                           1
 8 UNIX installation security and integrity                       1
 9 Unix made easy                                                 1
10 Unix : the minimal manual : mail files and directories : word pro>   1


Type a number to see more information -OR-
 FOR - move forward in this list       BAC - move backward in this list
 CAT - begin a new search              END - to end your session

Enter number or code, then press CARRIAGE RETURN
```

3.5 a) Title Search - A List of Headings



```
066 ASTON UNIVERSITY      - GEAC LIBRARY SYSTEM -   ALL *TITLE SEARCH

 This title: UNIX and C : a tutorial introduction       has 1  citation

AUTHOR: Cornes, P. (Phil)
TITLE: UNIX and C : a tutorial introduction
IMPRINT: London : Van Nostrand Reinhold , 1989
CLASS NUMBER: 005.435 UNI/COR

          Loan   Class
Location     Type   Number             Status

LEND         MEDIUM  005.435 UNI/COR        On Loan      24-02-93 24:00


 FUL - see complete citation          IND - see list of headings
 CAT - begin a new search             END - to end your session
 CMD - see additional commands

Enter code, then press CARRIAGE RETURN
```

3.5 b) Title Search - Detailed Information of the Selected Book

After seeing the details of the book, users have to type **ind** if they wish to review the list of headings. This means that, if users wish to see details of the first three books, they have to follow the following steps (table 3.1).

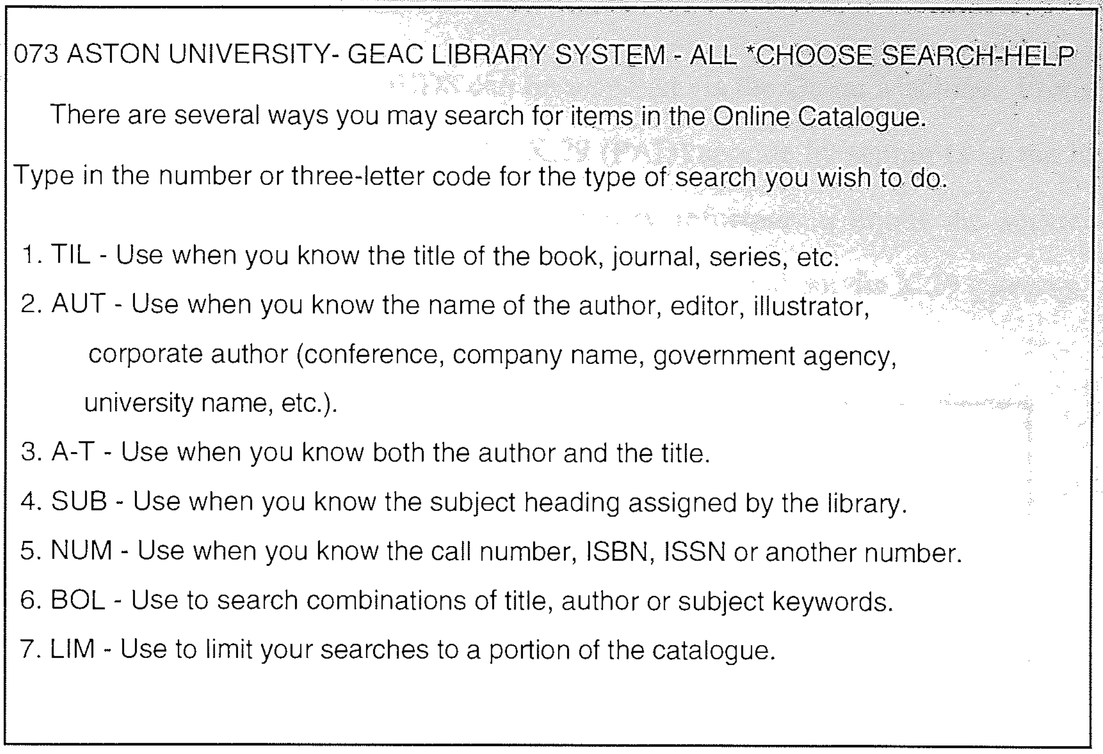
	Steps	**Results**
1	Type in the number of the first book	Details of the first book
2	Type **ind**	List of heading
3	Type in the number of second book	Details of the second book
4	Type **ind**	List of heading
5	Type in the number of third book	Details of the third book
6	Type **ind**	List of heading

Table 3.1 : Steps to view Three Books

Users have to keep on moving up and down in the menus by typing the book number and **ind** command alternately.

Another problem is, the number associated with each book is not fixed. Assuming that users wish to see the details of book number 4. After seeing the details, users type **ind** to return to the list of heading. The book number 4 which users have just seen the details is placed on top of the list and becomes book number 1. The first three books previously listed disappear from the screen. Users have to use the **bac** command, if they wish to review those books again.

There is no clue at all to indicate the availability of help facility. But, if by accident users enter a *carriage return* twice, a context sensitive help is displayed on the screen. Figure 3.6 shows the help screen when users invoke it from the Choose Search Menu. It provides a further explanation about the commands provided by the current menu.


073 ASTON UNIVERSITY- GEAC LIBRARY SYSTEM - ALL *CHOOSE SEARCH-HELP

There are several ways you may search for items in the Online Catalogue.

Type in the number or three-letter code for the type of search you wish to do.

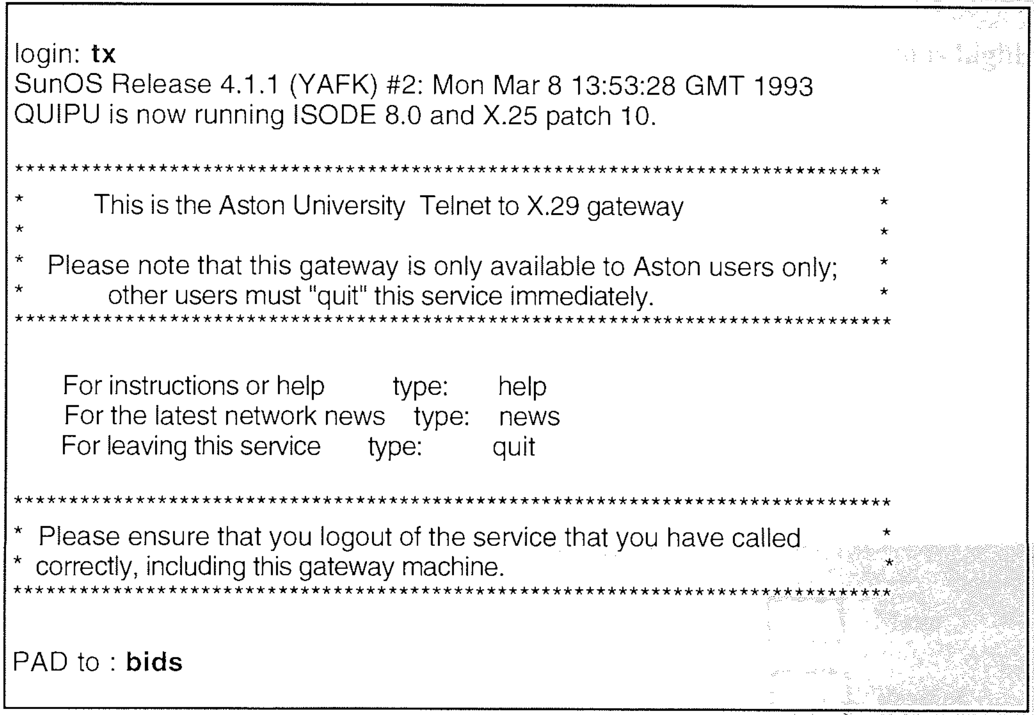
1. TIL - Use when you know the title of the book, journal, series, etc.
2. AUT - Use when you know the name of the author, editor, illustrator, corporate author (conference, company name, government agency, university name, etc.).
3. A-T - Use when you know both the author and the title.
4. SUB - Use when you know the subject heading assigned by the library.
5. NUM - Use when you know the call number, ISBN, ISSN or another number.
6. BOL - Use to search combinations of title, author or subject keywords.
7. LIM - Use to limit your searches to a portion of the catalogue.

Figure 3.6 : Help Screen for Choose Search Menu

Other libraries in the United Kingdom can be accessed via NISS. A review of some of these libraries reveal other flaws in the design of user interface. One of the university libraries does not provide a way to quit the system. The system could be designed for a specialised hardware that accepts quit from its keyboard. The problem arises when the system is accessed using different machines that do not support a similar hardware function.

3.3 BATH INFORMATION DATA SERVICES - BIDS

Similar to the library service, BIDS can be accessed via the Quipu Machine. From the machine, users must first log on to the X.29 (PAD) service by typing **tx** at the login prompt. When users enter **tx**, an introductory information about the service is displayed on the screen. Figure 3.7 shows the information about the X.29 gateway.



```
login: tx
SunOS Release 4.1.1 (YAFK) #2: Mon Mar 8 13:53:28 GMT 1993
QUIPU is now running ISODE 8.0 and X.25 patch 10.

******************************************************************************
*       This is the Aston University  Telnet to X.29 gateway                 *
*                                                                            *
*  Please note that this gateway is only available to Aston users only;      *
*          other users must "quit" this service immediately.                 *
******************************************************************************

      For instructions or help          type:     help
      For the latest network news    type:     news
      For leaving this service          type:      quit

******************************************************************************
* Please ensure that you logout of the service that you have called          *
* correctly, including this gateway machine.                                 *
******************************************************************************

PAD to : bids
```

Figure 3.7 : PAD Introductory Screen

The Quipu Machine has to access BIDS through the X.25 public network. The PAD (Packet Assembly/Disassembly) is needed to serve as an interface between the users machine and the network. The PAD accepts characters from the user machine and sends them across the network; it accepts packets from the network, filters the headers and displays the information on the user machine.

Users have to type **bids** at the 'PAD to:' prompt to invoke a connection to BIDS. BIDS will then request users to enter a valid identification and password before they can successfully access BIDS. This means that, from a terminal, users must know how to access the Quipu Machine and PAD system and then recall ID/password every time they wish to use BIDS.

The BIDS service provides a combination of prompts and well structured hierarchical menu-based interface. Users have to interact with the service by selecting an appropriate choice from the presented menus and the system responds by displaying another menu or a prompt. Figure 3.8 shows a sequence of menus and prompts before users can successfully retrieve their searched articles. Since the system is highly modal, it takes some time for novices to learn the system.

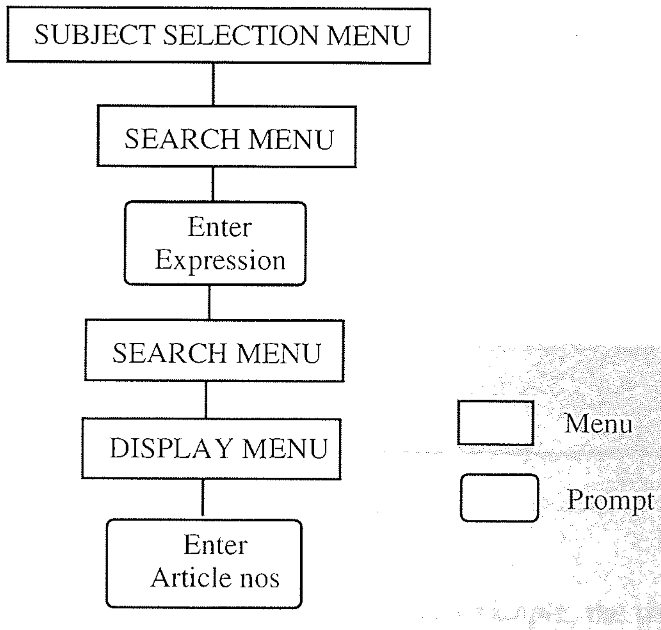
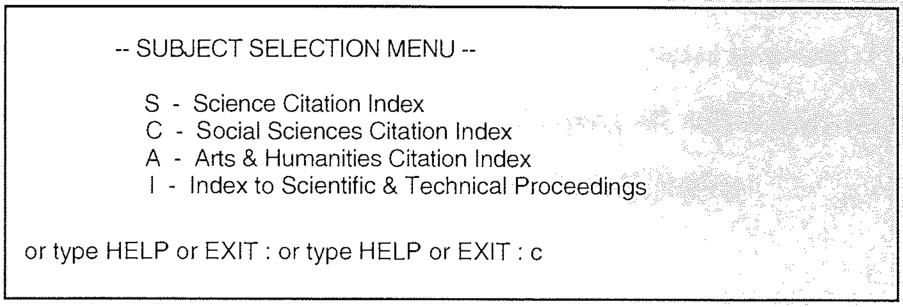


Figure 3.8 : Hierarchy of BIDS Menus

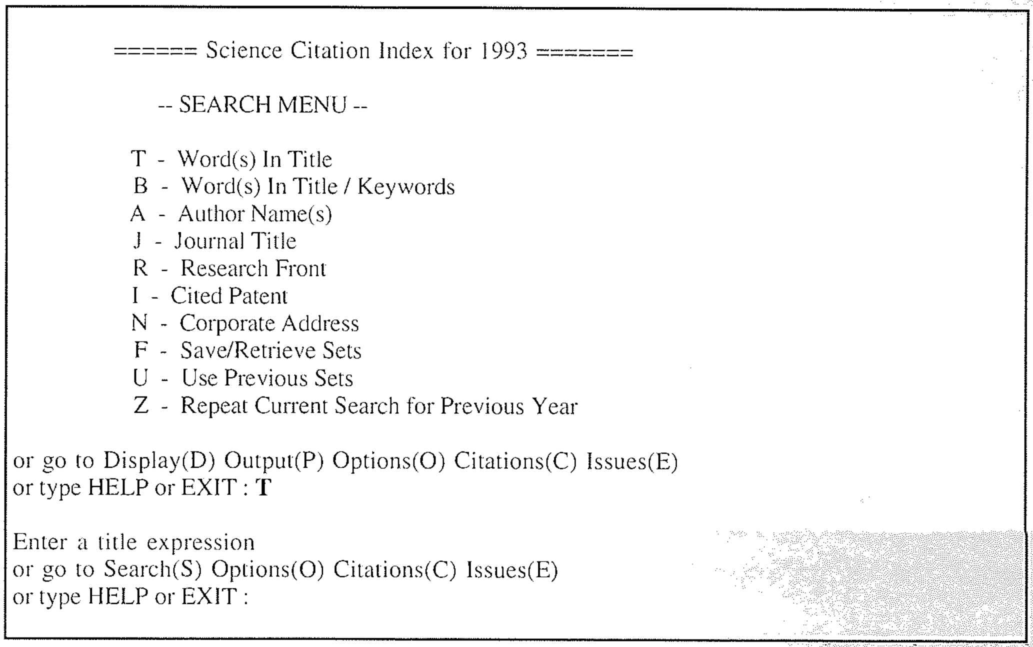
Once logged in, users are presented with some introductory screens, followed by the Subject Selection Menu as shown in figure 3.9.



```
            -- SUBJECT SELECTION MENU --

                S  -  Science Citation Index
                C  -  Social Sciences Citation Index
                A  -  Arts & Humanities Citation Index
                I  -  Index to Scientific & Technical Proceedings

or type HELP or EXIT : or type HELP or EXIT : c
```

Figure 3.9 : Subject Selection Menu

A subject of interest can be selected by typing the single letter next to the selection. Having selected a subject of interest, users are presented with the Search Menu. The Search Menu contains a choice of fields to search on. Figure 3.10 shows the Search Menu. The user is currently connected to the 1993 Science Citation Index and the system prompts the user to input a title expression when **t** (a search by title) is selected from the menu.



```
           ====== Science Citation Index for 1993 =======

                -- SEARCH MENU --

              T  -  Word(s) In Title
              B  -  Word(s) In Title / Keywords
              A  -  Author Name(s)
              J  -  Journal Title
              R  -  Research Front
              I  -  Cited Patent
              N  -  Corporate Address
              F  -  Save/Retrieve Sets
              U  -  Use Previous Sets
              Z  -  Repeat Current Search for Previous Year

or go to Display(D) Output(P) Options(O) Citations(C) Issues(E)
or type HELP or EXIT : T

Enter a title expression
or go to Search(S) Options(O) Citations(C) Issues(E)
or type HELP or EXIT :
```

Figure 3.10 : Search Menu

Figure 3.11 shows an example of a search session. In the example, the title/keywords search is chosen. The input for the search is *user interface* and 15 articles match. The articles are not displayed immediately. Users have to enter **d** (display) if they wish to see the details of the articles. The system will then display the Display Menu where users can choose one of the display formats. In the given example, only the first article is selected for display. Users may also view several articles at a time. For example, typing **f10** will display the first 10 articles and typing **all** will display all the articles. Several different input formats are available from the help facility.

```
=================================================================

Enter a title/keywords expression
or go to Search(S) Options(O) Citations(C) Issues(E)
or type HELP or EXIT : USER INTERFACE


      ==== Science Citation Index for 1993 ====

>>>>>>>>>>>>>>>>>>>>>>  Saved as set number 1      <<<<<<<<<<<<<<<<<<<<<<
>>>>>>>>>>>>>>>>>>>>>>  15 hits                    <<<<<<<<<<<<<<<<<<<<<<


            -- SEARCH MENU --

        T  -   Word(s) In Title
        B  -   Word(s) In Title / Keywords
        A  -   Author Name(s)
        J  -   Journal Title
        R  -   Research Front
        I  -   Cited Patent
        N  -   Corporate Address
        F  -   Save/Retrieve Sets
        U  -   Use Previous Sets
        Z  -   Repeat Current Search for Previous Year

or go to Display(D) Output(P) Options(O) Citations(C) Issues(E)
or type HELP or EXIT : D


      ==== Science Citation Index for 1993 ====

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>>  Set 1 : (USER INTERFACE)@(TI,WA,KP)                                      <<
>>  15 hits                                                                  <<
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

            -- DISPLAY MENU --

        T  -   Title only
        J  -   Journal, authors & title
        X  -   Full record (excl. citations & r.fronts)
        I  -   Full record
        B  -   Journal, authors & title in downloading format
        G  -   Full record in downloading format

or go to Search(S) Output(P) Options(O) Citations(C) Issues(E)
or type HELP or EXIT : I


Enter article nos. for display (total 15)
or go to Search(S) Display(D) Output(P) Options(O) Citations(C) Issues(E)
or type HELP or EXIT : 1


 Article 1 (total of 15 in this set)

(1)  TI: MULTILINK - AN INTERMEDIARY SYSTEM FOR MULTI-DATABASE ACCESS
     AU: WU_G, AHLFELDT_H, WIGERTZ_O
     JN: METHODS OF INFORMATION IN MEDICINE 1993 VOL.32 NO.1 PP.82-89

Enter M to mark article, or Q to quit display
or go to Search(S) Display(D) Output(P) Options(O) Citations(C) Issues(E)
or type HELP or EXIT :

=================================================================
```

Figure 3.11 : Search Session

BIDS provides a powerful facility which allows expert users to input more than one command in a single line. The commands have to be separated by semicolons. As an example typing **d;t;f5** after a successful search will display the first five articles in title-only format.

By default users are connected to data for the current year only. Users who wish to search for all articles for the last 5 years have to repeat the current search for the previous year 4 times. Repeating the current search for the previous year can be carried out by selecting **z** from the search menu. Every time users carry out a search for the previous year, the current year will be set to the year before. If users wish to make another search for all articles for the last number of years, they have to reset the current year to 1994 (current year) and keep on repeating the search a number of times. The option menu can be used to change various settings, including switching the current year to different year.

Users may view all articles for a particular journal by selecting **j** from the search menu. For the year 1992, if users input *ieee software*, after selecting **j**, 161 articles are found. But if users input *computer*, more than 3,000 articles are found. There is a journal named Computer, but the number of hits for this input is large because the system includes all journals that contain the word computer in their titles. Users who wish to view articles in the Computer Journal have to browse through a list of more than 3,000 articles.

Users may retrieve all articles in the Computer Journal only by specifying at least the volume of the journal. If users input *computer + v25*, all articles in the Computer Journal volume 25 will be selected. This means that users must know the journal and its volume number before they carry out the search. The search can be more selective by specifying the volume and the number of the journal. If users input *computer + v25 + N2*, all articles in the Computer journal volume 25 number 2 will be selected.

Users can use the **F** option in the search menu to save sets of searches they have carried out. Users will be given a unique identifier for their saved set along with the following notices:

a) make a note of the identifier for future use.

b) the set will be deleted if not accessed within 3 months.

c) the sets may only be used from within the current username.

The given unique identifiers are usually codes which are difficult for users to recall when they wish to retrieve the saved searches. Thus, it is easier for users to redo the search rather than keeping note of the identifier. They have to search for the note whenever they wish to retrieve the saved search. An alternative is to copy the searches in their own files and assign an appropriate name to them.

Users may obtain one or more screens of help by typing **help** at any prompt. The **help** screens contain the most recent enhancement to the system.

3.4 USENET NEWS

Two USENET news reader are reviewed. They are Tin and Xvnews. Tin is a menu user interface and Xvnews is a graphical user interface.

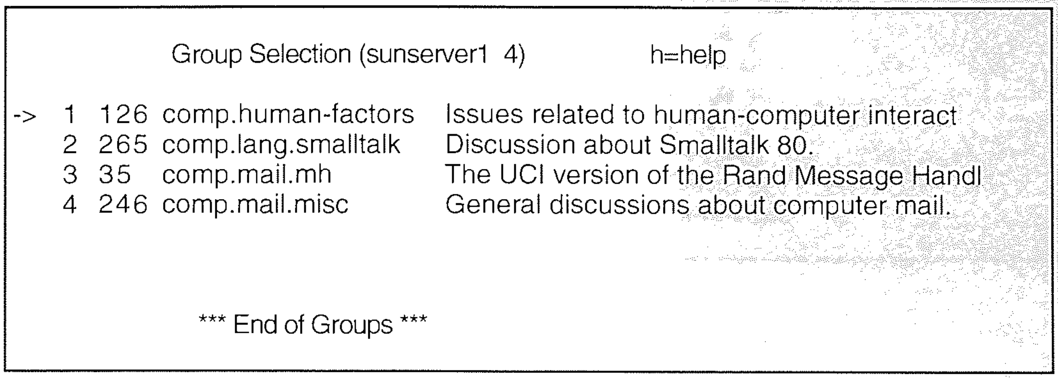
3.4.1 TIN

Tin is a full-screen user interface that helps users to read USENET news. To invoke Tin, users have to log-on to the Uhura Machine and then type **tin** at the Unix prompt. The following prompts are displayed on the screen when tin is invoked.

```
% tin
tin 1.1 PL7.uhura.1 (c) Copyright 1991-92 Iain Lea.
Connecting to sunserver1...
Reading news active file...
Reading newsgroups file...
```

The messages which are appended by the '...' are feedback given by the system to indicate different activities involved before information can be fully accessed. The feedback is essential because it informs the users that their request is currently being processed.

Figure 3.12 is the Tin display when it first starts up. It is at the **group selection level** and it shows all the three newsgroups the user subscribes to.



```
              Group Selection (sunserver1  4)          h=help

->  1  126  comp.human-factors    Issues related to human-computer interact
    2  265  comp.lang.smalltalk   Discussion about Smalltalk 80.
    3  35   comp.mail.mh          The UCI version of the Rand Message Handl
    4  246  comp.mail.misc        General discussions about computer mail.


               *** End of Groups ***
```

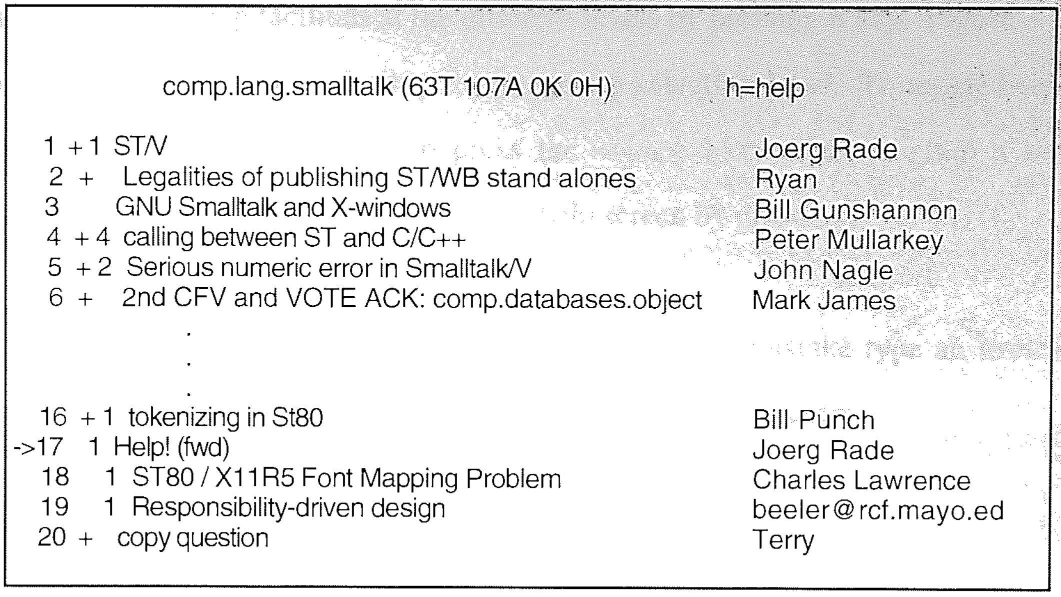
Figure 3.12 : Subscribed Newsgroups

Users may also obtain the main list, that is the list of all newsgroups by pressing letter **y** and return to the subscribed list by pressing letter **Y**. It is important to note that users are not required to enter a *carriage return* after entering a command. Figure 3.13 shows the first page of the main list. There are currently 2613 newsgroups and users may subscribe to any of them. Letter **u** shows that the newsgroup is not subscribed by users.

The arrow on the screen can be moved up and down by using up and down arrow keys respectively. The arrow indicates that the pointed newsgroup is the selected newsgroup. Pressing a *carriage return* will bring the user to the selected newsgroup level. Since the arrow is pointing at the **comp.lang.smalltalk** newsgroup, pressing a *carriage return* causes a display of the details of the newsgroup. Figure 3.14 shows the **comp.lang.smalltalk** newsgroup and a list of articles posted to the newsgroup.

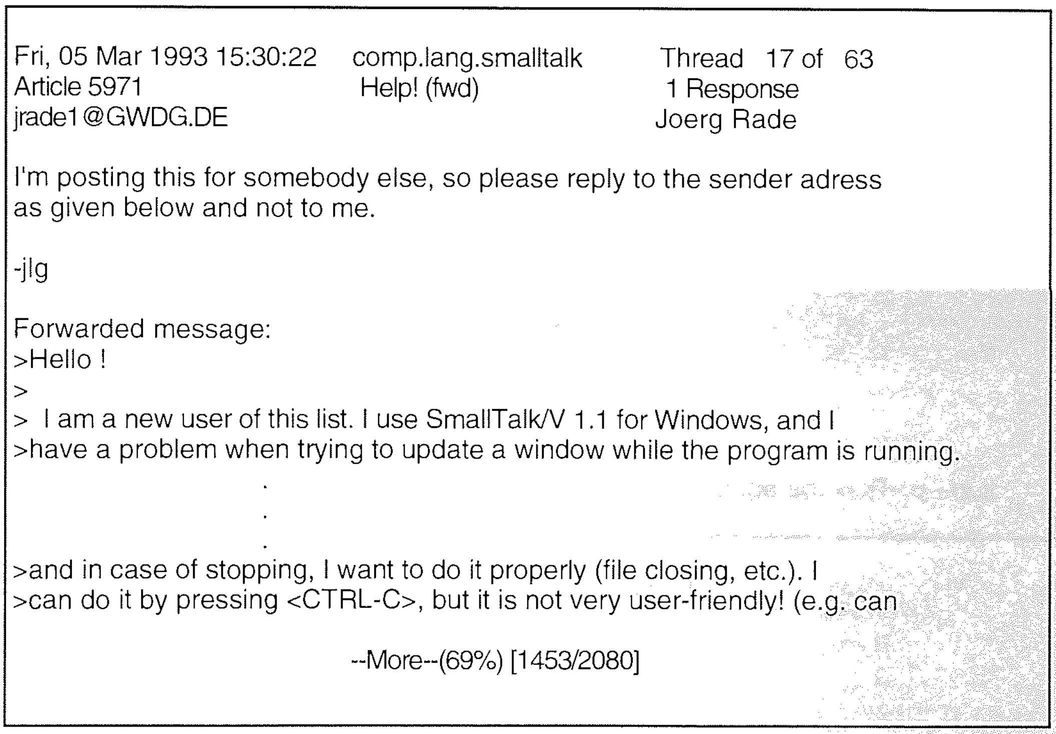
```
     Group Selection (sunserver1  2613)            h=help

   u  861 23639      comp.lang.pascal              Discussi
   u  862 17779      comp.lang.perl                Discussi
   u  863   826      comp.lang.pop                 Pop11 an
   u  864 11806      comp.lang.postscript          The Post
   u  865  5059      comp.lang.prolog              Discussi
   u  866  4216      comp.lang.rexx                The REXX
   u  867  4728      comp.lang.scheme              The Sche
   u  868   168      comp.lang.scheme.c            The Sche
   u  869    64      comp.lang.sigplan             Info & a
  ->  870   274      comp.lang.smalltalk           Discussi
   u  871 12876      comp.lang.tcl                 The Tcl
   u  872  1576      comp.lang.verilog             Discussi
   u  873  2361      comp.lang.vhdl                VHSIC Ha
   u  874  1725      comp.lang.visual              Visual p
   u  875     3      comp.laser-printers           Laser pr
   u  876  1936      comp.lsi                      Large sc
   u  877  2607      comp.lsi.cad                  Electric
   u  878  1059      comp.lsi.testing              Testing
   u  879  3347      comp.mail.elm                 Discussi
   u  880   379      comp.mail.headers             Gatewaye
```

Figure 3.13 : Main List



```
              comp.lang.smalltalk (63T 107A 0K 0H)          h=help

  1  +1  ST/V                                                Joerg Rade
  2  +   Legalities of publishing ST/WB stand alones         Ryan
  3      GNU Smalltalk and X-windows                         Bill Gunshannon
  4  +4  calling between ST and C/C++                        Peter Mullarkey
  5  +2  Serious numeric error in Smalltalk/V                John Nagle
  6  +   2nd CFV and VOTE ACK: comp.databases.object         Mark James
                    .
                    .
                    .
 16  +1  tokenizing in St80                                  Bill Punch
->17  1  Help! (fwd)                                          Joerg Rade
 18   1  ST80 / X11R5 Font Mapping Problem                   Charles Lawrence
 19   1  Responsibility-driven design                        beeler@rcf.mayo.ed
 20  +   copy question                                       Terry
```

Figure 3.14 : Comp.Lang.Smalltalk Newsgroup



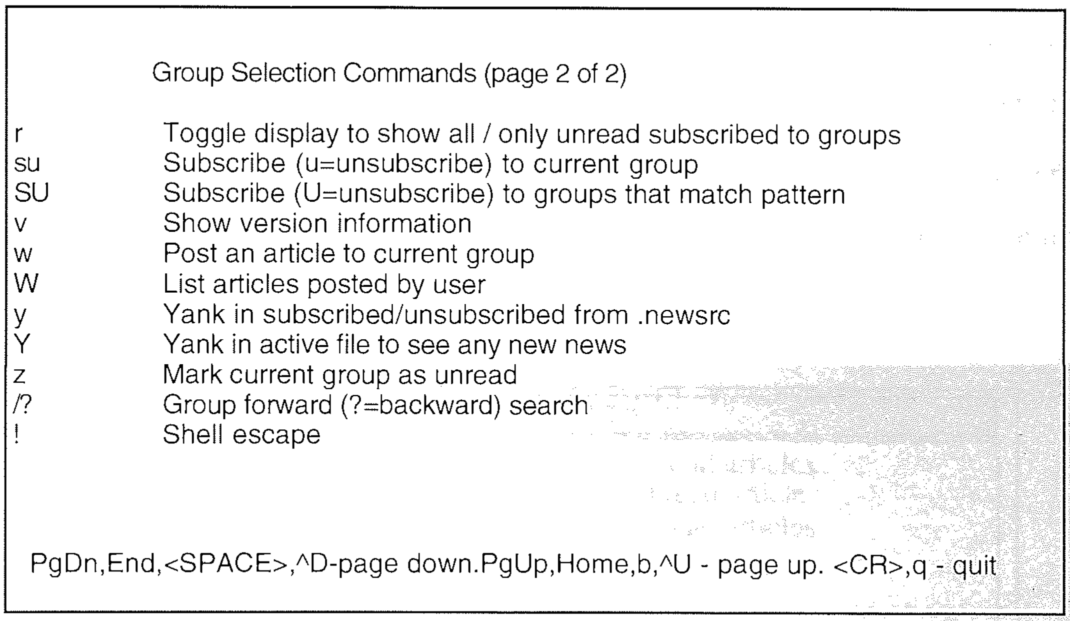
```
Fri, 05 Mar 1993 15:30:22   comp.lang.smalltalk     Thread   17 of   63
Article 5971                 Help! (fwd)             1 Response
jrade1@GWDG.DE                                      Joerg Rade

I'm posting this for somebody else, so please reply to the sender adress
as given below and not to me.

-jlg

Forwarded message:
>Hello !
>
>  I am a new user of this list. I use SmallTalk/V 1.1 for Windows, and I
>have a problem when trying to update a window while the program is running.
                    .
                    .
                    .
>and in case of stopping, I want to do it properly (file closing, etc.). I
>can do it by pressing <CTRL-C>, but it is not very user-friendly! (e.g. can

                         --More--(69%) [1453/2080]
```

Figure 3.15 : Display of the Selected Article

Similar to the previous screen, an arrow is used to indicate the selected item. Pressing a *carriage return* causes a display of the selected article. The selected article is displayed in figure 3.15. It contains the details such as sender's address, date sent, subject and body of the article.

Users can obtain help facilities at the different levels by pressing **h** key. Figure 3.16 is the second page of the help facility for the **group selection level**. To toggle between page one and two, users have to press the <space bar>. They contain a list of commands for the level. Users can quit the help screen by pressing **q**.

Tin does have a way of trapping user error. If users by mistake type an irrelevant command, the system may terminate and return a core dump to users.



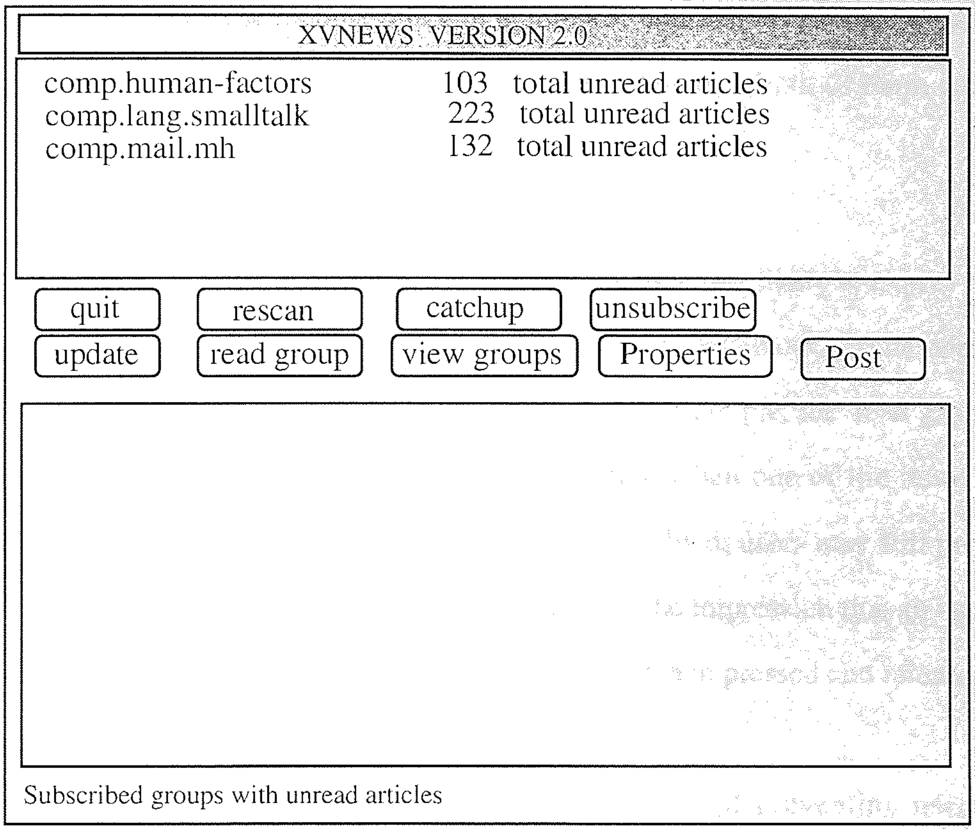
```
                 Group Selection Commands (page 2 of 2)

r                Toggle display to show all / only unread subscribed to groups
su               Subscribe (u=unsubscribe) to current group
SU               Subscribe (U=unsubscribe) to groups that match pattern
v                Show version information
w                Post an article to current group
W                List articles posted by user
y                Yank in subscribed/unsubscribed from .newsrc
Y                Yank in active file to see any new news
z                Mark current group as unread
/?               Group forward (?=backward) search
!                Shell escape


PgDn,End,<SPACE>,^D-page down.PgUp,Home,b,^U - page up. <CR>,q - quit
```

Figure 3.16 : Help Screen

3.4.2 XVNEWS

The Xvnews is a USENET news reader which is based on a graphical user interface. User interaction to the system is mainly by invoking the buttons or selecting items in the scroll list view. Each button represents a command to the system and a scroll list contains either a list of newsgroups or a list of articles. All of them are clearly displayed inside a window.

Figure 3.17 shows the arrangement of buttons and views in one of the Xvnews Windows. The scroll list view at the top of the window holds the list of subscribed newsgroups. It is not appropriate to label the window according to the name of the application especially when the application contains many windows. The content of the list scroll list is printed at the bottom of the window. Such important piece of information should be made as the label of the window because it describes the main object the window is holding.


```
XVNEWS VERSION 2.0
comp.human-factors        103  total unread articles
comp.lang.smalltalk       223  total unread articles
comp.mail.mh              132  total unread articles
```

Subscribed groups with unread articles

Figure 3.17: Subscribed Groups

Users have to wait for quite some time before the first window and all the information inside it appears on the screen when users first invoke Xvnews. It is understood that it takes some time for the system to read the USENET news. The system does not provide any feedback at all about what is going on. Users are generally suspicious when they do not find any immediate result to their request. Thinking that their request is not accepted by the system, some users might invoke the reader again.

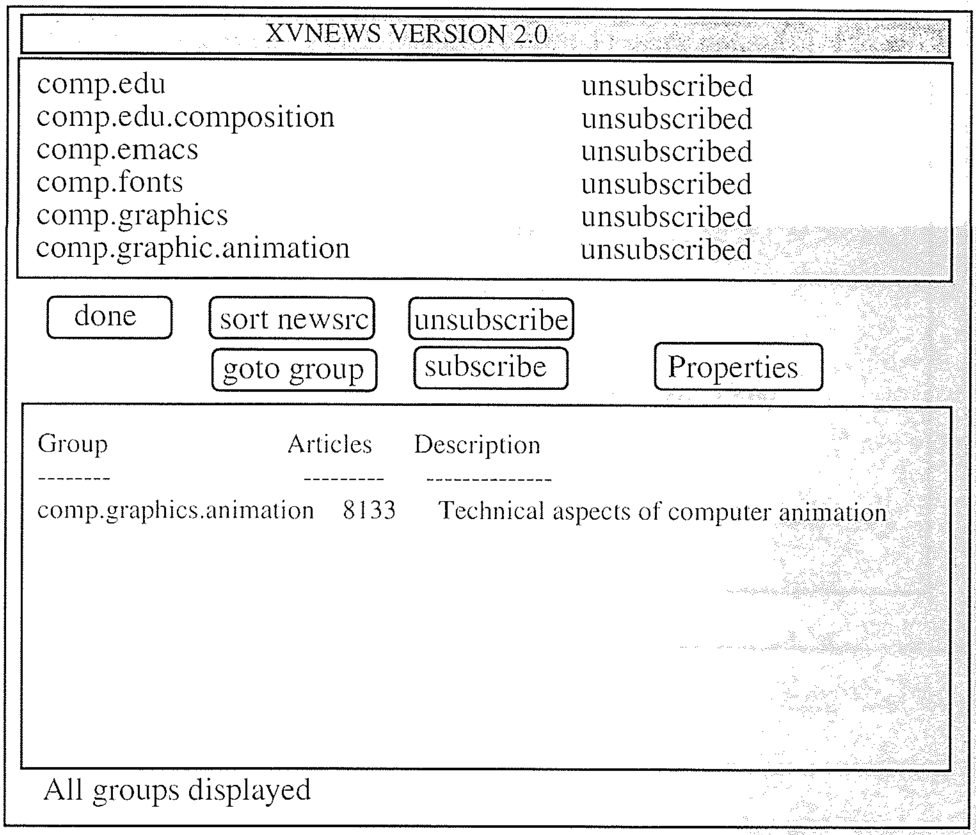
Rather than keeping users waiting without knowing what is happening, the system could at least prompt a simple message indicating that users request is being processed. A better approach is to open a status window and inform users about the progress of the processing.

The naming of many buttons is confusing and there is no help facility to explain the function of each of the buttons. A name such as 'catchup' does not really describe its function. The 'read group' and 'view groups' names almost carry the same meaning. There is also a problem of inconsistency in naming of buttons. In another window the name 'goto group' is used instead of 'read group' even though both of them carry out the same function.

In a modern graphical user interface based application, users are likely to expect buttons that represent currently invalid operation to be disabled or greyed out. Some operations are valid only when certain conditions are fulfilled. For example, the 'read group' and 'unsubscribe' buttons on the window are only effective when one of the items in the scroll list view is selected. Since the buttons are not disabled, users may still press any of these buttons without doing any selection. Users get the impression that an operation is taking place when they see the buttons change colour when pressed and released.

Disabling of buttons to show invalid operation is a way of preventing users from making mistakes. An alternative is to prompt users with an instruction to redo the operation by selecting any item in the scroll list before pressing those buttons. Neither disabling of button nor providing informative feedback technique is implemented by the Xvnews user interface.

Figure 3.18 will be displayed when users press 'view groups' from the first window (figure 3.17). It contains thousands of newsgroups which can be subscribed to by users by selecting the item from the list and then press 'subscribed'. One of the functions which is important but not provided by Xvnews is a search function. Using a search function, users can easily search for certain groups without browsing through the whole list of groups which can be time consuming.



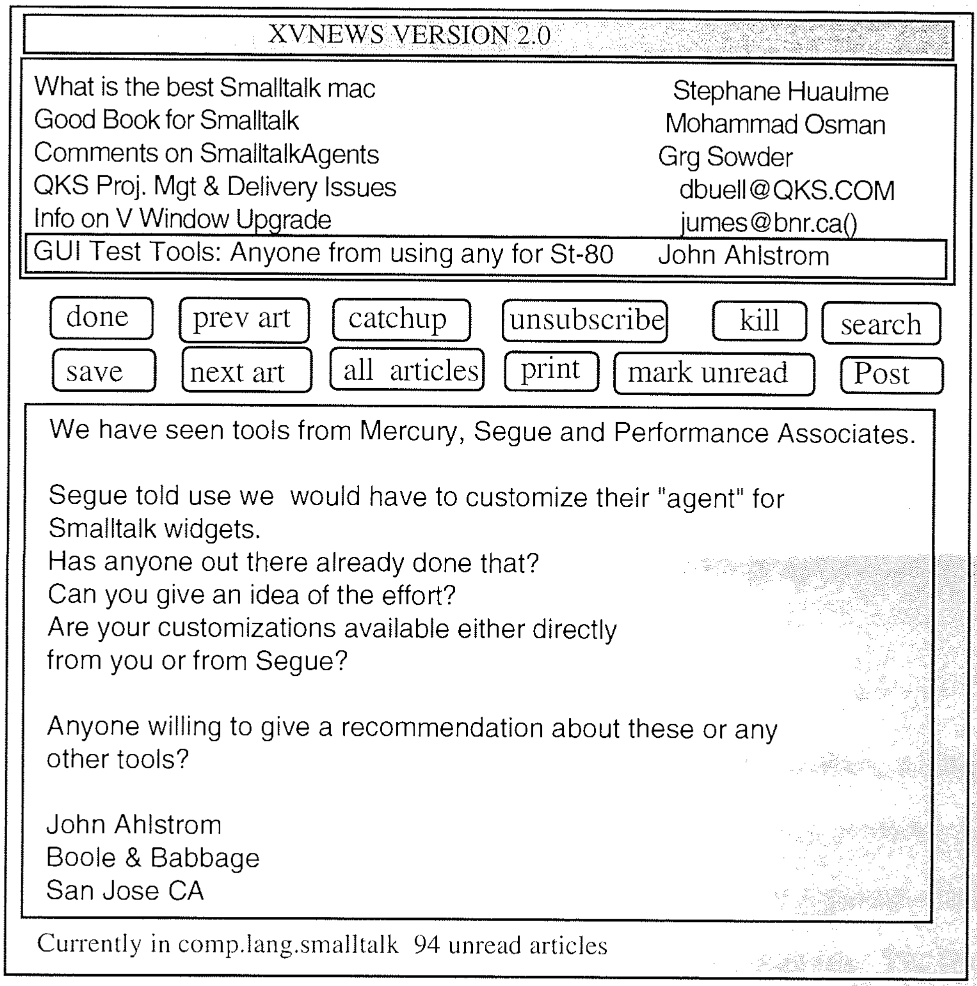
```
comp.edu                          unsubscribed
comp.edu.composition              unsubscribed
comp.emacs                        unsubscribed
comp.fonts                        unsubscribed
comp.graphics                     unsubscribed
comp.graphic.animation            unsubscribed
```

```
Group                    Articles    Description
--------                 ---------   --------------
comp.graphics.animation  8133        Technical aspects of computer animation
```

All groups displayed

Figure 3.18 : List of All Groups

Another window will be displayed when users select a group and invoke 'read group' from the first window or 'goto group' from the second window. The window is shown in figure 3.18 and it holds a list of articles posted to the selected group. The selected article is displayed on the main view below. Again, the title is placed at the bottom of the window.


XVNEWS VERSION 2.0

What is the best Smalltalk mac	Stephane Huaulme
Good Book for Smalltalk	Mohammad Osman
Comments on SmalltalkAgents	Grg Sowder
QKS Proj. Mgt & Delivery Issues	dbuell@QKS.COM
Info on V Window Upgrade	jumes@bnr.ca()
GUI Test Tools: Anyone from using any for St-80	John Ahlstrom

We have seen tools from Mercury, Segue and Performance Associates.

Segue told use we would have to customize their "agent" for
Smalltalk widgets.
Has anyone out there already done that?
Can you give an idea of the effort?
Are your customizations available either directly
from you or from Segue?

Anyone willing to give a recommendation about these or any
other tools?

John Ahlstrom
Boole & Babbage
San Jose CA

Currently in comp.lang.smalltalk 94 unread articles

Figure 3.19 : Articles in a Group

3.5 PP ELECTRONIC MAIL

Most of the email applications are monolithic. To use email, users have to start the mail program, usually by entering its name, such as mail, at a shell prompt. Then the program takes over, and within the program users may execute commands which are related to email. For example, users may read, print and delete email messages. Users are in the mail program until they quit out of it.

Unlike most email applications, users can use PP commands from a UNIX shell prompt. The commands are available at any time and users are not required to start or quit the mail system. In PP, each command is a separate program, and the shell is used as an interpreter. Hence, PP can fully utilise the power of UNIX shells such as pipes, redirection, aliases and so on. For example, users may send the output of a command to the printer directly by using a pipe. In addition, since PP is not monolithic, PP commands can be used in UNIX shell scripts or called from programs in high-level languages like C.

Monolithic mail systems keep all of the message in the system mailbox, a single file which holds all mail messages together. The mail system usually provides limited facilities to manipulate the messages such as splitting them into separate messages. Unlike monolithic mail systems, PP keeps each message in a separate file. The filename is the message number. The messages can be kept in one or more folders, which are actually UNIX directories. The PP setup offers many advantages. A disadvantage is that it takes more space to store the messages.

There are too many features of PP to be discussed in the section. Only a brief review of some operations provided by PP are described.

- Composing and sending a message.

Users may compose an email message by using the command **comp** at a shell prompt. Invoking the command causes it to respond by prompting the message header where users will be asked to enter the recipient address, carbon copy and subject of the message. Users must include at least the recipient address in the message header. After completing the message header, users are prompted with the message body. They may type their message in the message body. Users may quit the message body by entering **ctrl-d** after placing the cursor at the start of a new line. The message body is enclosed by the line "----------"

Users are prompted with What now? by the program. Users have to type **send** if they want to send the message, or **quit** if they want to quit without sending the message. Figure 3.20 shows a session to compose and send a message.

```
$ comp
To: abdullah
cc: research_group
Subject: Meeting - April
--------

Group meeting will be held on 13/4/94, Room MB268
Thank you

(CTRL-D)
--------

What now? send
```

Figure 3.20 : Composing and Sending a Message

- Incorporating new message

Users have to type the **inc** command to incorporate new messages from the system mailbox into their account. The mailbox is the place where all new mail are kept. The incorporated mail will be placed into the default folder, *inbox*.

- Listing of messages

To obtain a listing of messages in the current folder, users have to type the **scan** command. Examples of a listing of messages is shown below.

```
1  05/09 Renu Budhiraja    Binary enclosures<<Hello ! I wanted to add a new
2  04/14 Howard Jeffrey -  Sunnet X.25 error<<I have been getting an error
3  04/27 Giles Christian   earn-relay and letterbox<<A brief outline of the
4  04/28 Tim Goodwin       Re: If not PP, what?<<> Out of interest what wou
5  04/29 Richard Letts     Re: Wanted: local channel & hostname help<<David
6+ 05/02 Susan B. Jones    Re: SigD0c '94<<Your proposed paper has been acc
```

There is a one-line summary of each message. The summary shows the message number, the date when the message was sent, who sent it and the subject of the message, if any, is numbered and the current message is indicated by the '+' after the message number. If there is room, the first few words of the message body are also included.

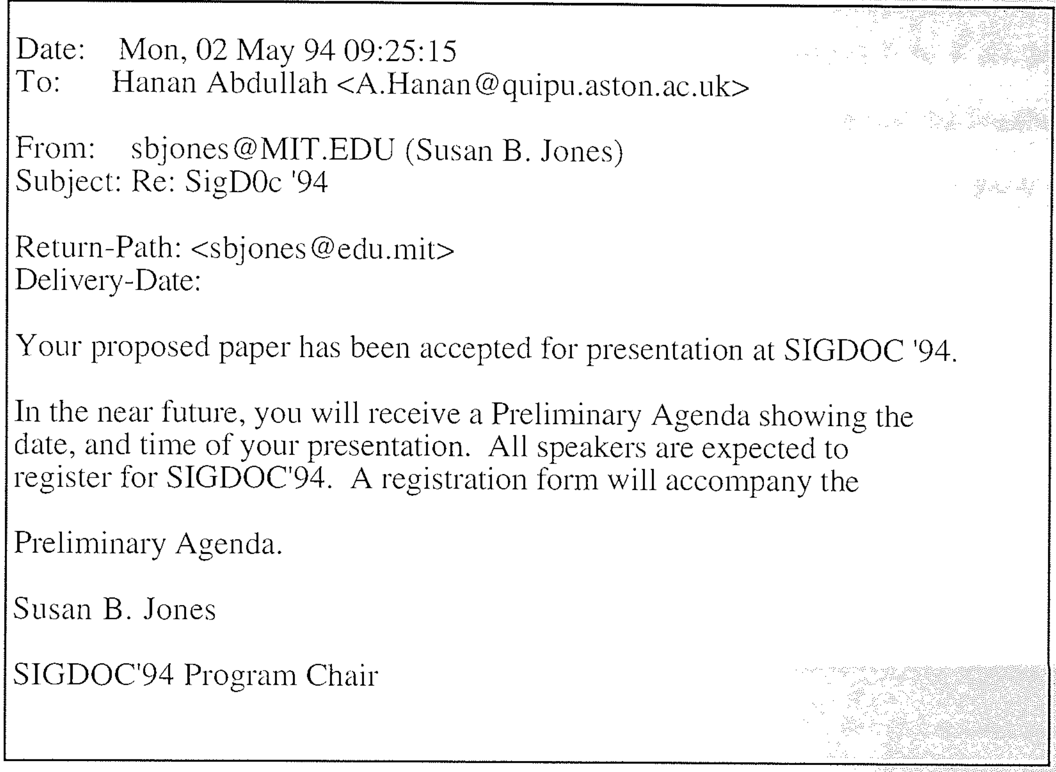
PP provides the facility to format the display of any command. For example, the **spscan** is a formatted **scan** command. The command aligns the subject of the messages and the words from the message body are excluded. The display of **spscan** is as follows:

```
1  05/09  Renu Budhiraja           #Binary enclosures
2  04/14  Howard Jeffrey - Cranf   #Sunnet X.25 error
3  04/27  Giles Christian          #earn-relay and letterbox
4  04/28  Tim Goodwin              #Re: If not PP, what?
5  04/29  Richard Letts            #Re: Wanted: local channel & hostname help
6  05/02  Susan B. Jones           #Re: SigD0c '94
```

Users may specify the range of messages they would like to see. For example, to scan the range of messages number 3 through 5 inclusive, they have to type **scan 3-5.** Users may also specify their scan format.

- Displaying the content of a message

Users may display the content of a message by using the **show** command. The command will display the content of the current message. Figure 3.21 shows the content of a message.



```
Date:    Mon, 02 May 94 09:25:15
To:      Hanan Abdullah <A.Hanan@quipu.aston.ac.uk>

From:    sbjones@MIT.EDU (Susan B. Jones)
Subject: Re: SigD0c '94

Return-Path: <sbjones@edu.mit>
Delivery-Date:

Your proposed paper has been accepted for presentation at SIGDOC '94.

In the near future, you will receive a Preliminary Agenda showing the
date, and time of your presentation.  All speakers are expected to
register for SIGDOC'94.  A registration form will accompany the

Preliminary Agenda.

Susan B. Jones

SIGDOC'94 Program Chair
```

Figure 3.21 : Content of a Message

- Replying to a message

The **repl** command is used to reply to the current message. Users may specify which message to reply to by including the message number, for example, **repl 3**, if they wish to reply to message 3. The command prompts with the message header filled-up by the system. Users have to type the message and then follow the same procedures as for a new message.

- Refiling messages

The **refile** command is used to move messages from one folder to another. The command **refile +myfolder** will move the current message to a file named myfolder. Users may check whether the message has been moved to the specified folder by using the command **scan +myfolder.** The list of all messages in the folder *myfolder* is displayed.

- Removing messages

The command for removing messages is **rmm.** The command **rmm 6 8** would remove messages 6 and 8 and **rmm 6-8** would remove messages 6 to 8 inclusive. After removing and refiling messages, there can be gaps and disorder in the numbering of messages. To renumber the messages, i.e., so that they are in the right sequence the command **folder -pack** may be used.

- Finding messages

The **pick** command can be used to search for messages. The command usually contains a switch, which allows users to specify the field of the search. The following are examples of the use of the **pick** command.

Commands	Explanations
a) pick -search email	find messages that contain the word email anywhere in the message
b) pick -from joed	find messages sent by joed
c) pick -after -7	find messages sent the last 7 days

The pick command returns the message numbers of the messages that match the search. The UNIX backquote can be used to collect the list of messages and pass it to other command. For example, to obtain a listing of message sent by joed, the **scan ` pick -from joed`** command may be executed.

- Folders and Folder operations

By default, all users' messages are kept in a single folder called *inbox*. PP also provides operations which are relevant to folders.

The command **folder** can be used to obtain the current folder and the summary of its content. The response to the command below shows that the current folder is test and it has 6 messages. The message numbers range from 1 to 6 and the current message is the message number 6.

```
$ folder

          test+ has    6 messages (  1-   6); cur=   6.
```

The command **folder +report** will make the folder report as the current folder if it exists. If the folder does not exist, then the system will enquire whether users wish to create a new folder named report. If users respond positively, the folder will be created.

The command **folders** will display all the folders and their summary. The command does not display any subfolder if there is any. To see all the folders and all the subfolders, users have to use the **recurse** switch. The following command displays folders, subfolders and the detailed information is suppressed by using the **fast** switch.

```
$ folders  -recurse  -fast
archive
colleague
colleague/england
colleague/wales
fskuk
inbox
test
```

- Aliases

Aliases can shorten the email address and hence make it easier to remember. Instead of typing a long address, users may simply use a short and easy to remember alias. Aliases also can be assigned to a group of addresses. The alias can be used whenever users send a message to the group.

Users have to update the alias file from time to time. Users may add new alias, delete or change an existing alias. To display the alias file, users have to type the **ali** command.

• Online Reference

PP provides a UNIX style online reference which can be accessed by using the **man** command. Detailed information about any command can be reviewed. For example, **man pick** will display a complete information about the command.

A review of switches provided by each command can be accessed by the help switches. For example, **scan -help** will display all the switches available with the scan command.

Users may enquire about all the possible options available by entering *carriage return* when they are prompted to enter the next command. For example, when they are prompted with what now?

```
What now? ( RETURN)
Options are:
 display [<switches>]
 edit [<editor> <switches>]
 list [<switches>]
 push [<switches>]
 quit [-delete]
 refile [<switches>] +folder
 send [<switches>]
 whom [<switches>]
```

The discussion on facilities provided by PP is to demonstrate the flexibility and power of the system. Unfortunately, it is hard to use and only expert users can utilise the power of the system. For example, to send an email message, users have to use a primitive line editor and then quit the editor using the UNIX command **ctrl-d**, which is quite awkward for non-expert computer users.

CHAPTER FOUR

DESIGN ISSUES FOR USER INTERFACES AND NETWORK SERVICES

4.0 INTRODUCTION

Networked services are highly interactive applications which require effective and efficient user interfaces. It has been established that properly designed graphical user interfaces (GUIs) are effective for interactive applications as they support both an intuitive presentation of information on the screen as well as easy interaction with the underlying system (Kappel & Min Tjoa, 1992).

This chapter discusses general design issues related to the prototype which is to be developed. The issues include the way users input commands and data to the system; how information and commands are presented on the screen; management of windows; feedback mechanisms to assist users interacting with the system such as transformation of pointer images; implementing online help facilities.

The design is based on a graphical user interface and it makes use of user interface design principles that were reviewed in chapter 2 section 2.5.

4.1 INPUT DEVICES

In some applications, the keyboard is the primary input device. Users type in commands and the application responds with typed responses or prompts. Users' interaction with the prototype is mainly by a pointing device. A pointing device supports the implementation of direct manipulation, which is an important aspect of a user interface. Using a pointing device, users may invoke or quit an application or activate various commands in an application.

A pointing device is essential for implementing the concept of see-and-point. Objects which can be manipulated are available on the screen. Users may select any of these objects and perform activities related to the object. The result of the users' action is immediately displayed on the screen.

The pointing device used for the interaction with the prototype is the mouse. The mouse and the three buttons are shown in figure 4.1. The first two buttons, namely the **select** and **operate** mouse buttons are used to communicate with the prototype. For the remainder of the thesis, **<select>** and **<operate>** will be used to describe the two mouse buttons respectively. Left and middle mouse buttons are used when describing the buttons to users.

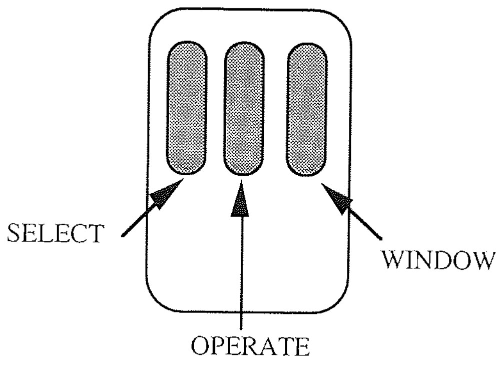


Figure 4.1 : Mouse Buttons

In the interest of consistency, the prototype defines standard functions for each of the mouse buttons. The <select> mouse button supports two main functions:

- Selecting objects to operate on.
- Manipulating objects and controls. These include selecting an item from a menu, clicking a button and moving a window.

The <operate> mouse button also supports two main functions:

- Accessing pop-up menus.
- Activating the online help facility.

The <window> mouse button is not used by the prototype.

A decision has to be made regarding the number of mouse buttons to be used for the prototype. The advantage of using all the three buttons is that more functionality can be placed on the buttons. For example, the Smalltalk-80 uses both the <operate> and <window> buttons to invoke different pop-up menus. The disadvantage of using many buttons is that novices have to recall the association between the button and function every time they want to carry out a certain task.

Some of the basic mouse vocabulary which is commonly used are 'move', 'press' and 'click' (Apple, 1992; Sun Microsystems, 1990). The meaning of these terms are:

Move - slide the pointer without pushing any mouse button.

Press - push a mouse button and hold it.

Click - press and release a mouse button while the mouse remains stationary. These terms will be used through out the thesis to describe users' interaction with the prototype.

Even though the mouse is the main input device for the prototype, the keyboard is still an effective device especially for functions that require text entry. A window or an editable view that accepts keyboard entry usually contains a special cursor which points to the next input character. A prompter is a special dialogue window that asks a question and waits for a response. The response must be a textual entry, where users have to key-in using the keyboard. An example of a prompter is shown in figure 4.2.

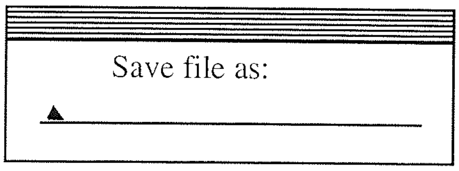


Figure 4.2 : Prompter

4.2 THE MAIN MENU

Access to the networked services can be made by the see-and-point paradigm. The prototype would display most of the networked services available on the main menu and users may invoke any of the applications simply by pointing on the right item and clicking the <select> button. Figure 4.3 shows a typical example of a main menu.

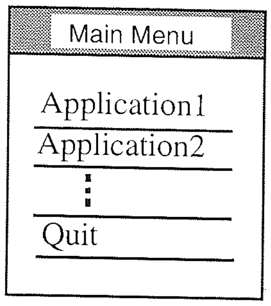


Figure 4.3 : Main Menu

When users invoke any of the items on the menu, the prototype would carry out the task of establishing the connection to the service. This simplifies the learning process to use the system because users have only one process to deal with, i.e. clicking a selected application as opposed to keying in variety of commands to establish the connection. Furthermore, the chances of users making mistakes would be greatly reduced because then the users would no longer be required to key in any command.

To invoke any of the items in the menu, users have to move the pointer onto the item in the main menu and press the <select> button. The item highlights to give users a feedback indicating that the item has been selected. It is a sort of acknowledgement that the system has heard the users' request. Releasing the <select> button while the item is highlighted causes the invocation of the selected item. Moving the pointer away from the item and releasing the <select> button removes the highlight from the item and cancels the invocation of the item. Such a visual communication is essential because it would make users feel that they are in control of the interface. A similar visual feedback is adopted for other controls such as buttons and pop-up menus.

The main menu is displayed on the screen every time users invoke the system and stays on the screen until users quit from the system. The continuous display of the menu is essential for the stability of the user interface. Since it displays all the choices which are available, it is the central point of reference by which users may launch various applications or to quit from the system. Users may iconise the menu if they wish to work on other applications. They may access the main menu again by double-clicking on it.

Activating Quit from the main menu causes the system to display a confirmer requesting a confirmation from users whether they really want to quit or not. If there are applications which are still active, the system would include a statement in the confirmer that all active applications will be turned-off by the system. Examples of confirmers are shown in figure 4.4.

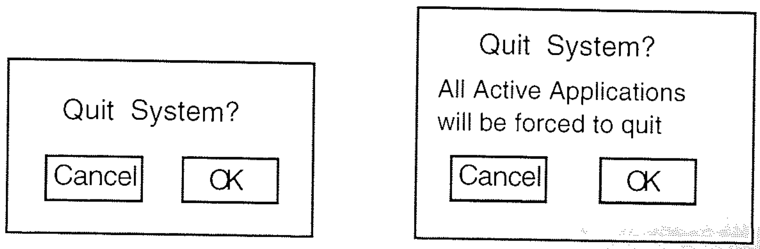


Figure 4.4 : Confirmers before Quitting

If users activate the OK Button, the system would further check for editing works which are not yet saved by users. A discussion on this subject is given in section 5.3.

A confirmer is a simple prompt from the interface that requests the users to confirm their action. Users have the choice, either to press the OK Button to confirm their action or press the Cancel Button to reverse their action. This is an example of reversibility where users are given the opportunity to return to the state just before they activated the command (Hix & Hartson, 1993).

A confirmer is also used to warn users if they choose a command which may cause a loss of data. This facility is essential because activating commands to the interface is so easy that users may by mistake choose a destructive command.

4.3 WINDOW MANAGEMENT

Users interact with the prototype through windows. Windows create a sense of perceived stability because they allow users to view and interact with all the different kinds of data (Apple, 1992). Since more than one window is usually required for the interaction and each one of them may behave differently, a proper management of windows is essential.

The prototype windows are mostly non-modal. Non-modal windows provide flexible interaction with applications. More than one window can be opened and users may work with more than one task a time.

Windows of the prototype can be classified into two types: primary and secondary windows. A primary window is the main window which is opened when users invoke an application from the main menu. It is through the primary window the main interaction between users and applications takes place. The interaction may be assisted by secondary windows.

A secondary window is a window which is generated through a primary window. The relation between the main menu, primary window and secondary window is hierarchical, i.e., a primary window is generated from the main menu and a secondary window is generated from a primary window. The relation is shown in figure 4.5.

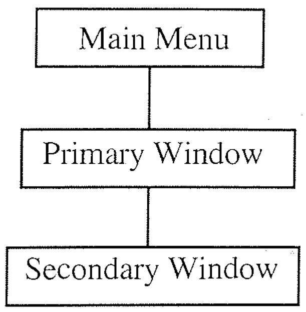


Figure 4.5 : Relation between Main Menu, Primary and Secondary Window

Even though the windows are hierarchical, users' interaction is not modal. Users working with any non-modal secondary window can simultaneously open any primary or secondary window. Users may also quit from any point in the system without having to back up through previous windows.

In some cases, a relation between the window and application has to be established. In a text editing environment, for example, warning must be issued if users try to close the window without saving changes to the text. Another example is the window that contains a running process, such as a process that checks for any new mail. The process must be forced to terminate whenever users close the window.

4.3.1 PRIMARY WINDOW

The primary window is an anchor for users; especially as, after opening so many windows, users can easily get lost and may wish to return to the primary window as a sort of restarting point (Hix & Hartson, 1993). Thus, there must be only one primary window for every application.

Since the primary window is the main place for interaction, its presentation is quite complex. It has to allocate space for variety of controls for users to input commands and to display different aspects of the application. In addition, the window has to allocate a supplemental view to help users in using the system.

Figure 4.6 shows a general layout of the primary window. It contains a set of buttons, main display, scroll list view, heading view and note view. The title of the window displays the name of an application and the heading view displays one important aspect of the application. In an electronic mail application, the heading view holds the name of the current folder.

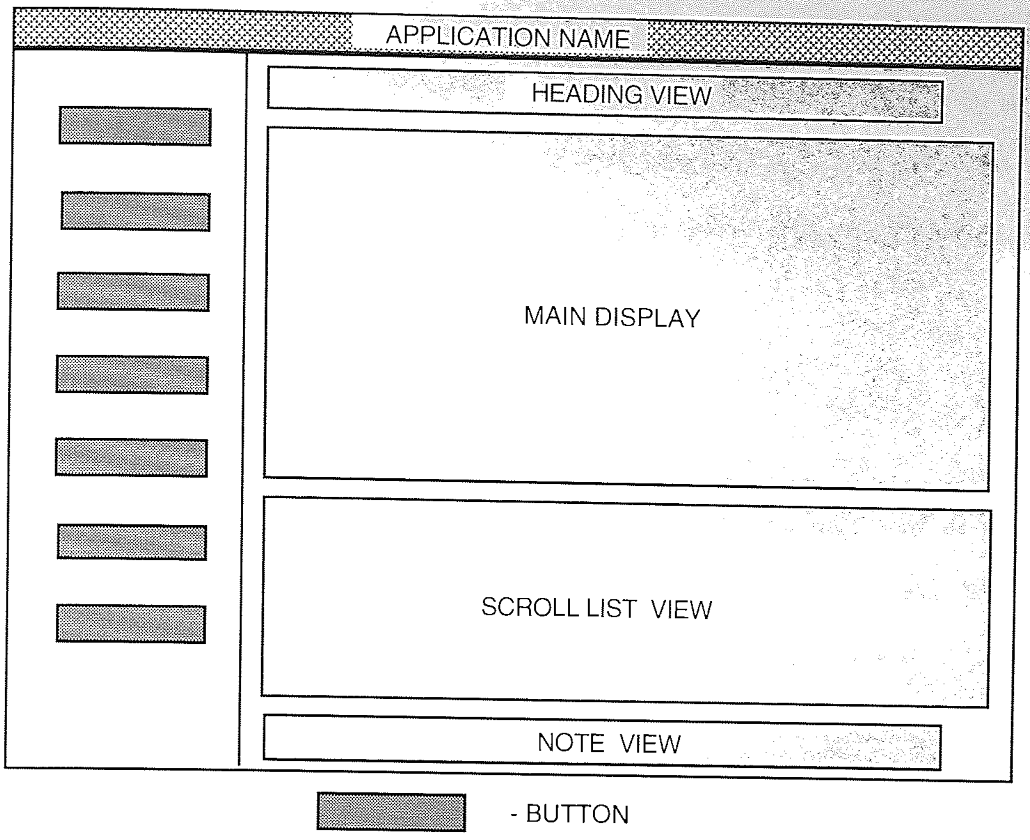


Figure 4.6 : General Layout of the Primary Window

The left part of the window is the main control area. The control area is like a control panel where users may input commands to the system. The area holds a set of buttons and each one of them is labelled according to its function. The buttons represent the main functions of the application and they are placed on the main window so that users are aware of the functions supported by the user interface.

The scroll list view is regarded as another control area since users may scroll the list or make a selection on any of its items. The scroll list view works together with the main display. The scroll list only holds a summary of items, and the content of the selected item would be displayed on the main display. Similar to the scroll list, the main display provides a scroll bar to allow users to view facts which are larger than the display. Both the scroll list and main display contain pop-up menus which can be invoked by pressing the <operate> button.

The note view is located at the bottom of the window. The functions of this view are to describe the function of command buttons in greater details and supervise users in using the interface. For example, the note view notifies users if they press the mouse button which is not used by the system. The view is described in greater detail in section 4.5.

4.3.2 SECONDARY WINDOW

The reasons for classifying the window are to simplify the management of windows and to make the user interface consistent. The secondary window can be further categorised according to its function. The summary is shown in table 4.1.

	Types of Window	Change the Primary Window	No. of windows allowed to open	Close with the Primary Window
1	Subservient window	Yes	one	Yes
2	Independent window	No	one	No
3	Text editor window	Depends on function	more than one	No

Table 4.1 : Classification of Secondary Windows

A subservient window is the window that causes a change to the primary window. The change may happen immediately or after users press the OK Button or enter a *carriage return* from a secondary window. Apple (1992) states that deciding when users' input on a non-modal window affects the primary window is an issue that needs to be further studied. In some cases, the users' input on the secondary window immediately updates the primary window and in other cases the users' input updates the primary window after pressing the OK Button.

A typical example of a subservient window is a prompter. Figure 4.7 shows a prompter and the primary window. After users have entered their request, the result is placed on the primary window, i.e., on the scroll list view.

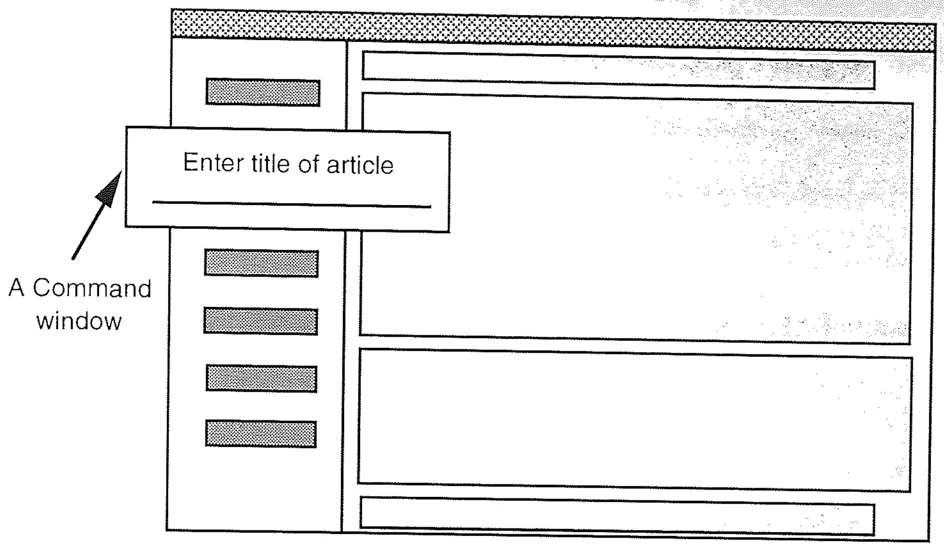


Figure 4.7 : Prompter - A Typical Subservient Window.

An independent window does not cause any change to the main window. Its functions are normally completely detached from the main window or other windows in the system. A typical example of an independent window is a window that maintains a list of names and email address. Users may update the list and the result is placed in the window itself. Another example of an independent window is a help window. The window displays information required by users.

The prototype allows users to open only one subservient or independent window for a particular function at one time. If users try to open any of these windows which are already open, the prototype would activate that particular window. This means that, if for example, the window is overlapped by three other windows, it would be placed on the top of all the three windows.

Unlike a subservient or independent window, a new text editor window is opened every time users invoke the function causing this window to open. Users may wish to open more than one text editing window so that they may compare or copy the contents from one document to another.

Quitting an application causes the primary window to close. This causes subservient windows which are dependent on the primary window to close. Some of independent windows which are related to the primary windows, such as help windows are also closed. All text editor windows are left open when the primary window closes.

This means that the system must maintain a list of all windows which must be closed when the primary window closes. All the windows in the list are forced to close with their primary window when users quit the application.

4.4 SCROLL LIST VIEW

The scroll list view is used extensively by the primary as well as by the secondary windows of the prototype. The scroll list view can be divided into three types according to the number of item(s) that can be selected. Selection can be made by moving the pointer to an item and clicking the <select> button.

Exclusive selection - where only one item is selected at a time. Selecting another item will automatically deselect the current selected item.

Exclusive (Variation) selection - where one or none of the item can be selected. Users can have none of the items selected by clicking the selected item.

Non-exclusive selection - where none, one or multiple items can be selected. Users may select one or many items from the list.

A browse through a collection of items with hierarchical structure can be made using a combination of two or more coordinated views or windows (Shneiderman, 1992). The

Smalltalk-80 System Browser contains four scroll list views to help users browse through the class categories, classes and methods supported by the system.

The prototype uses a combination of two scroll list views to help users browse through the hierarchy of folders. The folder browser is shown in figure 4.8. The arrow buttons below the views help users to browse through the different levels of folders. The view at the top displays the pathname of a selected folder.

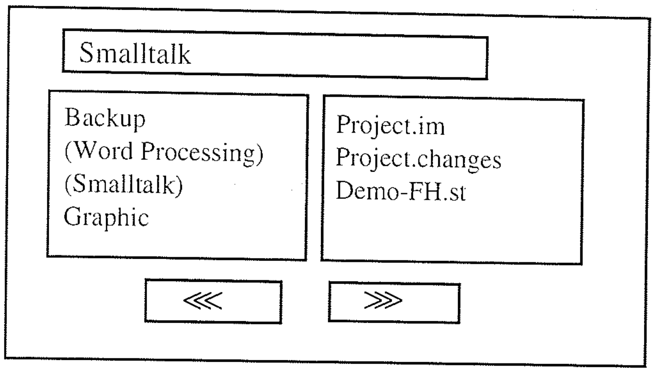
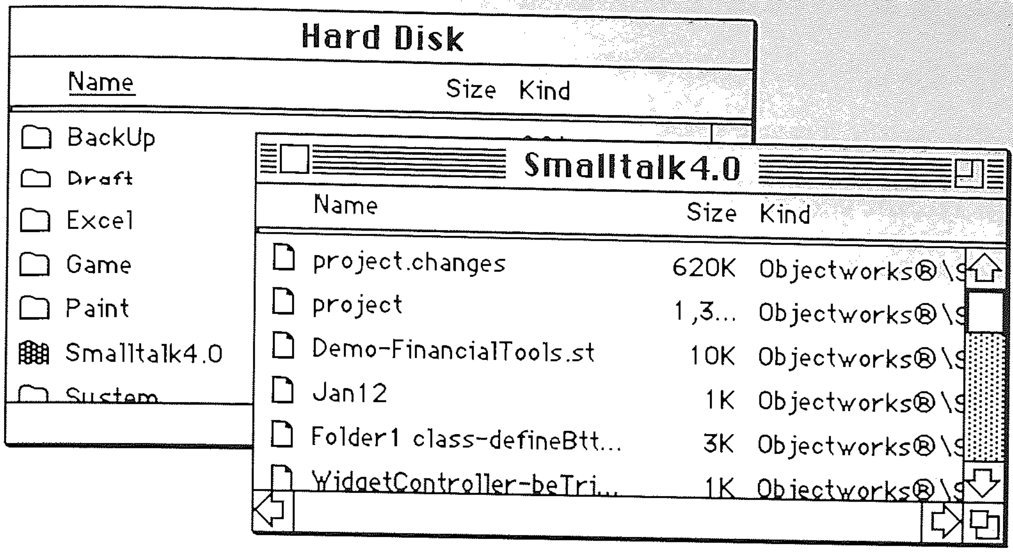


Figure 4.8 : Folder Browser

The advantage of using scroll lists for browsing through folders is that users can view a subfolder without opening any additional window. Figure 4.9 shows the method of viewing a subfolder as implemented by the Apple Macintosh. Users have to open a window every time they want to look for files and folders in a subfolder. Sometimes a new window overlaps over the main window and users have to keep closing window, otherwise the screen will be full of windows.



```
Hard Disk
Name                        Size  Kind
BackUp
Draft
Excel
Game
Paint
Smalltalk4.0
System
```

```
Smalltalk4.0
Name                              Size   Kind
project.changes                   620K   Objectworks®\S
project                           1,3... Objectworks®\S
Demo-FinancialTools.st            10K    Objectworks®\S
Jan12                             1K     Objectworks®\S
Folder1 class-defineBtt...        3K     Objectworks®\S
WidgetController-beTri...         1K     Objectworks®\S
```

Figure 4.9 : Opening a Folder on the Macintosh System

4.5 NOTE VIEW

The note view can be regarded as a multiple purposes view. The main task of this view is to help users using the interface. The note view can be broadly categorised into the following five functions:

- Describes the function of command buttons in brief.
- Helps users to complete a task.
- Advises/Navigates.
- Checks for errors.
- Indicates status.

The note view briefly describes the function of the button that contains the pointer on it. Users can review the function of other buttons by moving the pointer onto the buttons. Every button is labelled according to its function. The label is usually a single word which does not precisely describe its functions. A label 'TITLE' may refer to a title of an article, a title of a book, a title of a conference or a combination of any of them.

Completion of some tasks may require the user to execute more than one command in the right sequence. For example, the creation of an alias for electronic mail requires more than one step. After users have entered an alias, the next step is to include the real electronic mail address in one of the edit view. Since the sequence of actions may not be clear to users, the note view has to describe the next action in the sequence.

A similar interface design is implemented by the Excel version 4. The software contains a panel at the bottom of the screen that describes different commands on the pull-down menus and navigate users in using the system. Some tasks, such as building a graph, require the users to carry out more than one instruction. After users have selected the graph icon on the interface, the panel instructs users to drag the pointer to obtain the result. Such an instruction is a valuable piece of information for novices and infrequent users because it encourages them to explore the interface with confidence.

A pop-up menu can be activated from the scroll-list view. The advantage of this type of menu is that it requires less screen space. The disadvantage is that, users may not realise that a pop-up menu is available with the view. The note view advises users to make a selection when the pointer is on the scroll list view and the view is not empty. If a selection is already being made, the note view informs users about the presence of the menu.

On some windows the note view displays errors committed by users. For example, users may try to delete folders which are not empty and creates files using previously used names. The note view on the primary window displays a special message if users invoke a mouse button not used by the system. In a graphical user interface environment, invoking an inappropriate mouse button can be regarded as an error.

System 7 on the Macintosh offers users an advice-giving capability when users commit errors. For example, the system prompts a message when users try to install Flash-It facility in the system folder, rather than in the control panel. The message is shown in figure 4.10. The system also takes the task of correcting the error if users press the OK Button.

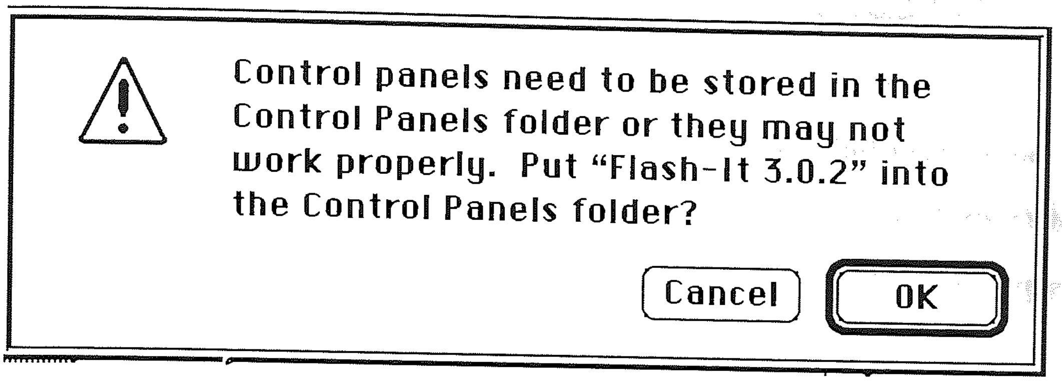

Control panels need to be stored in the Control Panels folder or they may not work properly. Put "Flash-It 3.0.2" into the Control Panels folder?

Figure 4.10: System 7 Advising Capability

The note view is also used as status indicator. The number of articles found can be quite large and to retrieve them may require a considerable length of time. The note view displays a counter showing the number of articles that have been retrieved. From the counter users may estimate how much longer they have to wait.

4.6 BUTTONS

Buttons of the prototype system may either immediately execute a command or open a window. The two buttons are shown in Figure 4.11a and 4.11b respectively. The button that causes a window to open is usually marked with the ellipsis character (...) after its label. The ellipsis character is a hint to users that they should expect a window when they invoke a button. According to Apple (Apple, 1992), the ellipsis character is used to inform users that the command requires more information to execute.

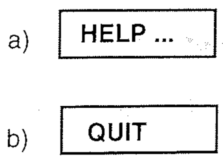


Figure 4.11 : Labelling Buttons

The use of ellipsis character to indicate a dialogue is adopted by popular GUIs, such as Macintosh, OPENLOOK and Window 3.1 (Apple, 1992; Sun Microsystems, 1990; Microsoft Corporation, 1991; ParcPlace Systems, 1992). ParcPlace Systems recently incorporated this character for Smalltalk-80.

A consistent visual feedback is an effective means of informing users what is happening. The buttons of the prototype can be in normal, inactive or busy state. Different visual feedback are used to present the different state of the buttons. Figure 4.12 shows the three visual responses to represent different states of buttons.

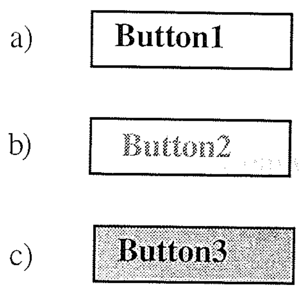


Figure 4.12 : Different States of Buttons -
a) Normal b) Inactive c) Busy

The following is the description of the behaviour of a button during the three states:

Normal - this means that the command represented by the button is ready for an execution. To activate the command, users have to move the pointer onto the button and press on <select> mouse button. This causes the button to be highlighted and the command will be activated only when users release the mouse button. If users move the pointer off the button before releasing <select>, the highlight will be removed from the button and the command will not be activated.

Inactive - this means that the command represented by the button is not ready for an execution. This could be due to the fact that the command is not valid at that particular time or the application is busy executing other commands. Pressing the button at this time will not have any effect.

Busy - this means that the command represented by the button is being executed. This state is represented by the button in the highlighted state. Pressing the button at this time also will not have any effect. When the execution is complete, the button will return to its normal appearance.

4.7 POP-UP MENUS

The pop-up menu contains commands which are relevant to its view. The advantage of pop-up menu is that users are not required to move the pointer away to execute commands. For example, after selecting an item from the scroll list, users may press on the <operate> button and execute delete from the pop-up menu to delete the item. Furthermore, a pop-up menu supports the concept of progressive disclosure. Functions which are of secondary importance are available but not displayed on the window. This makes the interface look simple for novices but provides sufficient functionality for expert users.

Many graphical user interfaces use a greying-out technique to represent functions which are currently invalid. Such a technique helps users to avoid errors by making invalid functions unavailable. The greying-out approach can lead to frustration if users do not know why the desired function is not available (Hix & Hartson, 1993). Some views of Smalltalk-80 flicker to indicate that pop-up menu is not available when users press the <operate> button on them. Such kinds of feedback discourages users using the system.

Rather than greying out the pop-up menu, the prototype displays a help message in the note view whenever users try to prompt the pop-up menu without making any selection. The help message would request users to make a selection before invoking a pop-up menu. If there is no selection to be made, the help message simply says that the view is empty.

4.8 POINTERS

Changing the image of the pointer is another effective way of communicating with users. This is due to the fact that for most of the time, users' attention is on the pointer. A common example is a word processing environment that changes its pointer from the I-beam pointer image to a normal pointer image when users move the pointer from the text editing window to the scroll bar. The text editing image indicates that users are in text editing environment and the normal image indicates that users are in direct manipulating mode.

Microsoft Word version 5.1 allows users to select a column in a box by a click of a mouse when the pointer image is transformed into a down pointed arrow. Users are effectively informed about the capability of the application by the proper transformation of the pointer. Figure 4.13 shows the difference in the image pointer when users move the pointer to the top of a box.

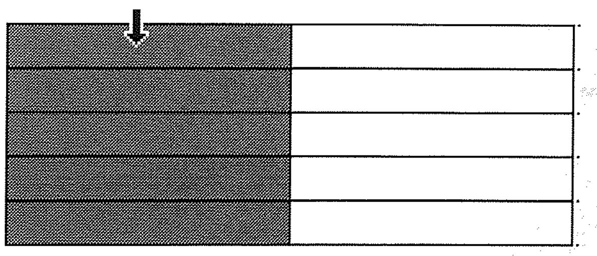


Figure 4.13 : Pointer Image in Microsoft Word 5.1

Different pointer images of the prototype used by the prototype are displayed in table 4.2. Each of these images represent its own meaning.

Pointer	Name	Used For
↖	Arrow	Normal mode
☝	Pointing Hand	Selecting mode (e.g. clicking)
⌶	I-beam	Text-editing mode
⌛	Hour-glass	Busy mode

Table 4.2 : Different Pointer Images

The following is the description of the behaviour of the pointer:

Normal - the pointer shows that the system is ready for an execution. Users may click any button or select any item from the scroll list view.

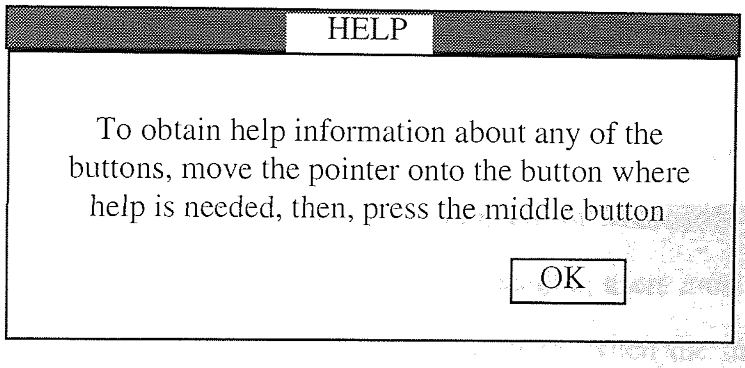
Pointing - the pointer changes to a pointing mode when users move the pointer onto a button or onto a scroll list. It will not change to pointing mode if the application is busy or the scroll list view is empty. The pointing image would help users to differentiate between a passive display and a control. They may click <select> mouse button when a pointing image appears.

Text - the text mode pointer is used for a text editing view. This type of pointer is the most appropriate for text editing because users can easily insert the next input character precisely.

Busy - the pointer changes to a busy mode when the application is initialising or executing a command. During this mode, the system will not respond to any input from users.

4.9 ONLINE HELP

One of the buttons on the primary window is labelled 'HELP'. This button functions as a navigator which explains to users how to obtain the online help. Activating this button by clicking <select> causes a help window to be displayed. The window and its content are shown in figure 4.14. A help window is displayed on the top right side of the screen so that it may not block the primary window.



HELP

To obtain help information about any of the buttons, move the pointer onto the button where help is needed, then, press the middle button

OK

Figure 4.14 : Help Window

Activating the same button by clicking the <operate> button causes a help window that holds information about the application itself. When users press the <operate> mouse button on any button on the primary window, a single entry pop-up menu labelled 'HELP' appears. If users release the mouse button without moving the pointer away, a help window is displayed on the screen. Figure 4.15 shows the behaviour of the button when users press the <operate> mouse button.

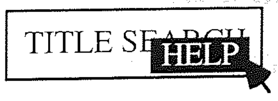


Figure 4.15 : Activating HELP

Activating Help will cause an online help information to be displayed on a special window similar to the one shown in figure 4.14. The system automatically puts users into a relevant subset of help information based on the request made by users. For example, if they request help from a search button, information regarding the subject is provided to them. They are not required to browse through an index and select the most appropriate keyword.

The Help Button is placed at a strategic place, i.e. together with other command buttons on the primary window so that users know how to obtain help when they need to know more information about the interface. A navigator that explains how to obtain help is as important as the help information itself. Many users are not using help simply because they do not know how to obtain it.

Duffy *et al.* (1992) point out the design flaw of accessing help in Microsoft Word 3.0 for the Apple Macintosh. The same design is adopted for Microsoft Word 5.1. To access the help system for this word processing application, users must pull down the apple menu, choose 'About Microsoft Word', and then, when the dialogue box is displayed, they must click on a button that says 'Help'. The problem with this design is that, the help command is not continuously visible on the screen and this is compounded by the fact that novices may not even realise the existence of pull-down menu behind the apple icon.

Openwin is an example of an interface that does provide enough clues for users to access its massive context-sensitive online help. Users may obtain information on almost every object on the screen by moving the pointer onto the object and press the keyboard labelled 'HELP'. In a graphical user interface environment, the user's attention is directed towards the object on the screen and the main interaction is through the pointing device. Since the keyboard is only a secondary input device, there is a tendency for the user not to be aware of the HELP key. Furthermore, the reliance on the keyboard would restrict the portability of the system. For example, the Macintosh may not have the HELP key on its keyboard.

The online help of the prototype is similar to the online help of the Openwin in the sense it is not modal. This is opposed to the System 7 Apple Macintosh online help which requires the users to be in help mode in order to access the online context-sensitive help. The online help facility can be turned on and off by users. When the facility is on, it keeps on displaying a balloon help whenever users move the pointer around and passes through different objects on the screen. The facility displays a balloon help for every object even though users may need help only for one or two objects. Users may lose concentration on the task they are working on by the balloon which keeps on flashing on the screen.

Duffy *et al.* (1992) highlight other points which are related to the design of help system. First, the help information must be displayed in less than eight seconds on the screen. Second, users must be advised to seek help when they commit errors. These points have been considered in the design of the prototype.

4.10 ESTABLISHING A CONNECTION AND TIME OUTS

The prototype is communicating with various applications on different computers through a communication network. Some of these applications reside on computers in the university and others reside on computers in different parts of the United Kingdom. Some of these applications may require a longer time to be initialised. This could be due to several factors, such as distance, large database, and network traffic. Some require user login validation.

Since invoking applications may require some processing time, the pointer image has to be changed into a busy mode. This shows that the prototype is establishing a connection or the applications need time for initialising. For applications which require a much longer time, such as BIDS, the display of status indicator is effective because it displays the estimated total time and the elapsing time of the operation. The indicator also gives information about the different stages of the operation.

Some applications have time-out facility where they are automatically disconnected if users do not enter anything for a certain period of time. Inappropriate time-outs, such as a window suddenly disappearing would strip users of the feeling that they are in charge of the interface (Hix & Hartson, 1993). A proper time-outs facility should prompt confirmer asking users whether they still want to work with the application. The application would be closed if users select 'NO' from the prompter or if users disregard the window for some time.

CHAPTER FIVE

USER INTERFACE DESIGN OF THE PROTOTYPE

5.0 INTRODUCTION

A prototype has been developed to illustrate user interaction with certain networked services. This consists of software which gives the user access to an electronic mail application, Bath Information Data Service (BIDS) and file editor. The file editor serves as a support service for the networked service applications.

The general design issues related to the prototype have been discussed in chapter 4. This chapter discusses issues which are specifically related to the application being developed.

Users interact with networked services by means of functions provided by the applications. Functions are activated by controls and the results or the output of these functions have to be placed on displays. Decisions have to be made in order to distribute controls and displays in the most effective way, so that it is easy for users to execute these functions and obtain their results.

5.1 ELECTRONIC MAIL

The first electronic mail systems simply consisted of file transfer protocols that hold the recipient's address on the first line of every file (Tanenbaum, 1989). Users have to edit a file, then save the file and quit the editor and finally invoke a file transfer program. At that time, the application was used by computer experts who needed to communicate among themselves. These people understand how computers work and they do not have much problem using the application.

Since there are many advantages provided by electronic mail, the application has been introduced to professionals in various fields. These people are mostly non-computer experts and they wish the application to work the way they think rather than being dictated by the system.

The basic functions of electronic mail are creation of messages and, sending and receiving of messages. Other functions such as archiving of messages and aliasing are also important in order to make the use of the application more effective.

Invocation of electronic mail and other applications supported by the prototype can be made by selecting the application from the Launcher. The Launcher is shown in figure 5.1.

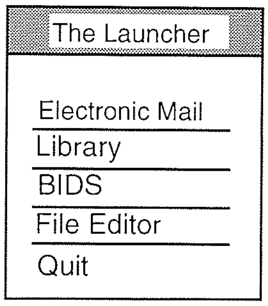


Figure 5.1 : Launcher

Figure 5.2 shows the primary window of the electronic mail application. The window contains a number of components which are needed by users to interact with the application. These include the heading view at the top of the window that displays the current folder. The scroll list view displays the list of messages in the folder. The buttons represent the commands which are supported by the application. The explanation of each of these commands is placed on the note view at the bottom of the window. The explanation of any of the buttons can be accessed by placing the pointer on the button.

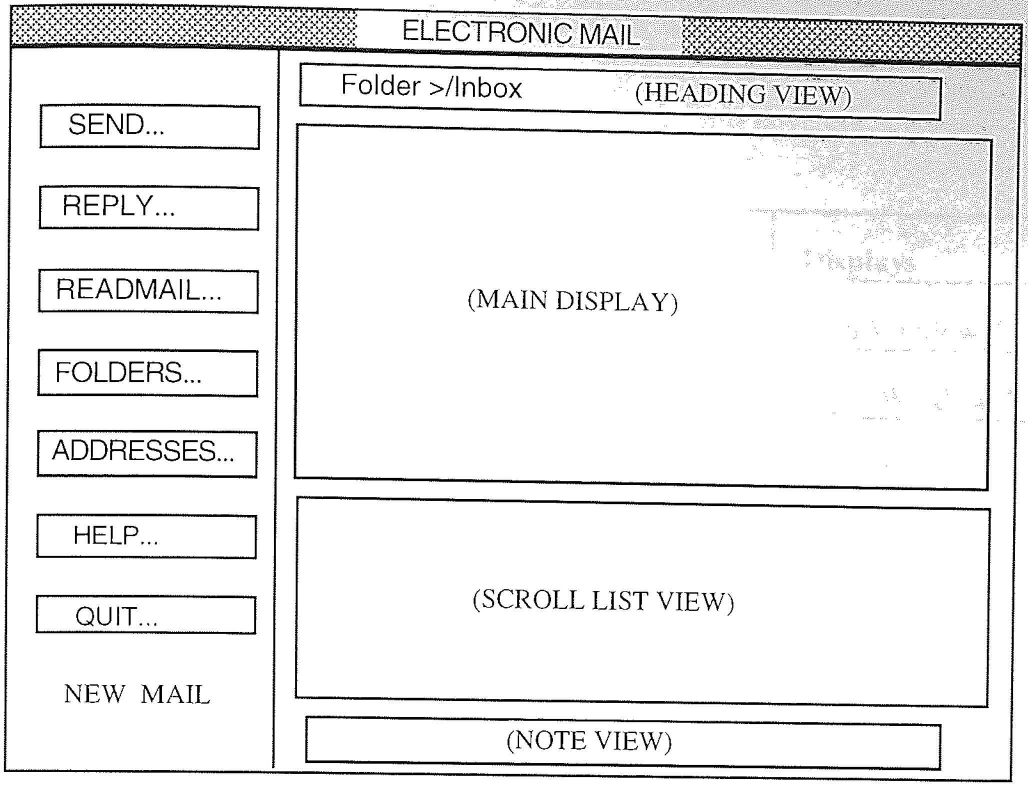


Figure 5.2 : Electronic Mail Primary Window

A function comprises two components, i.e. a **control** and a **display**. A **control** is a means whereby users input a command to the system and a **display** represents a means whereby the result of users commands can be obtained. Since the primary window represents the main place of interaction to the system, it has to accommodate as many functions as possible. By viewing the primary window users may have ideas about the capabilities of the application. Proper judgements have to be made because placing so many functions on a window causes clutter and this reduces the effectiveness of users' interaction.

The electronic mail application uses buttons to represent a group of related functions. For example, the Folder Button represents functions which are related to the folder operations such as activating and creating folders. Invoking the button causes the Folder Window to open. The window provides all the functions which are related to the folder.

Table 5.1 displays the functions of electronic mail which have been placed on the primary window of the prototype.

Functions	Controls	Displays
1) Send a new mail	Send Button	(Send Window)*
2) Reply to a selected message	Reply Button	(Reply Window)*
3) Read new mail	Readmail Button	Main Display
4) Folders management	Folders Button	(Folders Window)*
5) Address management	Addresses Button	(Address Window)*
6) Obtain the online help	Help Button	(Help Window)*
7) Quit the application	Quit Button	
8) Display the contents of a message	Scroll list view	Main Display
9) Deletion and refiling of a message	Pop-Up Menu	
10) Display the current folder	(Folder Window)*	Heading View
11) Display the summary of messages	(Folder Window)*	Scroll list view
12) Mail arrival notification	-	Display View
13) System and error messages		Note View

(window)* the display is in a secondary window

Table 5.1: Functions Supported of the Electronic Mail Primary Window

Send and Reply Functions

Send and reply are the basic functions of electronic mail. Users may wish to send new mail or reply to mail sent to them. Both the functions operate in two stages:

- Compose a message
- Send a message

Composition refers to the process of creating messages for electronic mail communication. Although any text editor can be used to create the content of a message, a special purpose electronic mail text editor usually provides an extra space for a message header where users can enter the name of the recipients, subject of the message and carbon copy. In the case of replying to a message, the editor extracts the sender's address and automatically places it on the message header.

Figure 5.3a) and b) show the Send Window and Reply Window respectively.

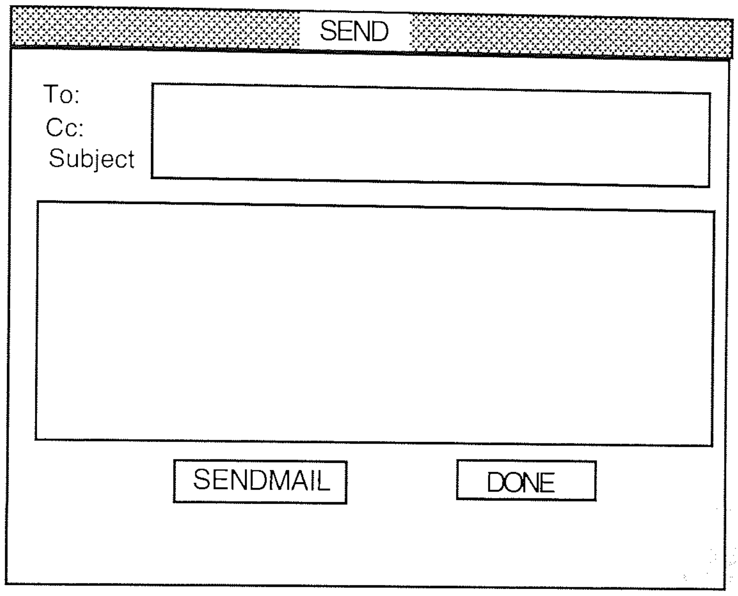
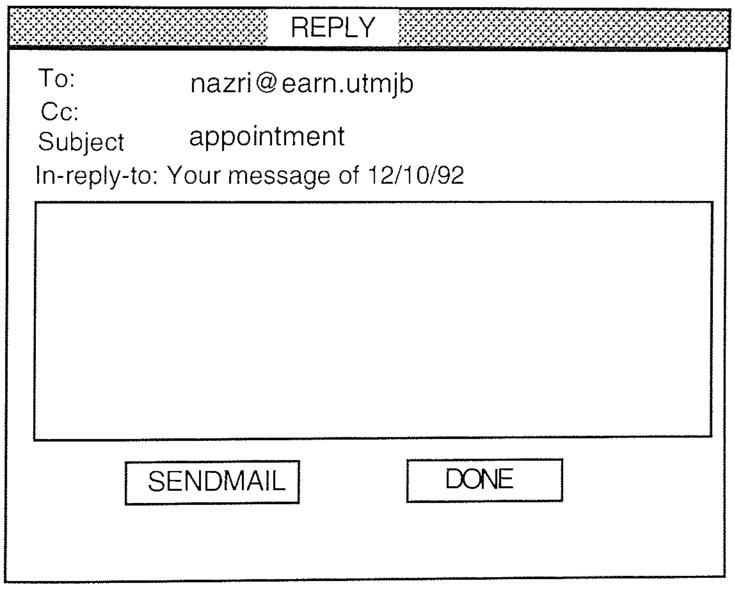


Figure 5.3a) : Send Window



To: nazri@earn.utmjb
Cc:
Subject appointment
In-reply-to: Your message of 12/10/92

Figure 5.3 b) : Reply Window

Some electronic mail interfaces such as Openwin and Xmh contain a button labelled 'Compose' on their primary window. The word 'Compose' is a technical word and it is not an appropriate word when communicating with users (Berry, 1992). Activating this button causes a text editor window to open. Users may send their messages after editing by activating a button labelled 'Deliver'. The button is located on the text editing window. The reason for placing the Compose Button on the primary window is that users have to compose a message first, then send the message by using the Deliver Button. The problem with such an arrangement is that the primary function, i.e. send is hidden by the function compose.

The differences between send and reply functions are as follows:

- The send function is always valid. The reply function is only valid when a message is specified. Hence, the Reply Button is greyed to indicate that the function is not available when no message is specified.

- The reply function automatically fills-up the message header of the text editor. The send function leaves the message header empty and it has to be filled up by users.

- The pop-menu in the text editing view of the send/reply function supports the basic text editing functions such as copy, cut and paste. The pop-up menu for the reply function contains an extra feature, i.e. include. The feature allows users to extract the sender's message into the editing view and at the beginning of each line the character '>' is inserted.

The send function has to provide an extra text-editing view for the message header. The view provides the following capabilities:

- Basic editing functions such as **copy**, **cut** and **paste**. The function paste for example, allows users to paste an address which has been copied from any electronic mail message.

- Scanning the users input. If the input is an alias name, the system converts it into a real electronic mail address.

- Error-checking capability. The system will make sure that at least one recipient is specified in the message header. An error message is prompted on the window if a message is sent with an inappropriate address.

Table 5.2 summarises the functions of the electronic mail text editor window.

Functions	**Send Message**	**Reply Message**
Send the message	SendMail Button	SendMail Button
Quit the edit window	Quit Button	Quit Button
Edit the content of the message	Edit View	Edit View
Edit the content of message header	Edit View	(Display of message header)

Table 5.2 : Electronic Mail Text Editor Window

Sending or replying to a message can be activated by using the SendMail Button. Sending/replying is a process that requires more than one step. The steps include scanning of the message header (send function only), users' confirmation, address validation and sending the message.

The flowchart for the sending process is shown in figure 5.4. Sending a message is an irreversible action. Once sent, the message is transmitted through the network and the action cannot be cancelled or reversed. Hence, a confirmer is prompted to the users whenever they activate the SendMail Button. Users are given a chance to review their message before they confirm that the message is complete and ready for transmission.

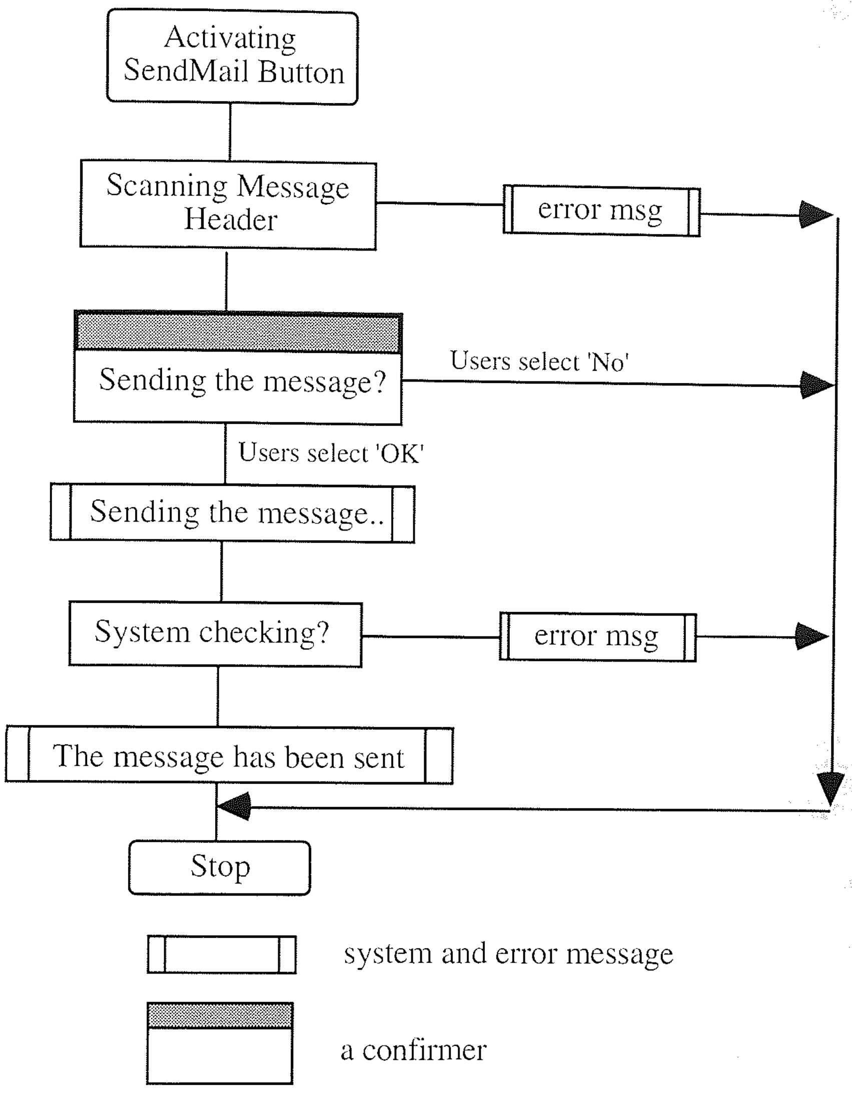


Figure 5.4 : Sending/Replying to Mail Messages.

The SendMail Button is highlighted and the pointer image is changed into an hour glass until the message is completely sent. This is to indicate that the system is sending the user's message. Messages are displayed according to the status of the sending process. If the message is successfully sent, a note confirming that the message has been sent is displayed in the note view. Similarly, if the operation fails, a note mentioning its problem is displayed.

Feedback messages regarding the message sent are quite important due to the following reasons:

- Users are informed that the message has been sent by the application.
- There is no other way for users to check whether the message has been sent or not. In the case of other commands such as delete a file, users may display a listing of a directory to confirm that the file has been deleted.
- Users may be interrupted by phone calls or visitors while working with the application. By the time they return to the application they have to recall where they stopped and whether they have sent the message or not.

Whenever users send or reply to a mail, a copy of the message is kept in the users' current folder. Users may wish to review the message that they sent before they communicate with the same person the next time.

Read New Mail

The electronic mail application contains a process that checks for any new incoming mail. The purpose of the function is to keep users informed whenever new mail arrives. Users who are expecting mail do not have to check their mail manually.

Whenever any new mail arrives, the following actions take place:

- A message 'New Mail' in Red Colour is displayed at the lower left of the primary window. This indicates that there are new messages which are not yet read by users.

- The ReadMail Button becomes normal. The ReadMail Button is similar to the Reply Button in a way that it is greyed unless the command is valid. In the case of the ReadMail Button, it is greyed when there is no new mail (the mail has been read) and turned to normal when new mail arrives.

- A message is prompted informing users to click the ReadMail Button to obtain the new mail.

Whenever new mail arrives and users invoke the ReadMail Button, the new mail is placed on the scroll list view of the primary window and the 'New Mail' message disappears from the window. Users may read new mail by selecting the summary from the scroll list.

An alternative design is to open a special read mail window that holds all new mail messages. When the window is closed, all the messages are placed on the primary window. The advantage of this approach is that, it is much easier for users to review a list of new messages because they are placed on a special window. This design is appropriate if users receive many messages every day. The disadvantage is that opening an extra window is required, thus interaction is difficult for novices and cumbersome for experts.

Electronic Mail messages

An electronic mail message can be viewed in two ways:

- Its summary - a single line description about the message. A summary of messages can be regarded as a simple index for users to access the real content of the electronic mail message. Since a message summary is smaller in size, more information can be placed on the screen without occupying a large area.
- Its content - a complete electronic mail message which comprises a header and a body. The content of an electronic mail message occupies a relatively larger area. The number of electronic mail messages displayed at one time must be controlled so that it does not clutter the screen space.

Many electronic mail applications simply place the sender's address in the message summary. Hence, the users own address appears in the summary if it is an outgoing message. It is more meaningful for users if they can identify:

- whether the message is a copy of their own message that they have sent out or the one they have received from somebody else.
- with whom they are communicating. This means that the summary should holds the recipient address if it is an outgoing mail.

The summary of a mail message of the application may appear in either one of the formats below:

- If it is a received message:

  < date > < sender's name > < subject >*

* - subject may not appear in the summary if it is not available in the message.

- If it is a sent message:

  < date > To: < recipient's name > < subject >*

A mail message usually contains sender/recipient names and real mail addresses in the 'To' or 'From' component of its header. For example, the 'To' component in a message is as below:

    To: Giles Christian <GNC1@uk.ac.rutherford.ibm>

The mail address is enclosed in the '< **...** >' brackets and the name is the word that appears before the address. The summary of messages contains the name of the sender/recipient rather than the real address because the name is the way the senders/recipients wish to identify themselves. If the name is not provided by the message, then the address is extracted.

The summary of electronic mail messages is placed on the scroll list view. The scroll list view is a special view since it has the following capabilities:

- Displaying of information - A list of items is displayed on the view.
- Acting as a control - Users can select an item from the list.
- Scrolling capability - This allows an indefinite number of items to be placed on the view.

Functions that can be applied on electronic mail messages are: **display, delete, refile** and **reply.**

- **Display** - selects any of the message summary in the scroll list view causes a display of its content on the main display. The scroll list view holds a summary of messages of a current folder. Users simply click on the message summary if they wish to view its content. They are not required to enter a message number to select a message.
- **Delete** - the interface prompts a confirmer when users try to delete a message. After users have confirmed that they want to delete, then the message is deleted.
- **Refile** - a refile window is opened when this function is activated. The function allows users to move messages from one folder to another.
- **Reply** - a reply window is opened when this function is activated. On the primary window, the reply function is placed on a button. An alternative is to place the function in the scroll list select menu. It is placed on a button because, it is an important function for electronic mail.

A combination of the scroll list view and main display is an effective way of controlling the screen from clutter. Only the selected message has its content displayed on the main display at one time. Selecting another message deselects the current selection and the content of a newly selected message is placed on the main display.

Folders Management

Smith (1991) states "the most valuable enhancement on any electronic mail front end is the facility for managing folders, ... ". After using electronic mail for some time, users may find their directory full with messages. They may wish to organise and archive their messages in different folders as they normally organise their files.

The folders management function of the prototype is delegated to the Folder Window. The Folder Window contains a heading view which displays the active folder. Browsing through the hierarchy of electronic mail folders is assisted by a pair of scroll list views and a pair of navigational buttons. Figure 5.5 shows the Folder Window of the application.

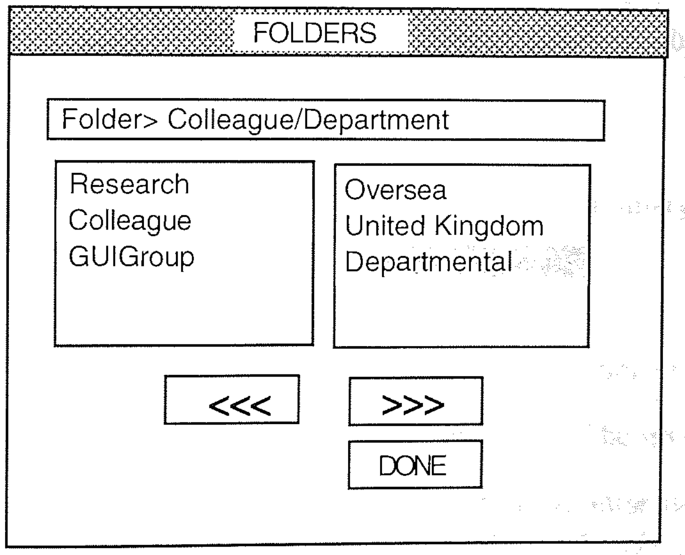


Figure 5.5 : Folder Window

The Folder Window also contains commands such as creation and deletion of folders. The commands can be accessed from the pop-up menu in the scroll list views. The window informs users about the presence of those commands when users select folders in the scroll list view.

Table 5.3 summarises functions which are supported by the window.

Functions	**Control/Display**
Display of current folders	Heading View
Browsing through folders	A pair of Scroll List View
Creation and deletion of folders	Pop-Up menu
Moving up and down in the hierarchy of folders	the button labelled '<<<' and '>>>'
Refiling messages	Pop-Up menu (on the primary window)
Quit the Folder Window	Done Button

Table 5.3 : Functions of the Folder Window

Selecting a folder in the scroll list view causes the selected folder to be the current (active) folder of electronic mail. When the active folder is changed, the following *immediate* results appear:

- The heading views on the Folder Window and on the primary window display the current folder.
- The scroll list view of the primary window displays the summary of messages of the new active folder.

The Folder Window is an example where the effect of users' choices on a non-modal window are immediately displayed on the primary window. The immediate result of the users' selection on the Folders Window is essential because users can see the summary of messages in different folders while browsing through their folders.

Initially a hierarchical menu was devised to help users to select folders. The advantage of using a menu is that users are not required to open a window whenever they wish to activate other folders. The menu system is shown in figure 5.6.

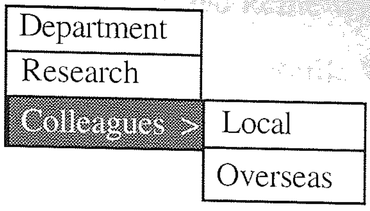


Figure 5.6 : Activating Folders Using A Hierarchical Menu

The problems with the menu are:

- It becomes messy when the number of folders or/and the depth of the folder hierarchy increases.
- It is not easy to incorporate other folder operations such as create and delete folders

Refiling messages from one folder to another can be invoked from the pop-up menu on the primary window. Activating refile function causes the Refile Window to open.

The Refile Window is to assist the user to select the target folder. The window is similar to the Folder Window. It contains an extra button, the Refile Button to allow users refile messages. Figure 5.7 shows the Refile Window.

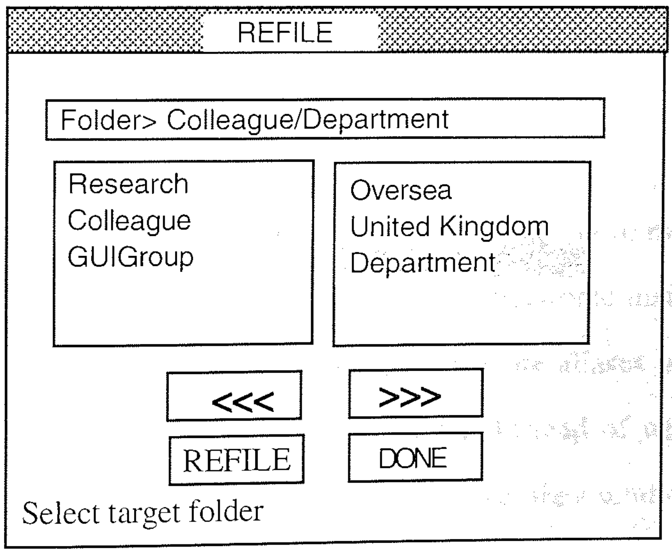


Figure 5.7 : Refile Window

The difference between the Folder Window and Refile Window are:

- Selecting or browsing through folders on the Refile Window does not cause any change to the primary window.
- Folder operations such as creation and deletion of folders is not available on the Refile Window.
- The Refile Window contains the Refile Button to allow users to refile the selected message. The refiled message is moved to the selected target folder. The Refiled Button is greyed if no message is selected by the users or the target is not folder.

Address Management

Electronic mail addresses are usually long and hard to remember. The following is an example of a valid electronic mail address:

```
ESAM@UK.AC.BIRMINGHAM.ACADEMIC-COMPUTER-SERVICE.VMS1
```

It is also noted that users prefer to archive messages because they can simply reply to that message rather than sending new mail when they want to communicate with the same person. By replying to a message, users do not have to recall and fill-in the recipient's address which is quite cumbersome. The disadvantage of replying to a message is that users are not able to specify the subject of the message. The old subject of the message remains with the new message, even though the content of the message is different.

Using the Address Window, the prototype provides the facility to maintain electronic mail addresses. The Address Window contains alias electronic mail addresses. The use of aliases is convenient because users may associate aliases which are easy to remember with different electronic mail addresses. Instead of writing a complete address, users may simply specify a valid alias whenever they send new mail. Figure 5.8 shows the Address Window.

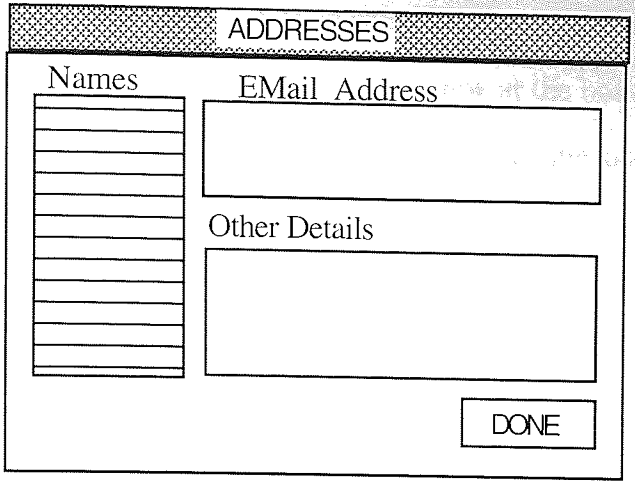


Figure 5.8 : Address Window

Table 5.4 shows functions supported by the Address Window.

Functions	Control/Display
Display a listing of aliases	Scroll List View
Creation and deletion of aliases	Pop-Up menu
Storing the real addresses	Edit View (top)
Storing other information -such as address and phone and fax number	Edit View (bottom)
System and error messages	Note-View
Quit the Address Window	Quit Button

Table 5.4 : Function Supported of the Address Window

Users may access a pop-up menu from the scroll list in order to add or delete an alias. Adding a new alias is an example of a task that requires more than two actions. The actions are:

- Invoke the pop-up menu from the scroll list and select new alias. This causes a prompter to be displayed to allow users to input a new alias.
- Enter a new alias at the prompter.
- Enter the real electronic mail addresses in the top left view.

If the new alias entered is a duplication of an existing alias, the alias is rejected and an appropriate error message is displayed in the note view at the bottom of the window. Having input a valid new alias, the note view reminds the users to include the corresponding proper electronic mail address in the top left view.

A two-field input prompter is used to allow an easy creation of aliases. Using this prompter, users may create an alias and its real address on the prompter itself. Interaction is easier because the whole task of creating an alias can be carried out in a single view. Figure 5.9 shows the two fields input prompter.

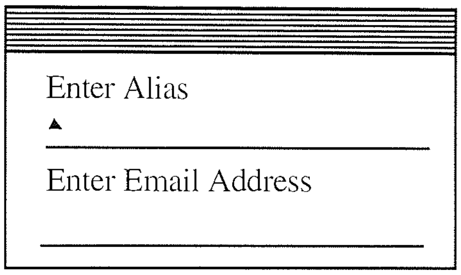


Figure 5.9 : Two-Field Prompter

The window supports two edit views (the views on the right of the window). The first view is to allow users to enter a real electronic mail address. The facility also allows users to store a mailing list. Mailing lists are usually aliases pointing to groups of users. Using the mailing lists facility, users can send a message to the whole group of recipients at once by specifying its alias. Users may enter as many addresses in the view as they like for a given alias. The addresses have to be separated by a space.

The second view allows users to include other details which are related to the selected alias such as departmental address, phone and fax number or comments. Both edit views allow users to update the information if there are any changes. They are supported by the basic editing functions. This allows users to copy an address or other information from a message and paste it on to it.

Window Management

The table 5.5 shows the classification of secondary windows of electronic mail. The way the windows are classified determines its behaviour in relation to the primary window. The discussion on the classification of windows has already been given in chapter 4.

Windows	**Classification**
Send and Reply Window	Text-Editing window
Folder Window	Subservient window
Address Window	Independent window
Help Window	Independent window

Table 5.5 : Classifications of Secondary Windows

5.2 BATH INFORMATION & DATA SERVICES (BIDS)

Users in the past have had to visit the library if they wish to carry out any literature search. Librarians are always available to assist them if they have any difficulty with their search. BIDS was introduced to provide the service which allows users to carry out search from their own departments. The information they need can be accessed through the network. Since users have to work out for themselves how to search for information, a good user interface is an important factor that determines the success of the application.

BIDS is accessible via the Janet computer network and it takes more than two minutes to establish a connection to the service. Since the waiting time is considered long, a status window is displayed on the screen when users activate BIDS. The window holds a status bar which displays the estimated length of time to complete the operation. The window also informs users from time to time about the different stages to establish the connection. Even though users have to wait for some time, they are being informed about what is going on with the system. Figure 5.10 shows the status indicator.

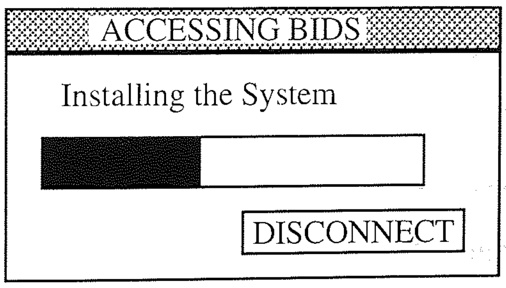


Figure 5.10 : Status Indicator

Establishing a connection to BIDS is not easy. First, users have to log-on to an intermediate host that supports a connectivity service to BIDS. Having logged-on to the host, users have to execute different procedures and enter a valid ID/password combination in order to be connected to BIDS. The prototype takes the task of establishing the connection to BIDS every time users activate the service. Hence, users are not required to recall the procedures and ID/password to access the service.

There is a question of access control since users are no longer required to enter their identification and password to access the application. It is noted that only members of subscribed organisations are allowed to use the application. Since the prototype is developed in a UNIX environment where only registered users (they are members of the organisation) can access it, to request BIDS identification and password is unnecessary.

BIDS may not be available at certain times of the day. This could be due to system maintenance. If the service is not available, the status window prompts a message indicating that BIDS is currently out of service. During a peak time, users have to queue before the connection to BIDS can be established. The status window prompts a message that users have to wait in a queue. The window provides users with the Disconnect Button to allow users to quit the service and try it later.

Figure 5.11 shows the primary window of BIDS. The layout of the window and its components such as the heading view and main display are similar to the electronic mail application. The functionality of these components are slightly different. For example, the heading view displays the current year and database the application is currently connected to.

The main display displays BIDS logo and introductory information about the BIDS services. The information is quite essential especially for users who have never used the services. It helps them to get started with the service.

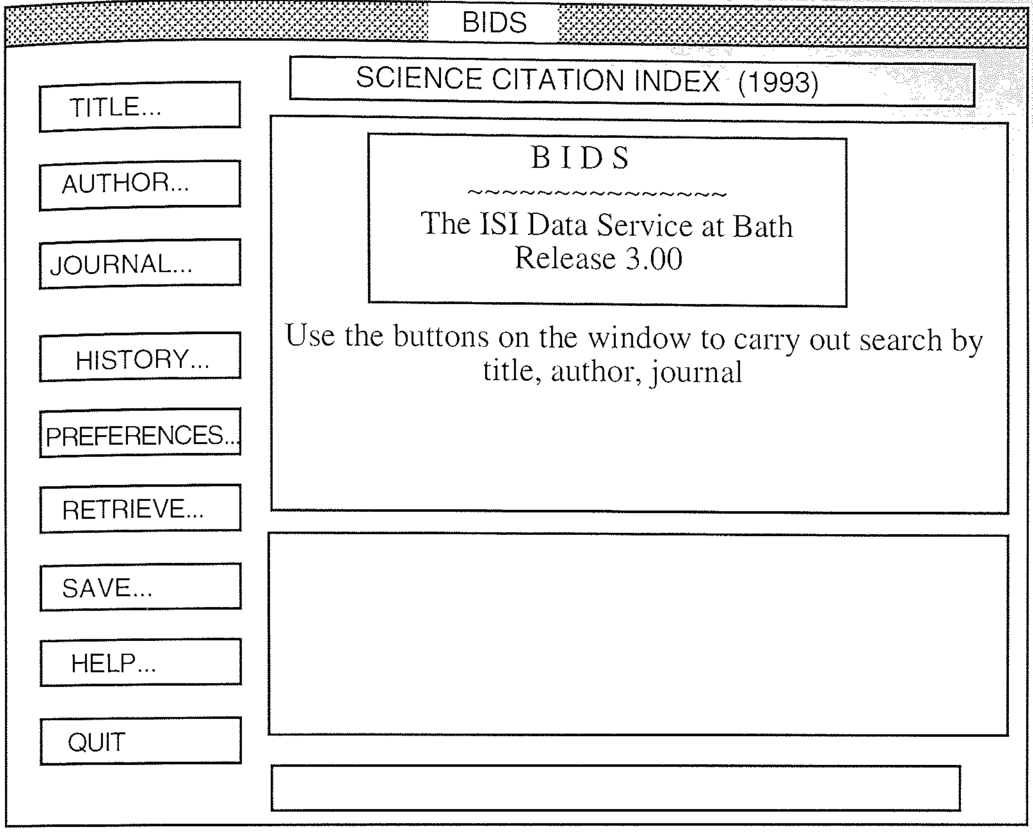

BIDS
TITLE...
AUTHOR...
JOURNAL...
HISTORY...
PREFERENCES...
RETRIEVE...
SAVE...
HELP...
QUIT
SCIENCE CITATION INDEX (1993)
B I D S
~~~~~~~~~~~~~~~~
The ISI Data Service at Bath
Release 3.00
Use the buttons on the window to carry out search by title, author, journal

Figure 5.11 : BIDS Primary Window

Searching for articles is the most important function of BIDS. The first three buttons on the primary window allow users to carry out searches by title, author and journal respectively. Activating any of these buttons causes a prompter to be displayed on the screen.

A prompter is a special window that requests users to enter word(s) for the search and it accepts input from the keyboard. The Smalltalk Prompter (see figure 4.2) does not contain any button to help users accepting or cancelling the operation. Using the prompter, users may cancel operations by entering an empty input. Such a procedure is not obvious for users who are not familiar with Smalltalk.

Figure 5.12 shows the prompter used by the prototype to allow users to input their search. When users invoke the OK Button or enter a *carriage return*, the input is accepted by the system and the prompter is closed. Users may also cancel the operation by invoking the Cancel Button.
~~~~~~~~~~~~~~~~

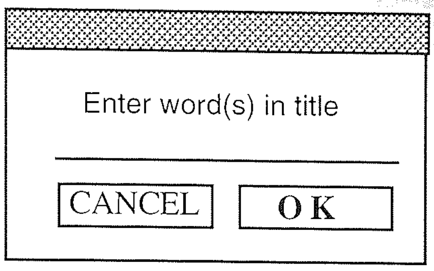


Figure 5.12 : BIDS Prompter

Other functions which are related to the service are also provided. Users may access these functions via buttons which are available on the primary window. Table 5.3 summarises the functions of BIDS which have been placed on the primary window of the prototype.

Functions	Controls	Displays
1) Search articles by title	Title Button	(prompter)
2) Search articles by author	Author Button	(prompter)
3) Search articles by journal's name	Journal Button	(prompter)
4) Review of previous searches	History Button	(History Win*)
5) Set the citation index and year	Preferences Button	(Preferences Win*)
6) Retrieve saved searches	Retrieve Button	(Retrieve Win*)
7) Save searches	Save Button	(Save Win*)
8) Obtain the online help	Help Button	(Help Win*)
9) Quit the application	Quit Button	-
10) Display a summary of searches		Scroll list view
11) Display the result of a search	Scroll list view	Main Display
12) Select citation index and year	(Preferences Win*)	Heading View
13) System and error messages		Note View

Win* - window

Table 5.3 : Functions Supported by the BIDS Primary Window

Searching Process

When users enter a search, the button for the search is darkened until the search is found. This is to indicate which search operation is being processed. The pointer is also changed into a wait mode pointer. To change the pointer also has its own advantage because users' attention is always at the pointer.

Table 5.4 shows the three steps involved when users carry out a search.

Step	Commands	Results
1	Press a search button	A prompter is displayed
2	Enter search	The summary of the result (number found) is displayed on the scroll list view. (The view flickers 2 times)
3	Select the items	The result is displayed on the main display

Table 5.4 : Searching for Articles using BIDS

After users have entered a search from the prompter, its partial result is placed on the scroll list view of the primary window. The scroll list view flickers to bring users' attention to the results of the search. The partial result is a one-line summary of the result that contains the following information:

- the number of articles found.
- the words that users have entered.
- search type, i.e., title, author or journal.

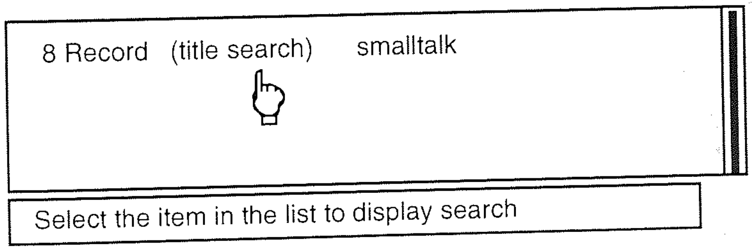
The summary of the result is of dual function:

- Allows users to recall back the search that they have made.
- Puts the users in control of the interaction. Users may decide whether they wish to see the list of articles or not. If the number of articles found is too high, it takes more time to retrieve the search and it is harder to browse

through a longer list of articles. If they wish, users may refine their search. For example, if the number of articles found for the word 'interface' is too high, users may re-enter another search such as 'user interface'.

If the search was not successful, i.e. no record was found, the summary of search will contain a note stating that there is no record found. At the same time, the note view prompts a message advising users to consult the online help facility. The failure to obtain a result could be due to improper input from users. As an example, users have to input 'dialog' rather than 'dialogue' because only American spelling is accepted by BIDS. If users try to select the summary, a message is placed on the main display of the primary window stating that 'No record found'.

If the search is successful, the summary of the result indicates the number of articles found. The pointer is forced to move so that it points to the summary of search and a message is displayed on the note view at the bottom of the window requesting the users to select from the summary of search in the scroll list to see the result. Figure 5.13 shows the scroll list view and note view after users activated the Title Button and entered 'smalltalk' from the prompter. Eight articles having the word 'smalltalk' in their title are found.


8 Record (title search) smalltalk

Select the item in the list to display search

Figure 5.13 : Scroll List View and Note-View

Selecting the summary in the scroll list view causes a complete display of the result on the main display. Since to retrieve the complete result takes some time, the pointer is changed into a busy mode pointer. The note view also provides a message stating that search is being processed and the record number of the current processed record is printed on it.

There are only three steps required to obtain the final result of a search. Keeping a number of steps small makes the search easy for novices and less cumbersome for experts. In addition, entering commands to the system is mostly done by clicking a mouse button on the right item. As a review, the steps to search for articles are:

- Click the search button.
- Enter the words to be searched at the prompter.
- Select the search summary in the scroll list view.

Main Display

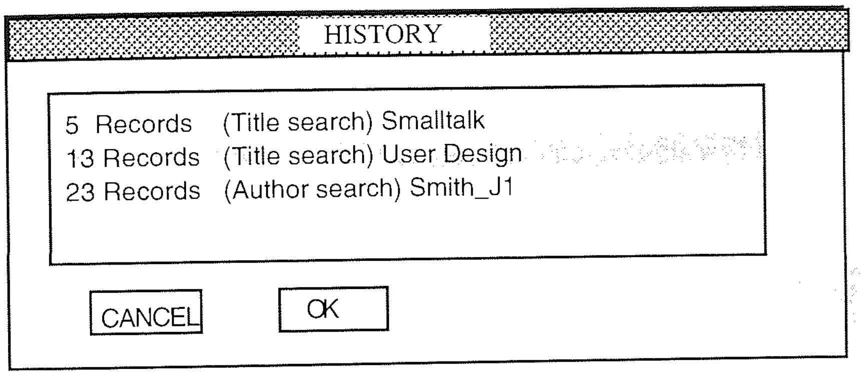
The main display is a read-only view on the primary window. The display occupies the largest area of the primary window. Its primary function is to display the result of a search. It contains a pop-up menu which supports the following functions:

- Copy - allows users to copy part or all of the information on the main display to a text-editing view such as a file editor.
- Copyright - obtains an information regarding copyright issues of the service
- Messages - obtains an information about the message of the day from the service. The message may contain information about shut down time or any update in the services.
- Hardcopy - allows users to obtain a hardcopy of the search listing.
- Search.. - allows users to search for the word specified by users. Such function is useful especially when the listing is quite long. If the word that users are looking for is found, the note view informs users to carry out repeat function to find the next word.

- Repeat Search - allows users to search for the next word in the listing after they had carried out the search function.

History Function

The history function allows users to review the searches that they have made so far. Whenever users activate a new search, the previous search is kept in a history and it can be reviewed using the history function. The function can be activated by the History Button and this causes the History Window to open. Figure 5.14 shows the History Window. The window holds a summary of archived searches in its scroll list view.



5 Records (Title search) Smalltalk
13 Records (Title search) User Design
23 Records (Author search) Smith_J1

Figure 5.14 : History Window

The scroll list view is modified to allow users to carry out multiple selection. Users may select zero, one or more items from the scroll list view. When users select the OK Button, the window is closed and all the selected items are displayed on the primary window, i.e. the scroll list view. Users may review the full listing of searches on the main display by selecting the items.

Save/Retrieve Function

The save function allows users to save to the disk the searches that they have made. Activating the Save Button causes the Save Window to open. The figure 5.16 shows the Save Window of the application. Similar to the scroll list view of the History Window, users may select one, two or more items which they wish to save. Figure 5.15 shows the Save Window.

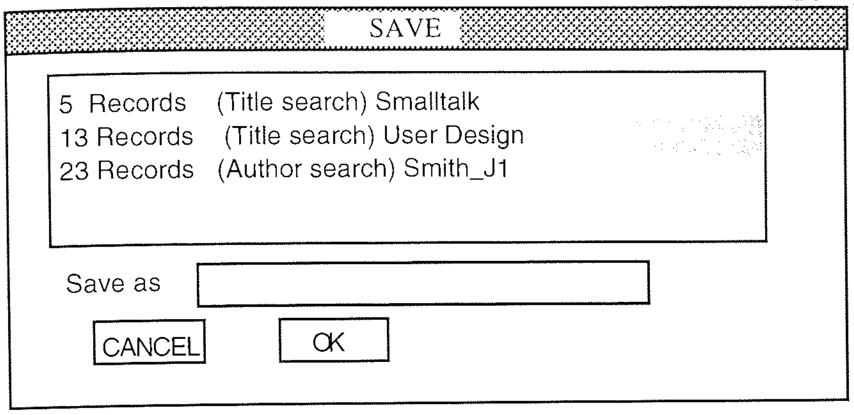

SAVE
5 Records (Title search) Smalltalk
13 Records (Title search) User Design
23 Records (Author search) Smith_J1
Save as
CANCEL
OK

Figure 5.15 : Save Window

The window contains a special view for users to write the name of the file where the searches are to be saved. Users may write the most appropriate file name so that it would be easy to retrieve the next time. The system will make sure that a valid input is entered by the users, otherwise an error message is displayed on the note view.

The Retrieve Button allows users to retrieve searches they have saved. Activating this button causes the Retrieve Window to open. The window is shown in figure 5.16.

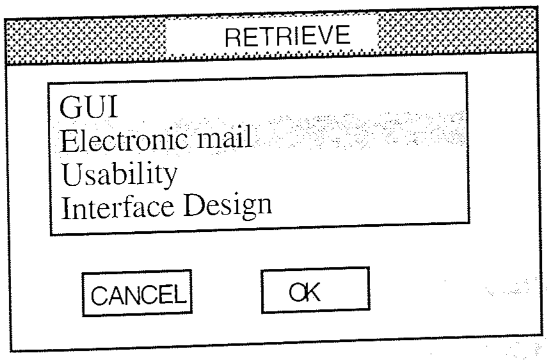


Figure 5.16 : Retrieve Window

The window displays the filenames that contain previous saved searches. The display of files help users recall back the files that they had saved. When users select files and click the OK Button, all the summaries of searches in the selected files are displayed on the scroll list view of the primary window.

Preferences Function

The preferences function is used to change various settings of the interface. The window contains a set of radio buttons to represent the different database indexes. Figure 5.17 shows the Preferences Window of the application.

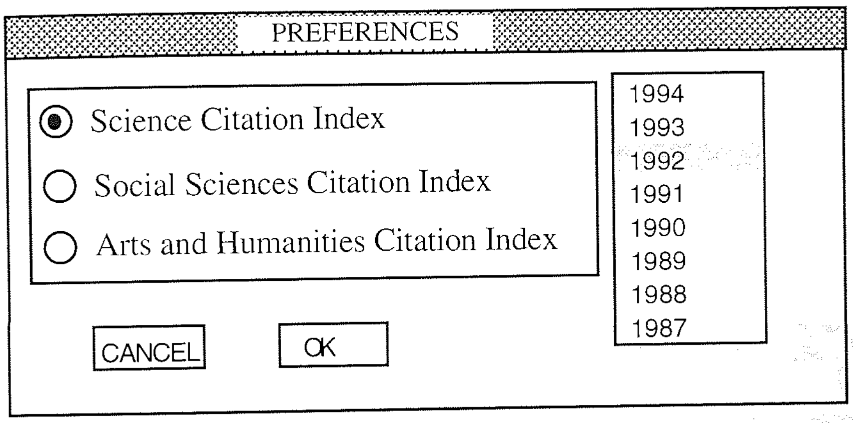


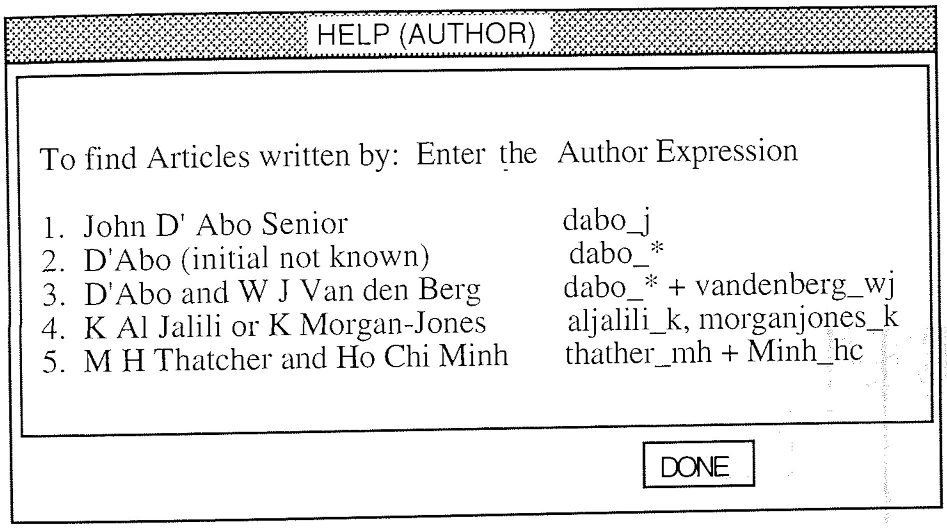
Figure 5.17 : Preferences Window

The use of radio buttons is appropriate in this case because the number of items available are constant and exclusively one item can be selected. Likewise, only one item has to be selected for the year selection. In the case of the year selection a scroll list view is used because the list can easily accommodate an increase in number of items in it. Each year the interface must automatically increment the selection list to include the current year.

Selecting the *Science Citation Index* and the *year 1993* causes the interface to be connected to the index for 1993. The result of the selection is placed on the heading view on the primary window. Any search carried out will be referred to articles in the selected index and year.

The secondary windows of BIDS are mostly subservient windows and thus they are dependent on the primary window. They are closed whenever users quit BIDS from the primary window. Only one window of the same function is allowed to open at one time. Invoking an active function (window for the function already open) causes the window that supports that function to be activated.

Users may access help facility by bringing the pointer onto the button where help is needed and press the <operate> button. Figure 5.18 shows the help window when users invoke help from the Author Button.



To find Articles written by:	Enter the Author Expression
1. John D' Abo Senior	dabo_j
2. D'Abo (initial not known)	dabo_*
3. D'Abo and W J Van den Berg	dabo_* + vandenberg_wj
4. K Al Jalili or K Morgan-Jones	aljalili_k, morganjones_k
5. M H Thatcher and Ho Chi Minh	thather_mh + Minh_hc

Figure 5.18 : Help Window

5.3 FILE MANAGER/EDITOR

A file editor is essential in a networked service environment. Information obtained from a library search, BIDS or USENET can be copied onto a file, manipulated and sent for printing. Users also may wish to access some or all of the information in a file and send it to colleagues via the electronic mail facility.

The file manager is the primary window which allows users to carry out various operations which are related to files and folders. Figure 5.19 shows the file manager of the prototype.

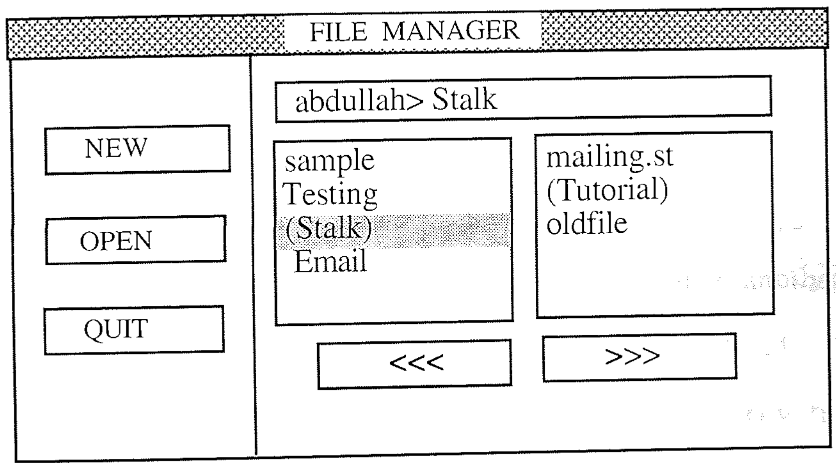


Figure 5.19 : File Manager

When the file manager is opened, the left scroll list view displays all folders and files in the users home directory. For security reasons, files which are not related to the system are not displayed by the manager. A folder can be differentiated from a file by a pair of brackets which enclose its name. Selecting a folder causes its content to be displayed on the right scroll list view. Using the arrow buttons users may traverse through the hierarchy of folders. The heading view at the top of the manager displays the path of the selected folders.

Table 5.5 shows the functions supported by the manager.

Functions	**Control/Display**
Open a new file	New File Button
Open an existing file	Open File Button
Quit the File Manager	Quit Button
Display of current folders	Heading View
Browsing through files and folders	A pair of Scroll List View
Creation and deletion of folder	Pop-Up menu
Moving up and down in the hierarchy of folders	the button labelled '<<<' and '>>>'
Deletion of files	Pop-Up menu

Table 5.5 : Functions of the File Manager

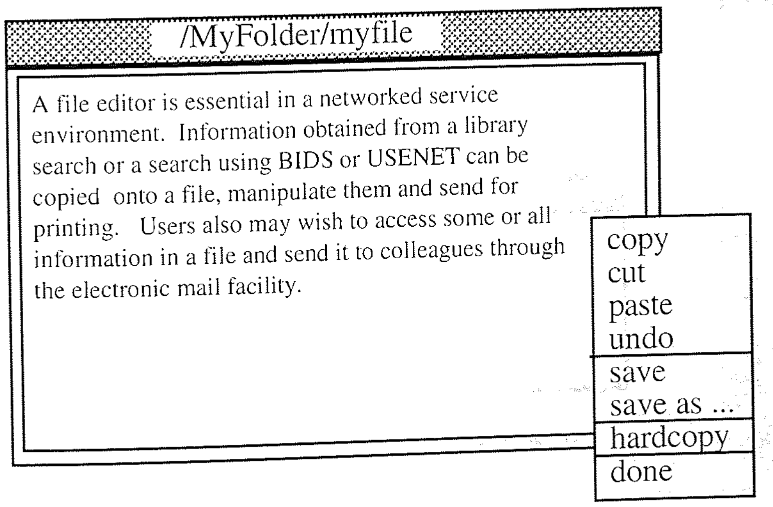
The New Button allows users to create a new file editor. Pressing this button causes a new file with a title 'newFile' to open. If users open another new file before closing the 'newFile', the title of the new file is 'newFile1'. Similarly, opening subsequent new files before closing older ones creates a new file with the same title, appended with a different integer. However, these are temporary titles and users are prompted to input a title of their choice when they save their work.

New files or folders are created under the current selected folder. The current folder is specified by the heading view of the file manager. The result of the creation or deletion of folders or files is immediately displayed on the file manager. This immediate display of the result of the user action is very important because users will gain confidence with the interface. They are confident that the interface is responding according to their request. This is an encouragement for them to explore the system further.

The Open Button is used to open an existing file. This button requires users to select a file from the manager before clicking on it. If no selection is made, the button is greyed to indicate that the command is not available. The button also helps users to differentiate between a folder and a file because the button appears normal when a file is selected and turns grey when a folder is selected.

It takes more time for users to open a file in a windowing environment. The file manager transforms the pointer into an hour-glass to indicate that the computer is processing the users' request. Such a feedback may seem trivial because it appears only a few seconds on the screen, but the lack of any feedback causes anxiety to the users. There is a tendency that users make more mistakes with the interfaces that do not provide any feedback.

Operations which are related to folders and files can be carried out from the pop-up menu which is available on the scroll list. The operations include creation and deletion of new folders and deletion of files. Figure 5.20 shows an example of a file editor and its pop-up menu.



A file editor is essential in a networked service environment. Information obtained from a library search or a search using BIDS or USENET can be copied onto a file, manipulate them and send for printing. Users also may wish to access some or all information in a file and send it to colleagues through the electronic mail facility.

Figure 5.20 : File Editor Pop-Up Menu

A file editor window supports the following functions:

- undo - reverses the most recent cut or paste.
- copy - places a copy of the highlighted text in memory.
- cut - places a copy of the highlighted text in memory, then delete the text.
- paste - deletes the highlighted text (if any), then place the most recently copied or cut selection in that location.
- save - saves the edited text to the file. If it is a new file, and the filename is not yet specified by users, the system would prompt for a filename.
- save as... - the system would prompt for a filename. The text would be saved in the file specified by the users. The system would check for any error in specifying filename. If it is a duplicated filename, users have the choice of either cancelling the operation or overwriting the existing file.
- hardcopy - prints a copy of the text on paper.
- quit - quits text editing and close the window.

Whenever users close a text-editing window, the system checks to make sure changes made are saved by the users. If the content is not yet saved, a confirmer is prompted on the screen. Figure 5.21 shows a confirmers prompted by Smalltalk.

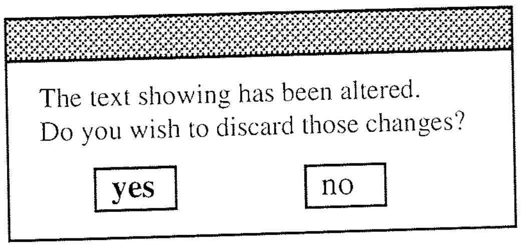

The text showing has been altered.
Do you wish to discard those changes?

Figure 5.21 : Smalltalk Confirmer

The Smalltalk confirmer does not allow users to save edited works directly. Users has to select 'no' from the confirmer and then save their work using a pop-up menu provided. Such an implementation should be suitable for Smalltalk so that the Smalltalk codes are protected from any accidental input from the keyboard.

The confirmer for file editor is modified to allow users save their work directly from the confirmer. The confirmer is shown in figure 5.22.

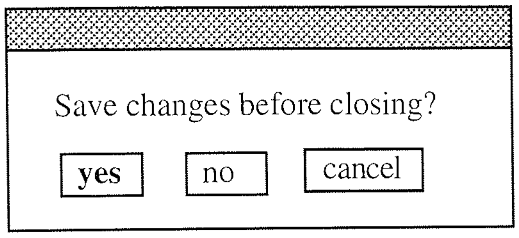


Figure 5.22 : File Editor Confirmer

File editors reside on text-editing windows which are independent of the file manager window. Thus, users are allowed to open many file editors at one time and closing the file manager does not effect file editing window.

Snaphots of the prototype windows are placed in appendices C.

CHAPTER SIX

IMPLEMENTATION ISSUES

6.0 INTRODUCTION

This chapter discusses issues related to the implementation of the prototype. Discussion includes the advantages of an object-oriented approach and the Smalltalk language. The chapter also reviews the concept of Model-View-Controller (MVC) methodology adopted by Smalltalk and examines each of the components in depth, using simple examples and experiences from developing the prototype to illustrate variations of the MVC theme. The source code of the prototype is available from **cs.aston.ac.uk** ftp site in the bg/write.me/hanan folder

6.1 OBJECT-ORIENTED APPROACH

The importance of object-oriented programming for the development of modern user interface is emphasised by Graham (1991). He states "the sheer complexity of modern GUI makes object-oriented programming a necessity rather than a luxury: if it didn't exist then GUIs would have forced us to invent it". He quotes the example of Apple Lisa that requires 200 man years of effort, much of which was dedicated to the development of the interface. It was inconceivable for Apple to move to the newer series of hardware without reusing the Clascal code used for the Lisa.

The human mind and object-oriented systems are similar in a way that they perceive a world problem as a hierarchy of elements which interact with one another. Object-oriented programming helps programmers to think at a higher level about a problem and thus reducing the gap between programming and the way a programmer thinks (Che Mee, 1990). Unlike object-oriented programming, conventional programming

forces the programmers to think according to the way the machine works.
Four key concepts that explain the advantages of object-oriented approach are abstraction, encapsulation, inheritance and polymorphism.

The meaning of abstraction given by the Oxford English Dictionary closest to the meaning intended in this discussion is "the act of separating in thought". Graham (1991) defines abstraction as "representing the essential features of something without including background or inessential detail".

Encapsulation means that an object contains both a state and the operations that can be performed on that state. An object contains everything it needs, and by doing it this way, no other object need ever be aware of the object's internal structure (Graham, 1991). Data can be obtained from the object by sending a message to the object.

With encapsulation, users do not have to bother how an object performs its function and, more important, what it can and cannot do. They can be confident that adding an object to an existing code will not cause unpredictable things to happen to different parts of the program. They can also try out the objects' behaviour on the spot to determine whether they have chosen the right objects before incorporating them in their program.

Inheritance is a mechanism that allows a new class (subclass) to inherit characteristics from an existing class (superclass). In this scheme, classes higher up the structure represent more general abstractions, and classes lower down the structure describe specializations of the former.

Inheritance promotes software reuse. Reusing rather than reinventing code speeds up the development of large applications. For example a `Cat` class can be created by inheriting all the methods from `Animal` class and adding all the methods that are needed to make its definition complete; there is no need to duplicate the methods contained within the class `Animal`.

A conventional programmer might reuse code by copying and editing, but an object-oriented programmer can accomplish this automatically by creating a subclass and overriding some of its methods. A method in a descendant can override a method in a ancestor class simply by creating a new method of the same name.

Furthermore most object-oriented languages provides a large library of classes covering basic abstractions such as collections and dictionaries, as well as graphical classes which include interface components such as windows and scrollbars. Class libraries may also provide frameworks such as the Model-View-Controller framework for building graphical user interfaces, as discussed in section 6.4.

Inheritance also eases maintenance because code shared by several classes in an inheritance chain is found in one place only. Hence, any change needs to be carried out at that particular place only, and this change is automatically propagated to all the subclasses (Cox, 1984).

Polymorphism is "the ability of different objects to respond differently to the same message" (Thomas, 1989). Sending the draw message to a `Circle` invokes its draw method; sending the same message to a `Square` invokes a different method. Without polymorphism, separate draw functions such as *drawCircle* and *drawSquare* are required for each type of object.

6.2 SMALLTALK-80

Smalltalk was developed in the early 1970s by Alan Kay and his colleagues at Xerox Palo Alto Research Centre. Smalltalk evolved from a simulation language, Simula and it became the first language to introduce the object-oriented concept in the field of computing. The language went through three major versions: Smalltalk-72, Smalltalk-76 and Smalltalk-80. Smalltalk has become a model for many modern object-oriented programming languages and yet still retains its popularity, partly due to the robustness of its large class library (Bourne, 1992).

Smalltalk was selected for the development of the prototype. Object-oriented systems in general and Smalltalk in particular offer several advantages to developers. Smalltalk is not just a language, but a complete programming environment. Users work within an object-oriented framework of browsing the class library, deleting objects, experimenting and testing them out and incorporating them into their applications. More importantly, Smalltalk forces users to adopt the object-oriented approach (Thomas, 1989).

As far as reusability is concerned, the different Smalltalk systems can regarded better than any other object-oriented languages. Smalltalk provides an extensive class library which permits rapid creation of new classes by inheritance. Bourne (1992) compares the number of classes supported by four commercial object-oriented languages, Smalltalk-80, Smalltalk/V, Objective-C and C++. The comparison of these languages is shown in table 6.1. Objective-C allows purchase of additional classes to support user interface creation.

Name	**Number of Classes**
Smalltalk-80 (release 4)	~ 330
Smalltalk/V	~ 175
Objective-C additional classes	> 20 > 60
C++	0

Table 6.1 : Classes supported by Object-Oriented Languages(Bourne, 1992).

Another outstanding feature of Smalltalk-80 is portability. Code generated on one platform will run without any changes on different platforms. Such a capability is achieved because the system is composed of two main components, the image, which consists of objects that make up the development systems and layered applications, and the virtual machine which executes the image on a given platform (Udell, 1990).

One of the essential requirements of the development of user interfaces is the massive use of memory. It is noted that some languages, like Lisp and Smalltalk manage memory more efficiently. Both languages support automatic garbage collection, i.e, both languages reclaim any unused memory space by removing objects or pointers that are no longer used. Smalltalk Release 4 implements an incremental garbage collector which runs as a background process within the image itself (Cook, 1991).

Compared to other object-oriented languages, Smalltalk's performance is relatively slow (Winblad, Edwards & King, 1990). Execution speed was once an important factor for choosing between languages. However, this factor has become less significant as computer performance has increased and the cost of expanding computational power has decreased.

Smalltalk is not easy to learn. This statement is based on the author's experience and it is in agreement with other Smalltalk users (Diederick & Milton, 1987; Nielsen & Richard, 1990; Hix & Hartson, 1993). Hix & Hartson (1993) points out that it takes about 3 months for programmers who are familiar with a procedural approach to become proficient in Smalltalk. Object-oriented methodology is completely different from the conventional approach with which most programmers are familiar. Learning object-oriented languages, such as Smalltalk is like trying to adapt to a new culture (Che Mee, 1990).

Browsing through the Smalltalk classes can be a difficult and frustrating task (LaLonde *etc al.*, 1985). The operations associated with a class are distributed throughout a hierarchy of classes. The system browser only displays operations which are relevant to a particular class. In most cases, this forms only a small subset of the operations for that class. Finding the whole set involves browsing through each individual class in the superclass hierarchy.

An enormous amount of time has to be spent investigating the class libraries and extensive experimentation may be required to understand how these classes work. Despite the difficulties, it is still worthwhile putting the effort to master the language. After the learning period, the flexibility and expressive power of the language as well as a large class library of predefined classes make it easier and faster to develop most applications.

6.3 DEPENDENCY MECHANISM

The world is full of objects which are dependent on other objects. For example, a light is dependent on a switch. When the switch is turned on, the light is on and when the switch is turned off, the light is off. A similar relation can be created in Smalltalk. Any object can be made dependent on another object by using the dependency mechanism.

The followings are properties of every object in Smalltalk relevant to the dependency mechanism: (ParcPlace System, 1992).

- Any object is allowed to register itself as a dependant of another object.
- Any object may send the message *changed* to itself. The message is to notify that the object has changed. When the object sends *changed* to itself, each of its dependants automatically receives an *update* message.

The dependants have to implement their own *update:* methods. They may redisplay themselves or take other updating action.

This mechanism is particularly important for the implementation of the Model-View-Controller (MVC) (see 6.4). In the MVC methodology, one or more views is registered as a dependant of a model. Whenever the state of the model changes, the model sends a *changed* message to itself. The view (and any other objects that are registered as dependants of the model) automatically receives the message *update*. The default method for the update message in the Object class is to do nothing, but most views have a protocol to redisplay themselves whenever they receive an *update* message.

Figure 6.1 shows a dependency mechanism between a model and its views. The model holds a collection of dependants, that is objects which are dependent on it. The dependants are view1 and view2. When the model sends change to itself, the dependency mechanism notifies the views to update themselves. In many cases, a view has to obtain the latest information about the model before updating itself accordingly. A view is usually associated with one model, but a model can have many dependant views.

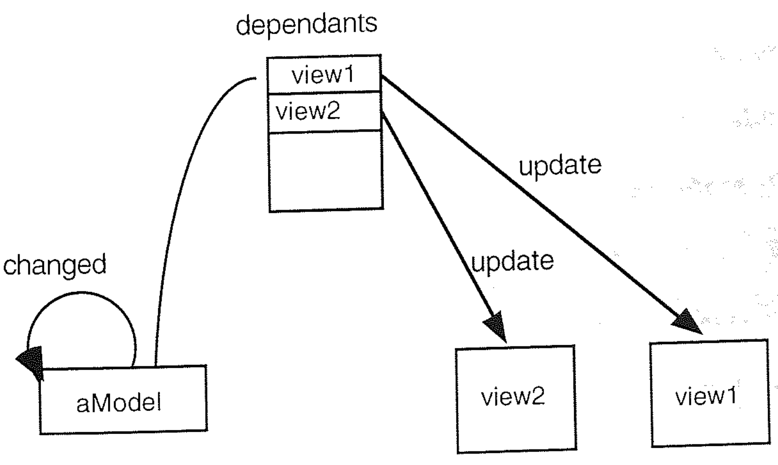


Figure 6.1 : Dependency Mechanism

A variety of changed messages are supplied by the dependency mechanism. The variants are as follows:

aModel *changed*
aModel *changed*: **aSymbol**
aModel *changed*: **aSymbol** *with*: **Parameter**

The argument 'aSymbol' allows the model to inform which aspect has been *changed* and the corresponding view can take an appropriate action by examining the 'aSymbol'. The model also may wish to provide an extra information and send it as a parameter with the changed message. As an example, the following changed message is sent by a model to its dependant (a window):

self *changed*: #windowLabel *with*: ' TestWindow '

Sending the message above causes the window to update itself by changing its label to 'TestWindow'

6.4 MODEL-VIEW-CONTROLLER (MVC) METHODOLOGY

Interactive applications usually consist of two parts:

1. The information model, which handles data storage and processing.
2. The user interface which handles input and output.

Separation of the user interface from the application has several advantages (Dodani, Hughes & Moshell, 1989). First, the user interface can be further subdivided into components that can be glued together. Developers may use these components without a detailed understanding of the underlying implementation. Second, the interface can be rapidly modified to be reused in other applications. Third, the interface can be altered without having any adverse effect on the application codes. Finally, the interface can be developed in an iterative manner, where successive prototypes are produced until a satisfactory one is completed.

The Smalltalk interface paradigm implements a similar principle of separating the application model from the user interface. It further separates the user interface into two components, i.e. the view and controller. The functions of the three components are as follows:

- The model deals with the underlying functionality.
- The view presents the model on the display screen.
- The controller manages the interaction of the system with the user.

The combination of the three components is called the Model-View-Controller methodology or MVC.

Communication among the components of MVC is shown in figure 6.2. The communication is handled by sending messages and by the dependency mechanism of Smalltalk. The view and controller are tightly coupled. The view stores an instance variable that points to its controller and vice versa. The view and the controller also store an instance variable pointing to the model. This means that both the view and the controller may directly access information about each other and the model.

The model is slightly isolated in the MVC triad. It neither has a pointer to its view nor its controller. Hence, the model is incapable of displaying itself or interacting with the user. Using the dependency mechanism, the model must notify changes to its view and controller whenever one or more of its aspects are changed. The view and controller respond by querying the model and update themselves to reflect the change. The model may be associated with more than one view-controller pair so that different aspects of the information can be displayed.

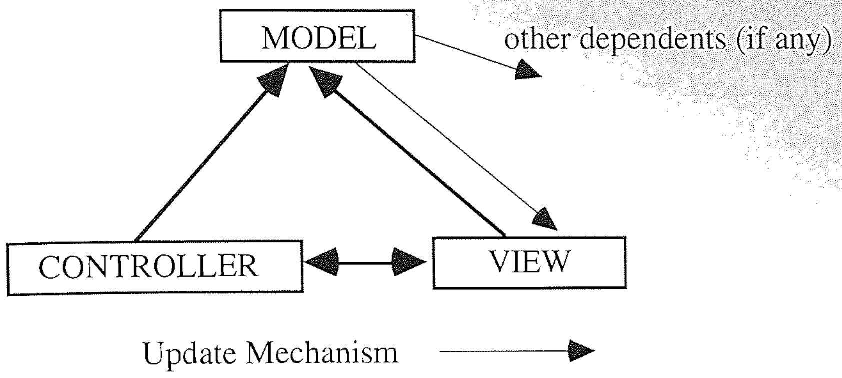


Figure 6.2 : Model-View-Controller

The Smalltalk MVC methodology is a powerful tool that allows an easy and systematic development of an interactive application. Most of the components required to build an application based on this methodology are available in the Smalltalk library. The same methodology is also used by the Smalltalk itself for implementing the tools of the programming environment, for example the System Browser. The System Browser consists of five views, and its information model is the library of Smalltalk classes.

6.4.1 MODEL

A `Model` can be a simple object that holds an integer or complex applications such as a word processor or an electronic mail. An information model is usually created as a subclass of class `Model`. The class `Model` contains an instance variable dependents that holds all the dependants of a model and it has the machinery for notifying its dependants when it is changed.

The concept of an adaptor was introduced to make views adaptable to multiple models (ParcPlace Systems, 1992). An adaptor can be thought as a translator that provides a flexible communicaton between a view and a model. There are two classes which serve as adaptors, i.e. `ValueHolder` and `PluggableAdaptor`. Both classes are inherited from the class `Model`, and their parent class is `ValueModel`. The hierarchy is shown in figure 6.3.

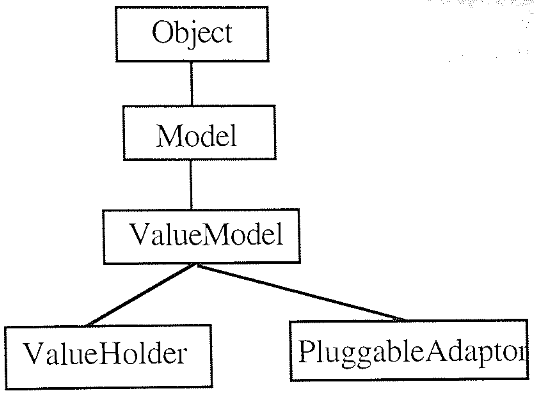


Figure 6.3 : ValueHolder and PluggableAdaptor

A `ValueHolder` acts as translator for simple objects that do not have the mechanism to behave like a model. The objects include numbers, strings and booleans. They must be placed in `ValueHolder` before participating in the MVC triad. The `ValueHolder` then assumes the responsibilty of registering dependent objects and notifies any change in itself to its dependants.

For example, suppose a simple application needs to display a counter in one of its views. An integer can be the model for the application. Creating a new class and developing the integer acting as the model requires writing extra code. A more simple solution is found by placing the Integer in a `ValueHolder`. The `ValueHolder` has the capability of accessing the value of the integer and acting as a model for the view. Figure 6.4 shows a `ValueHolder` that holds an integer and assumes the task of informing the view when the value is changed.

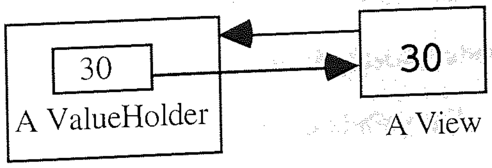


Figure 6.4 : ValueHolder

A `PluggableAdaptor` acts as a translator for a complicated model which has a different vocabulary from the view. For example, the adaptor can be used as a link between a button and the main information model. The use of `PluggableAdaptor` class is shown in the listing in Figure 6.5.

```
button := LabelledBooleanView new.
button beTrigger.
button controller beTriggerOnUp.
button beVisual: 'Search' asComposedText.

button model: (
          (PluggableAdaptor on: aModel)
                    getBlock: [:model | false]
" =====>"           putBlock: [:model :value | model doSearch ]
                    updateBlock: [:model :value :parameter | false]).
```

Figure 6.5: PluggableAdaptor (Program Code)

A temporary variable button is used to represent a button. The first group of expressions creates the button and assigns its attributes. The second group of expressions links the button to the main information model, 'aModel'. The group consists of three expressions:

- The *getBlock* - to acquire the current state of the model. The information may be used to adapt the appearance of the button depending on the status of the model.

- The *putBlock* - the expression is executed when users click the button. Execution of this expression causes 'doSearch' to be sent to the model.

- The *updateBlock* - the block provides the option to include the following arguments: the model, the aspect of the model that has changed and the extra parameter in the changed message, if any. In the listing above, the value returned is always false. This indicates that there is no updating to be done.

Figure 6.6 shows the communication between the button and the information model. The button is an instance of `LabelledBooleanView`. As its name suggests, it returns true or false. The method *beTriggerOnUp* defines the behaviour of the button. A boolean true is sent when users click the button (press and release without moving the pointer away). In the given example, `PluggableAdaptor` can be regarded as a black box that accepts a boolean true as input. Upon receiving the value true, the `PluggableAdaptor` sends the method 'doSearch' to the information model.

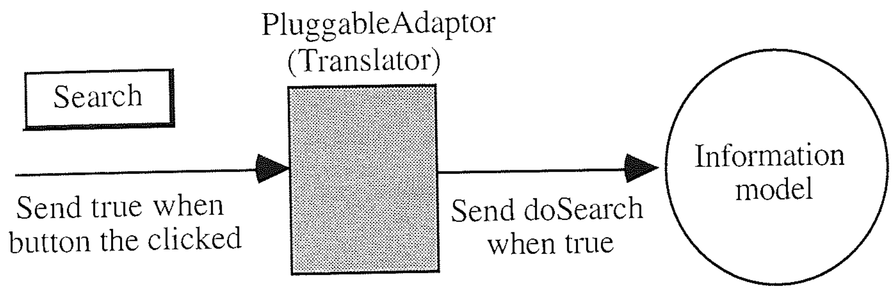


Figure 6.6 : Application of PluggableAdaptor

6.4.2 VIEW

Views are what users see on the screen within a `ScheduledWindow`. A `ScheduledWindow` is a subclass of Window which provides a controller that allows users to move, resize and close the window. A `ScheduledWindow` usually holds a `VisualComponent`, the abstract class that defines the common behaviour of views and other displayable objects.

VisualComponent can be classified into:

- Visible type - such as view, image and text
- Supporting type - this has 2 components
    1) A `CompositePart` and its subclasses hold a collection of other visual components.
    2) A `Wrapper` and its subclasses which provide services such as placement and bordering to a visual component.

The relation between between the different subclasses of `VisualComponent` is shown in figure 6.7.

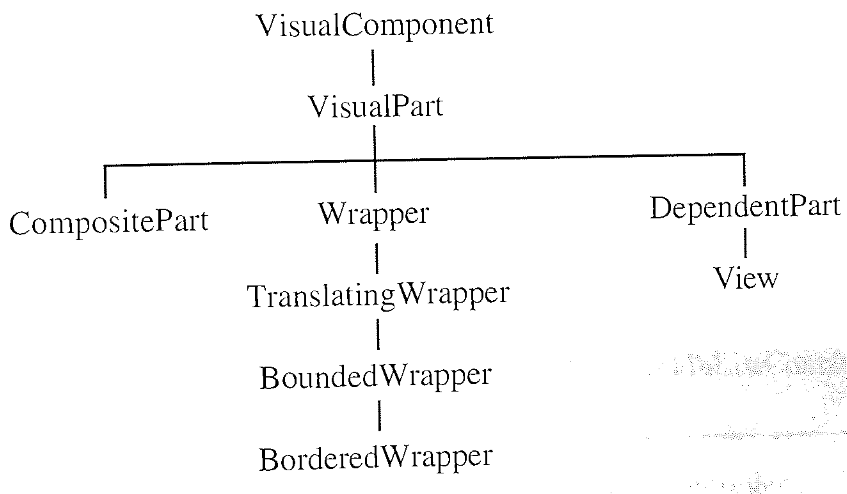


Figure 6.7 : Visual Components and its Subclasses

Figure 6.8 shows the way different visual components are layered on top of a window in an application. A window containing two views would require a composite that contains two wrappers, each of which holds one of the views.

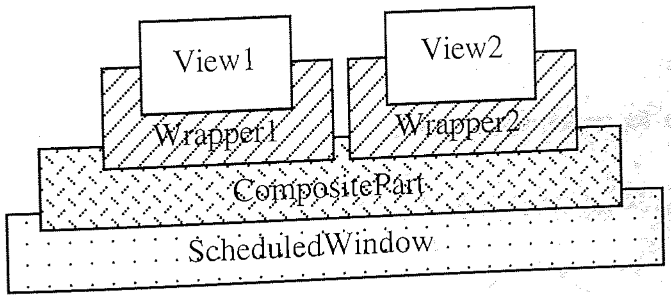


Figure 6.8 : VisualComponents and ScheduledWindow

Every view has to be enclosed in a `Wrapper`. A `Wrapper` is a mediator that forwards messages from its container (`CompositePart`) to its component (`View`) and vice versa. The `Wrapper` and its subclasses also provide the needed bookkeeping information such as translation, clipping and bordering. The primary role of a `CompositePart` is to pass messages to the correct subcomponent as users move the mouse pointer from one subcomponent to another in a window.

Views are numerous in the Smalltalk class library and they can be divided into four categories. Table 6.2 shows examples of the four categories of views and their respective controllers.

Type	**Name of the View**	**Controller**
Text	CodeView (text editing)	CodeController
List	SelectionInListView (select an item from a list)	SelectionInListController
Dialog	DialogView Dialogs (e.g.fill-in the blank)	DialogController
Graphic	BarChartView (supplied in Tutorial)	No controller

Table 6.2 : Categories of Views

A view must be redisplayed after the following events:

- the model updates - the view has to respond accordingly.
- Window damage repair for example, the window is refreshed or an overlapping window is moved.

A view must implement two methods in order to handle either kind of event. The methods are:

- *displayOn*: - this is a display driver that allows the view to redisplay itself. It should always consult the current state of the model.
- *update*: - This method should send an *invalidate* message to itself. This is an inherited method which ultimately invokes the view's *displayOn:* method.

Implementing *displayOn:* and sending the *self invalidate* message in the *update:* method allows a view to respond to any redisplaying situations in a unified way. The *invalidate* method will manage the redisplay especially when both events occur simultaneously.

6.4.3 CONTROLLER

The controller accepts input from the user, normally via the mouse or the keyboard and informs the view and the model of pertinent user actions.

The host operating system passes control to `ControlManager` when a Smalltalk window is activated. Control in this context refers to ownership of the user input. `ControlManager` then passes control to the window which contains the pointer. The window takes control if the <window> button is pressed, else it offers control to its component containing the pointer. That particular component may accept control depending on the design of its controller. For examples, the controller for a button or a check box usually accepts control only when the <select> button is pressed.

ScheduledController is a global variable which is an instance of `ControlManager`. It is the function of this variable to pass control to the active window's controller. The controller for the window is the `StandardSystemController`. This controller requests the window to check whether any of its views wants control. Each view then asks its controller *isControlWanted*. The controller that responds true is sent a startUp message by the `ControlManager`.

Once a message *startUp* is sent to a controller, the controller initializes itself, goes into a control loop, and finally performs a termination routine. The process is shown in figure 6.9. Initialization is carried out by a *controlInitialize* method. Many controllers do nothing in response to this message. This method can be reimplemented in the controllers class so that some special action can be carried out such as changing the pointer image. For example, in a text editing view, it is more appropriate to change the normal pointer image to a text editing image pointer.

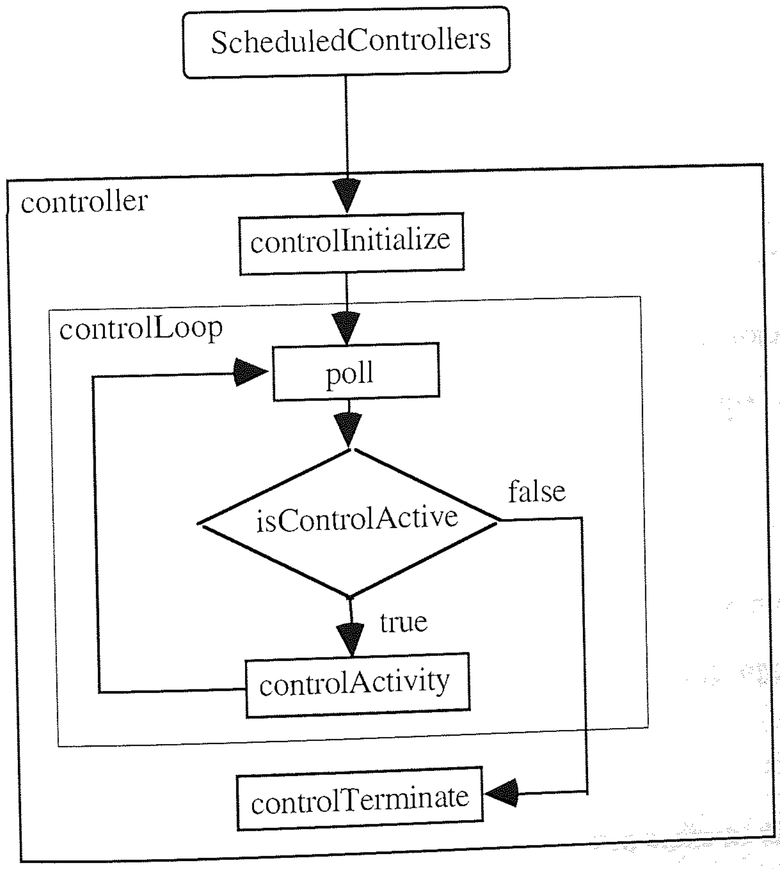


Figure 6.9 : Controller Activities

In the controlLoop, polling is carried out to see if there has been any activity. If there is no activity for a certain period, a flag(semaphore) is set and the polling can stay idle until a signal is received indicating that there is activity. When activity resumes, a controller verifies that the condition for maintaining control (*isControlActive*) is still true. If so, it sends *controlActivity* to itself, but if the test fails, control reverts to the ScheduledController, which resumes polling to find a new control receiver.

The *controlActivity* is the method that decides how the controller should respond to the user input. The *controlActivity* method for the `StandardSystemController` is to respond to the <window> button. When this button is pressed, the controller prompts the standard pop-up menu for a window. The *controlActivity* method for the `CodeController` (text editing purposes) is to respond to the <select> and <operate> mouse buttons and the input from the keyboard.

Most controllers gives up control when the pointer passes out of its view. However, modification can be made to the *isControlActive* method to change that policy. For example, the `DialogController` refuses to yield control until an answer to the dialogue is accepted by its model.

Understanding the controller mechanism is essential if the behaviour of the controller is to be customized. The controller for the button of the prototype is described in the next section.

Since ScheduledControllers is a global variable, it is used by the prototype to close a window and to find out the status of different windows, e.g. open or close. The messages below shows how ScheduledControllers

a) is used to close the current window (the window which contains the pointer).

ScheduleduledControllers *activeController close*.

b) is used to close the System Browser.

ScheduleduledControllers *scheduledControllers do*:
[:i | (i view label) = ' System Browser' ifTrue: [i close]].

6.5 IMPLEMENTATION OF MODEL-VIEW-CONTROLLER

In the prototype, the different stages involved in establishing a connection to BIDS is displayed on a status window. Figure 6.10 shows the communication between the communication process and the display process. The display process is represented by the status window. Both processes run simultaneously.

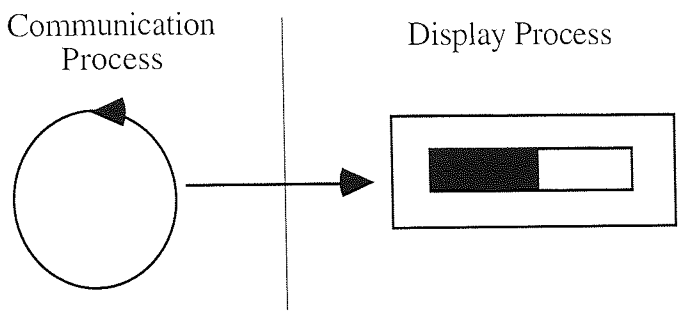


Figure 6.10 : Displaying the Status of the Comminucation

The communication process has a dual function:

- To read information from the communication port. The reading from the port must be properly synchronised so that all the information is captured.
- To act as a model for the status window. It has to inform the window about the different stages of connectivity by using the MVC mechanism

By default, a window updates itself when its receives a *checkForEvents* from its controller. The standard controller sends that message each time it is polled for activity. In the case of the status window, the indicator does not update itself immediately when it receives an update message. Since, more than one process is competing for the processor, there is a significant delay between the time the model changes and the view is updated.

An immediate update of the status window can be achieved in 2 ways:

1) Forcing the view (of the status window) to update by sending the message

    <u>ScheduledControllers</u> *checkForEvent*

   every time *self changed* message is sent by the model.

2) By changing the default value in the *invalidateRectangle* method. The method is in the VisualPart class. By default, Smalltalk implements *lazy repair damage,* i.e., delaying any update until the processor is free. Changing the parameter for repairNow to true enables the message to override the implementation.

    self *invalidateRectangle*: aRectangle *repairNow*: true

6.6 IMPLEMENTATION OF THE PROTOTYPE

The prototype comprises of two main parts:

- The communication part - manages the communication between the Smalltalk image and networked services.
- The user interface part - manages user interaction and displays of output. It translate users' actions into commands which are understandable to networked services and transforms the output from the services before it is displayed.

The implementation of the prototype can be described two phases:

1. <u>Establishing the communication with network services</u>

   This first phase in the implementation of the prototype is to ensure that Smalltalk is able to communicate with the outside services which are available through the computer network. These services are the model for the prototype.

   The first step after the connection is established is to start a preliminary test on the model. Since, at this stage the user interface part is not yet constructed, existing facilities provided by Smalltalk are used to simulate the input and the output of the model. The Smalltalk `DialogView` is used to create a dialogue window. The window is prompted to accept input for the model and the result is displayed on the Transcript Window. During the initial testing phase, the procedure to establish the connection, such as entering the login name and password is carried out manually by using the dialogue window. Interactions to the services are carried out by entering appropriate commands and the results are printed on the Transcript Window.

2) Developing the user interface

The development of the user interface is based on the MVC methodology. Most of the components which are required for the interaction, such as windows, buttons and pop-up menus are available in Smalltalk class library. Initially, a simple interface that comprises of a few buttons to represent commands and a main display is constructed on a window. The buttons are then linked to appropriate commands and the output from the networks services is captured as a variable and displayed on the main display.

The interface is further developed by including more buttons and incorporating pop-up menus so that more commands can be attached to the interface. Different types of views to display different types of information are also incorporated. These views are then tested to make sure that they produce the expected results.

Other aspects of the user interface which help users to interact with the system are finally included. These include changing the behaviour of the mouse pointer image (e.g. changing it to waiting mode during the processing time) and adding system messages and online-help facilities.

The implementation of the prototype is based on incremental problem-solving. Pinson &Wiener (1988) define incremental problem solving as "adding incremental capability to an existing base of workable software (the image)". This approach is well supported by Smalltalk.

Incremental problem solving includes the following activities:

- finding the right class and reusing it rather than reinventing new codes.
- creating new classes to reduce complexity.
- implementing an inheritance hierarchy in a way that facilitates the addition of new functionality.
- improving interaction between objects.

Three classes, namely `EMail`, `Bids` and `Editor` are created to represent the three applications of the prototype. They are declared as subclasses of `Model` (see figure 6.11). The `Model` class is selected because it has the machinery for notifying dependent objects (views and controllers) when it is changed. `Model` holds the collection of dependants in an instant variable.

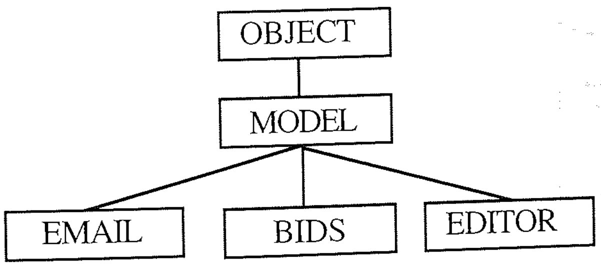


Figure 6.11 : Creation of New Classes

The dependency mechanism is also provided by the `Object` class. `Object` stores dependants in a global dictionary. This provides universal dependency relation for any object, but with the advantage that dependants cannot be removed by the garbage collector.

6.6.1 IMPLEMENTATION OF ELECTRONIC MAIL

`Email` is the parent class that provides the following functions:

- Maintains the information related to electronic mail application. These include the list of folders and electronic mail messages.
- Runs a process that constantly check for any new mail.
- Delegates some of electronic mail functions to its subclasses.

`EMail` stores some of its information in class variables. This allows the class and all its subclasses to access and modify the content of the variables. Figure 6.12 shows the `EMail` class, its class variables and subclasses.

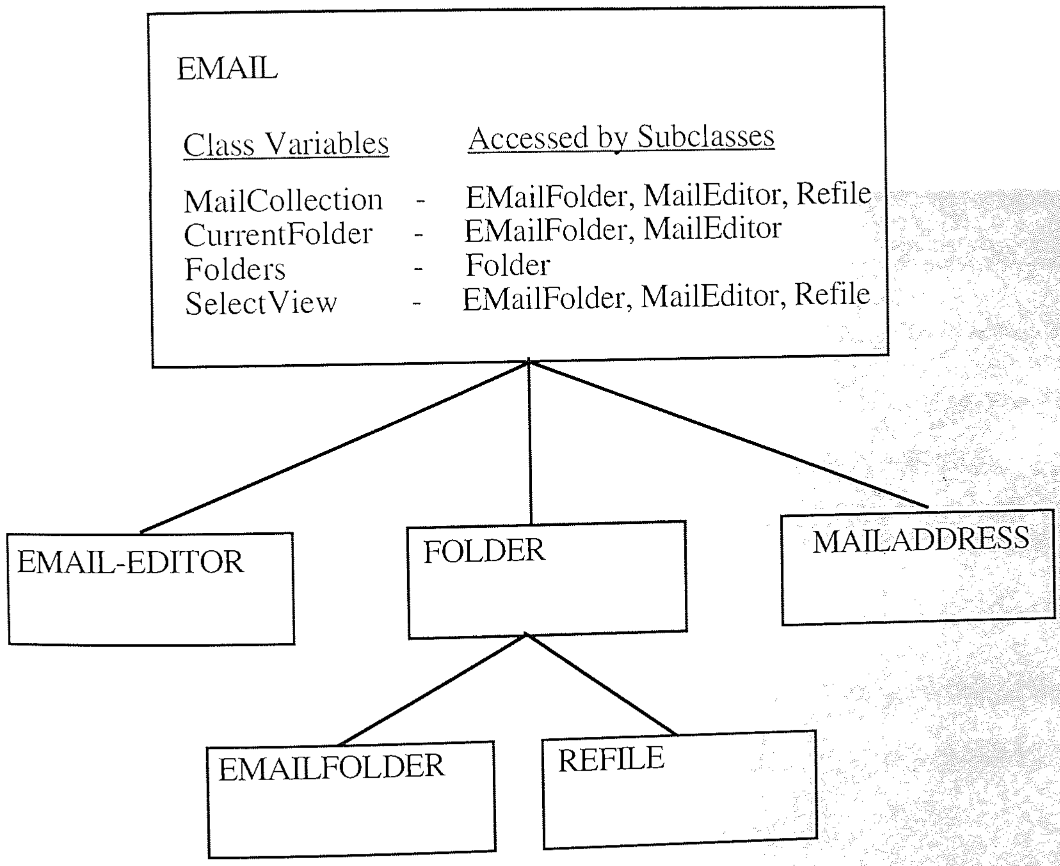

EMAIL
Class Variables
Accessed by Subclasses
MailCollection - EMailFolder, MailEditor, Refile
CurrentFolder - EMailFolder, MailEditor
Folders - Folder
SelectView - EMailFolder, MailEditor, Refile
EMAIL-EDITOR
FOLDER
MAILADDRESS
EMAILFOLDER
REFILE

Figure 6.12 : EMail and its Subclasses

The followings are `Email` subclasses and methods provided by the `Email` class to access them:

- `EMailEditor` invoked by the *doSend* and *doReply* methods.
- `EMailFolder` invoked by the *doFolder* method.
- `Refile` invoked by the *refile* method.
- `MailAddress` invoked by the *seeAddress* method.

This methods are executed whenever users execute controls, i.e. buttons or a pop-up menu on the electronic mail primary window.

Table 6.3 describes the EMail class variables:

Class Variables	**Class Types**	**Descriptions**
MailCollection	Dictionary	Hold a list of all email messages in different folders. Folder is the key of the association.
Folders	String	Holds a list of folders
CurrentFolders	String	Holds the selected folder
SelectView	SelectionInListView	Displays the summary of messages
FolderView	DisplayTextView	Displays the current folder

Table 6.3 : EMail Class Variables

SelectView and FolderView are declared as class variables so that they can be updated from other windows (objects). SelectView also holds information needed by the subclasses, such as the selected message.

`EMailEditor` provides the mail editing capability. This class has to access information from its parent class when *doReply* is invoked. It has to know which message to reply to, the sender and the content of the message.

`MailAddress` provides the facilities to store and maintain aliases. Aliases are stored in a class variable PersonalData. The variable is of class type `Dictionary`. `EmailEditor` needs to communicate with `MailAddress` if the alias is specified in the message header. Specific methods have to be sent to `MailAddress` to access aliases from the class variable .

The `Folder` class is an abstract class. The primary function provided by the class is to browse through the hierarchy of email folders. Two classes, namely `EMailFolder` and `Refile` are defined as subclasses of `Folder`. Both subclasses need to browse through the email folders. There are some differences in terms of the functionality provided by the two subclasses. `EMailFolder` provides functions which are related to folder operations. These include activating, adding, deleting a folder. In the case of `Refile`, users need a folder browser to specify target folders in order to refile electronic mail messages.

`EMail` and its subclass (`EMailEditor`) communicate with the UNIX mail facility by sending *cshOne*: or *cshBullet*: messages to `UnixProcess`. The parameters of the messages are valid UNIX commands. The difference between the two messages is that only *cshOne*: returns a result.

6.6.2 IMPLEMENTATION OF BIDS

The class `Bids` provides the following functions:

- Maintains the information of BIDS application such as the year and database the application is connected.
- Manages communicatons with networked services.
- Searches articles by title, author and journal in BIDS databases.
- Delegates some of the tasks to its subclass, `BidsTasks`.

Figure 6.13 shows `Bids` and its subclass, `BidsTasks`.

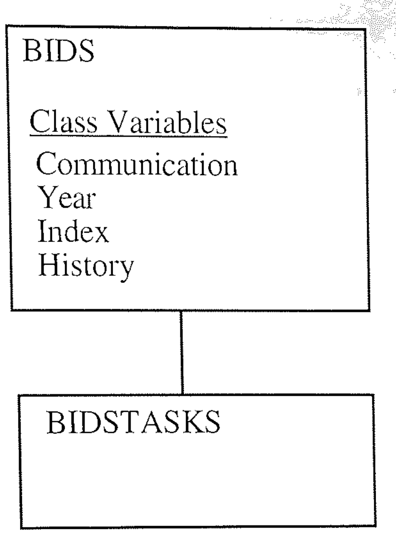


Figure 6.13 : Bids and its Subclass

The followings are methods and their respective functions provided by `Bids`. These methods cause the invocation of `BidsTasks`.

• seeHistory	- reviews a list of searches made.
• doSave	- saves searches on the hard disks.
• doRetrieve	- retrieves saved searches.
• doPreferences	- alters connection to BIDS such as the effective year of search.

Table 6.4 describes the `Bids` class variables:

Class Variables	**Class Types**	**Descriptions**
Communication	Terminal	Provides communications between `Bids` and networked services
Year	Integer	Holds the effective year of search
History	OrderedCollection	Holds a collection of searches
Index	ValueHolder	Holds integer to represent the selected database

Table 6.4 : Bids Class Variables

The implementation is carried out using Smalltalk Release 4, but this version does not provide any class that manages interactive communication between the Smalltalk environment and external programs. Such a facility is provided by the class `Terminal` in Smalltalk Release 2.5. Hence, this class was filed out from the Smalltalk image and incorporated into Smalltalk Release 4.

Communication is an instance of `Terminal`. It is created when the message *getCshTerminal* is sent to the class `Terminal`. The message invoke a fork system call and it executes a UNIX C Shell program. This means that Communication is a child process which runs concurrently with the user interface.

Figure 6.14 shows the communication between `Bids` and `Terminal`. `Terminal` provides two methods for receiving and sending messages to networked services. The methods are:

- sendAll: (arg) -accepts argument(user input) and sends it to networked services.

- getData - The information received from the networked service is stored in one of the instance variables of the communication channel. The method returns the information to the sender (`Bids`).

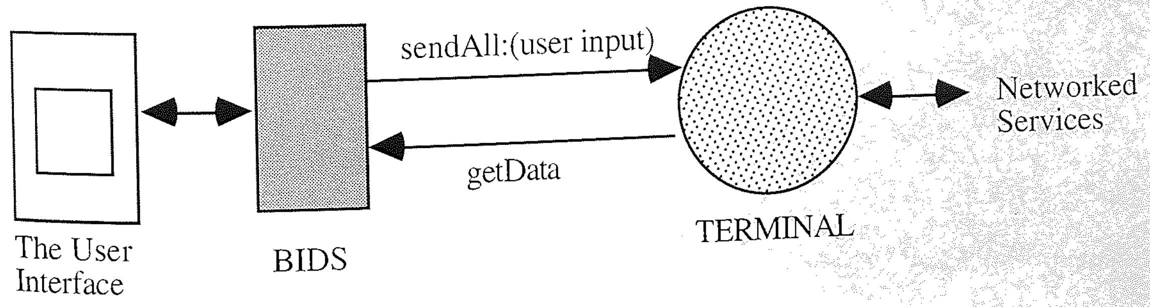


Figure 6.14 : Communication between Bids and Terminal

One of the functions of `Bids` is to manage communication with the networked services. For example, `Bids` will not allow data to be sent to `Terminal` when it is in the process of receiving information from networked services. This is done by disabling command buttons and changing their colour to grey.

6.6.3 REUSING AND REFINING EXISTING CODES

Reusability of software components is one of the major benefits of structured and object-oriented programming. Existing codes can be reused in the construction of applications rather than writting program from scratch. New classes may also be created by refining some of the existing classes. The amount of programming by refinement is proportional to the amount of difference between the desired behaviour and the behaviour of existing classes that are being refined (Goldberg & Pope, 1989). In mature object-oriented systems such as Smalltalk, the amount is quite small.

Smalltalk provides an extensive class library which permits rapid creation of new classes by inheritance. Figure 6.15 shows the class `View` and some of its subclasses in the Smalltalk class library. Only those classes which are related to the creation of views on the window for the electronic mail and BIDS aplications are displayed. Classes such as `DisplayTextView` and `ReadOnlyView` are created using the inheritance mechanism.

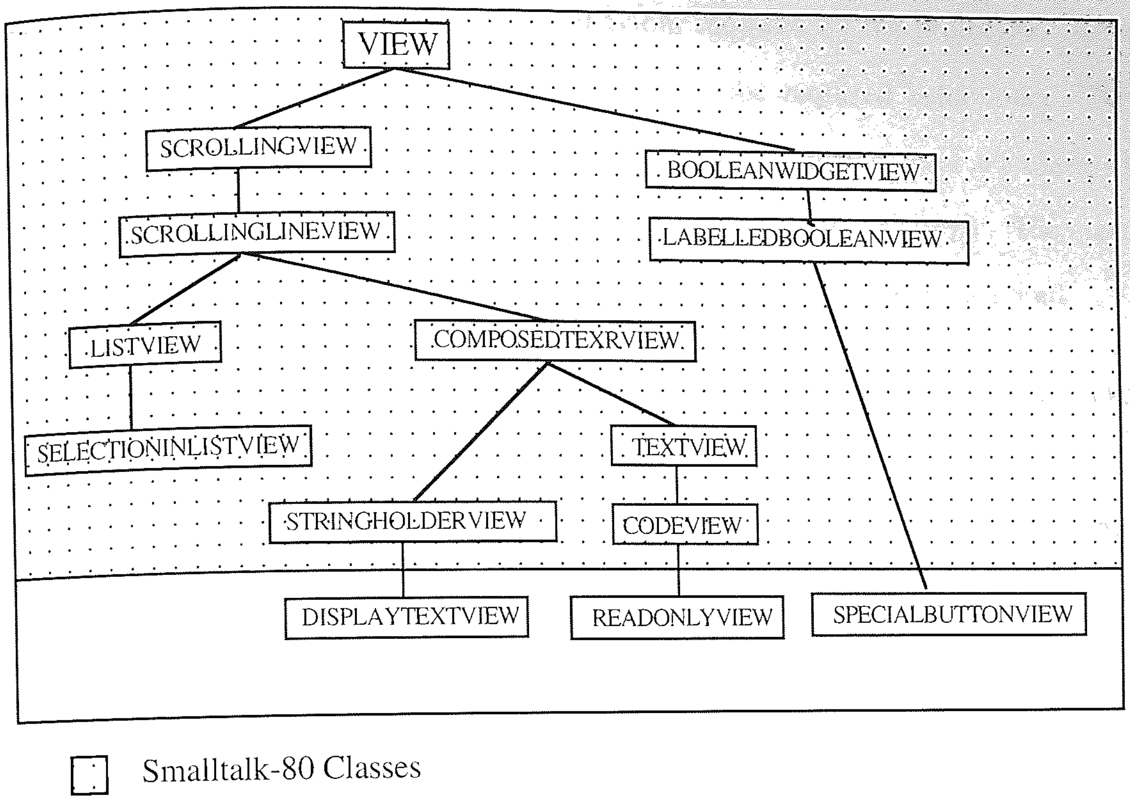


Figure 6.15 : Partial Subclasses of the Class View

Some existing classes are reused for the construction of the prototype. For example, the class `SelectionInListView` is used by the Scroll List View of the prototype. `CodeView` provides text-editing capabilities and it is used by Smalltalk as one of the views in the System Browser. This view is reused by the File Editor and Mail Editor Window of the prototype. Examples of capabilities provided by the view are copy, cut and paste. Large amounts of code would have to be written if such views were not available for reuse.

The creation of any view automatically defines a link to a model or an aspect of a model. For example, in the electronic mail application, the model for the Scroll List View (an instance of `SelectionInListView`) is the list of messages in the current folder. If the aspect of the model is defined as #messageList, the view would check the new list of messages in the current folder whenever the model sends self changed:#messageList.

Existing views may not provide all the behaviour required by the prototype. New subclass are created to provide the prototype with the required behaviours. For example, a new class `SpecialButtonView` is created as a subclass of `LabelledBooleanView` (its controller is `WidgetController`). The class `LabelledBooleanView` (and its controller) provide the following behaviour:

- Perform the specified command on clicking using the <select> mouse button. The command is specified during the creation of the button.
- The colour of the button darkens when pressed (using the <select> mouse button) and returns to its normal appearance when released.

The controller for `SpecialButtonView` is `SpecialButtonController`, which is a subclass of `WidgetController`. The view and controller pair inherit all the behaviour of their parents. In addition, they also provide the following capabilities:

- Transform the pointer into a pointing finger image when the pointer is inside the button area.
- Display a description of the command performed by the button in the note view of the window.
- Open a help window when clicked using the <adjust> mouse button.
- Make the button inactive when the command is not available. During the inactive period, the label of the button is greyed. This facility is used by the ReadMail Button to indicate that there is no new mail.

A conventional programmer might reuse code by copying and editing, but an object-oriented programmer can accomplish this automatically by creating a subclass and overriding some of its methods. A method in a descendent can override a method in an ancestor class simply by creating a new method of the same name. For example, `SpecialButtonController` overrides the controlLoopBody of its parent by implementing a new controlLoopBody. The new method adds code that constantly checks whether the <operate> mouse button is pressed.

Similarly, `ReadOnlyController` supresses the keyboard input by reimplementing its own readKeyboard method. The method ignores any input from the keyboard. `DisplayTextView` supresses the highlighting of selected text by reimplementing the selectionShowing method and assigning selectionShowing variable to false.

Each of these classes has its own controller. For example, the controller for the class `View` is `Controller` and the controller for the class `CodeView` is `CodeController`. The method defaultControllerClass is implemented by each class to specify its controller pair. If the method is not implemented, the view will assume the controller of its superclass.

CHAPTER SEVEN

EVALUATION OF THE PROTOTYPE

7.0 INTRODUCTION

Evaluation of user interfaces using real users provides some performance indicator on how the design is likely to perform in everyday usage (Johnson, 1992). Evaluation also provides invaluable information on how users use the system and what their problems are with the interface being tested.

The main activities for the evaluation of the prototype were:

- Selecting test users.
- Conducting the evaluation session. During the session, users were requested to run test tasks and then answer the questionnaire. User details forms (see appendix A2), an introduction to the procedure of the evaluation (see appendix A1) and a list of test tasks (see appendix A3) were given to test users at the beginning of the session. The questionnaire (see appendix A5) was given after users had completed the test tasks.
- Analysing the result of the evaluation.

This chapter discusses the results of the evaluation and analyses the rating of the prototype. Recommendations are given based on the results of the evaluation.

7.1 RESULTS OF THE EVALUATION

Thirty test users participated in the evaluation of the prototype. The users had different computer backgrounds. Some of them were familiar with the Macintosh and IBM PC only. Others were familiar with Sun workstation. Generally, users who have some experience using Sun are familiar with the IBM PC or Macintosh, or both of them.

The majority of test users had some kind of experience with word processing. Approximately eighty percent of the test users had used electronic mail before. Generally, test users who were from departments such as English and Pharmacy were not familiar with electronic mail. Twenty percent of the users mentioned that the facility was not useful for them because most of their colleagues were not using it.

There were not many users who had used BIDS before. The study showed that only researchers and some computer technicians had experience using BIDS and they used the service occasionally. Users who never used BIDS said that they had heard about the service but they simply had not found time to investigate what was BIDS all about. Others mentioned that they disliked BIDS because it was hard to use.

Generally, users do not use any service until they know its advantages or if there is an immediate need. In the case of word processing, users had to use it because they were expected to do assignments using computer word processing. A similar finding is discussed by Thomas and Kellogg (1989). They point out that a motivational factor plays an important role in the real world. The convenience of having access to cash 24 hours a day has led thousands of customers to master a variety of teller machines, often with inconsistent user interfaces. On the other hand, setting the clock on the VCR, a task of comparable complexity is rarely mastered.

Most test users were able to complete the given tasks in between 20 to 50 minutes. This was quite fast considering that they never had any experience using the prototype before. For some test users, it was the first time they had used electronic mail and BIDS. Experienced users were testing the prototype with more confidence and completed the given tasks faster.

It was observed that test users who were not familiar with Sun generally faced the following difficulties:

- They assumed that input from the keyboard would be automatically placed on the active window. This was their experience with personal computers. They were very much disappointed when they found that the system did not accept their typing. Users who were familiar with Sun usually moved the pointer onto the view where they wished to enter data before they typed using their keyboard.
- They faced problems using the Sun 3-mouse buttons. It took them some time to make sure that their fingers pressed the right button.
- The pop-menu were also new for them. It took some time for them to familiarise themselves to press the <operate> button and hold it to view all the possible choices on the menu and then select a command.

System message are used by the system to guide users using the system. They are printed on the note view at the bottom of the window. The messaging is quite useful especially for tasks that require more than one user action. Initially most users were not aware of the presence of the messaging facility at the bottom of the window. This could be due to the absence of such facility in many applications. Once they realised that the facility was there, they gave more attention to the view especially when they were having difficulty in carrying out the given tasks.

Many users expressed that they started to gain more familiarisation and confidence with the system after they had completed a number of test tasks. They were able to complete the subsequent tasks that followed faster. This was due to the experience that they had gained after doing a number of tasks.

It was noted that the way users approached their problems varied from one user to another. Some users preferred to use the help facility whenever they were not sure to how accomplish the given task. Others preferred to carry out tasks by a trial and error method. When they were stuck and had no other options, they started consulting the help facility. Since the Help Button is continuously displayed on the primary window, users are reminded to use the facility to accomplish the task.

Observation: Users Carrying Out Test Tasks

Observing users carrying out test tasks is very useful because it identifies the mismatch between how the designer expected the system to be used and how the user actually uses it.

- Electronic Mail

Reading new mail is an easy task. Users were able to read new mail without any error. The Readmail Button is explicitly displayed on the primary window. When the button is activated the new mail is placed inside the scroll list view of the primary window.

The procedure to send a mail message is also quite straightforward. It involves the following three steps:

- First, press the Send Button. The button is explicitly displayed on the primary window. This causes the Send Window to open.
- Second, specify the recipient and compose the message in the Send Window.
- Third, activate the SendMail Button. A confirmer is prompted requesting the user to confirm his action.

Most test users were able to send a mail message successfully without much difficulty. One test user was confused with the buttons SendMail and Done in the Send Window. She thought that she had to press Done to indicate that she had composed the message and then to press SendMail to send the message.

Most users pressed the right button, i.e., the Address Button when they were requested to add a new alias. The problem was that they had difficulty finding for the command to create a new alias. This was due to the use of pop-up menu to access the command that allows users to add new alias. A few users commented that it would be a lot easier if the command was placed on a button. It was not surprising that the task was slightly difficult because it adopted the concept of progressive disclosure and was not designed for beginners.

Creating a new folder was not that difficult for many users. They started to realise that the pop-up menu was available and may be accessed from the view on the window by using the operate mouse button. One test user criticised the design that users who were not familiar with Smalltalk may not be able to associate the <operate> mouse button with a pop-up menu.

Refiling a message into another folder was quite hard for many users. The commands to carry out previous tasks were available from the secondary window which was opened by activating one of the buttons. Many users assumed that the command can be accessed in a similar manner. Some of the users thought that the command can be found in the Folder Window because it is related to folders.

The command to refile a message could be accessed from the pop-up menu in the scroll list view on the primary window, but it was hidden until users press the operate mouse button on the view. Three test users realised that the 'v' symbol on top of the view was an alternative way to access the pop-up menu using the select mouse button.

- <u>Bath Information Data Services</u>

The search for articles by title involved two main steps. The first step was very straightforward because the was a button labelled 'Title'. Most users activate the right button to start a title search. One user was confused with the Retrieve Button, thinking that the button can be used to retrieve a search. Most users were able to enter the search on the prompter and then pressed the OK Button or entered a *carriage return*.

The second step, i.e., selecting the summary of the search is not easy for some users even though the prototype implements at least two navigating strategies, i.e. forcing the pointer to point at the summary of the search and the use of the note view requesting users to select the summary. This was partly due to the following reasons:

- Users did not realise that they had to carry out another action to complete the task. They expected to obtain the final result after entering the search.
- Since the retrieval of the summary requires some time, it was observed that many users did not concentrate much on interface changes. Some users were looking at the test tasks, "what to do next". Others were trying to discover various features supported by the prototype. It was also noted that some users had the habit of moving the mouse to the place where their attention was focused. This caused a failure to the navigating mechanism of the prototype.

Most users pressed the Save Button when they were asked to save searches. This caused the Save Window to open. An instruction was written at the bottom of the window requesting users to select the summary of searches and enter a file name. Many users simply did not care to read the message at the bottom of the window.

Most of the users had the experience of saving a file by just writing a filename followed by pressing the OK Button. They repeated the same experience but failed to save because the system requires an extra step, i.e., selecting the summary of searches.

The label 'Preferences' on one of the buttons was quite straightforward for many users. Others found it hard to understand its meaning and they suggested other names to make it more understandable. But once the right window was open, changing year was easy. They simply selected the year in the year list.

It was also noted that users moved and clicked the pointer on the heading view, the view that holds the information to be manipulated, thinking that by clicking on it, a pop-up menu would come out. Whenever users moved the pointer onto the view, a note view prompted a message informing users to use the Preferences Button to change the selection.

When users were given the task to search by author whose surname was 'Colomb', most users simply input the name without checking the input syntax. It was observed that even a librarian who was one of the test users did the same mistake. Since entering surname alone is acceptable by the library service, they just assume it is the same for BIDS.

BIDS requires users to append the character "_*" whenever author's initial is unknown. The majority of the users did not input the name with the right syntax and this was why no record was found. Generally users were anxious when the prototype prompted a message stating that no record was found. They asked whether they had entered the right spelling or they had not input the name in the right way.

Users who are familiar with BIDS liked its interface. There was a greater appreciation from the test users who had a bitter experience using BIDS. An information specialist expressed her great satisfaction with user interface to BIDS. She remarked, "the prototype has made an unfriendly interface usable".

The file editor/manager was generally easy to use. Most of the users were able to carry out the given tasks without any difficulty. A few of them expressed their great satisfaction using the file manager and they praised it a lot.

7.2 RECOMMENDATIONS

Recommendations given are based on the results of the evaluation. User's comments during the evaluation are placed in appendix B3.

It was noticed that users faced difficulties after they had opened a secondary window. Some of them asked whether a help facility was available. Hence, it is important to place a Help Button on every secondary window. The help information should be related to functions supported by the current window.

In between test tasks, users were asked at random to carry out simple actions that were related to the given tasks. For example, users were asked to scroll a view to see a complete display. It was found that some users had difficulty with scrolling because the Smalltalk scroll bar was quite slim. The width of the scroll bar should be increased so that it is more convenient for users to scroll.

It was noticed that test users faced no problem executing tasks that were available on buttons. Thus, it is recommended that more buttons should be placed on windows because they are simple and easy to use. The trade-off is they occupy more space on the window.

It was also noticed that many users neglected the instruction on the note view. A few of them said that they did not give much attention to the instruction because it was placed at the bottom of the window. An alternative is to place important instructions at the top of the window and further evaluation is required to test its effectiveness.

The prototype should also scan user input to BIDS when users carry out searches by author. It should be able to accept other standard formats which are more familiar to users. For example, if users enter 'Colomb', then it is taken as a surname by the prototype. The prototype could also implement a form fill-in interface where different columns are provided for surname and initials.

BIDS should prompt a modal window after users have entered a search. The window should display the partial result of the search and request a decision to go ahead with the search by pressing one of the window buttons. The use of modal window will ensure successful completion of the search . They would also have an alternative if they wish to cancel the search. Figure 7.1 shows a suggested modal window.

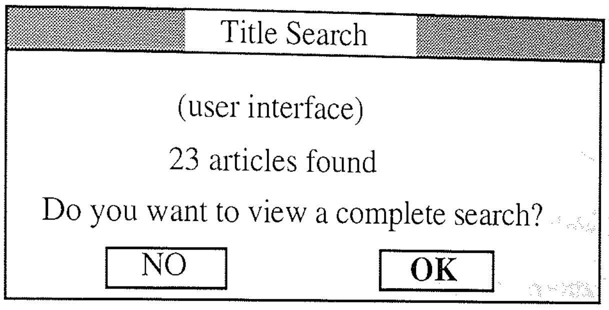


Figure 7.1 : Modal Window

7.3 QUESTIONNAIRE RATING

The average rating of the prototype is given in the appendix B1.

Nielsen and Levy (1993) carried out a study on 127 user interfaces. A rating scale of 1-5 was used with 1 being the worst and 5 the best. They found that the average rating was 3.6. They conclude that 3.6 seems to be a better estimate of average subjective satisfaction for a scale rating of 1-5.

In the present study a similar scale rating of 1-5 was used, but 1 is taken as the best and 5 is taken as the worst. Hence, a rating not worse than 2.4 is to be achieved for statements that require subjective evaluation. The value is derived from the study conducted by Nielsen and Levy (1993).

The rating for statement 1 (learning to use the system is easy) is 2.0. Generally, users agreed that the prototype was easy to use. The rating is better than the 2.4 average rating. The author believes that the rating could be better if the following factors were taken out:

- Some users were having difficulties due to the lack of experience in using Sun.
- It was the first time users use the system and they were asked to carry out tasks which were designed for advanced users.

Statement 7 (The system is protected from problems due to operating system or network) receives the best rating, i.e. 1.2. No crash occurred during the test session. Statement 6 (The prototype is reliable, ..) receives a worse rating compared to statement 7. This was probably due to bugs which still exist in the program. A few test users refused to give rating to statement 6 and 7. They argued that if during their testing no crash occured did not means that the system was robust. It could occur at other times.

Statement 8 (The Email application is better that the package I usually use) receives the worst rating, i.e. 2.4. Some users mentioned that they still prefer the email package that they were using because the email of the prototype does not provide the functions which are supported by their packages. The rating of this statement was expected because there are a number of email packages which are quite easy to use. It seems that there is a possibility of bias because users tend to like something that is familiar to them.

Statement 9 (The BIDS application is better that the package I usually use) receive a good rating, i.e. 1.4. This is due to the lack of good user interface softwares for BIDS. It is also an indication that users like interfaces which are based on a graphical user interface.

Statement 6 (Help system is good) is the subjective statement that receives the worst rating. The rating is 2.2, thus it is still acceptable (better than the 2.4 average limit). The author believes that the rating for this statement is poor because of the following reasons:

- Low usage of the system during the test session.
- Design problems, for example, there is no help facility on secondary windows.

Appendix B2 shows the grouping of test users according to their background experience. It also compares the rating given by the different groups of users. The average rating given by each group is in the range between 1.75 to 2.0 which is better than the average rating for the scale 1-5. There is no significant difference between the rating given by different groups of users. There are many factors that affect the judgement of test users. Experienced users may give a worse rating because of their experience using a wide range of software applications. On the other hand, they may give a better rating because of their appreciation of the prototype. Inexperienced users may give a worse rating due to problems using the system and extra effort in learning to understand new applications. There is also a possibility that they give a better than justified rating, blaming themselves for difficulties in using the system.

The fact that there is no significant difference between the rating given by different groups of users and the rating is better than the average rating indicate that the prototype is usable to different categories of users. The rating given by Group1 (users who are familiar with Sun and the application domain) is slightly better than other groups (users who are not familiar with either Sun or the application domain or both). Group1 perhaps shows a greater appreciation of the prototype because of their knowledge and experience with the system and the application domain.

Further research with greater number of users and a revised set of questionnaires are required in order to study how different users with different computer backgrounds respond to the system.

CHAPTER EIGHT

CONCLUSIONS

8.0 INTRODUCTION

Networked services are a rapidly growing environment and they facilitate communication among users and encourage the creation and dissemination of information. Even though, the services have undergone rapid changes, software tools that help users access and use networked information are still lagging behind (Dilation *et al.*, 1993). Recently, several books related to networked services have been published (Kahn & Stout, 1994; Dern, 1994; Tolhurst, 1994; Gardner, 1994). Most of them describe how to use disparate tools to access networked services. This reflects the range and the complexity of the problem.

The aim of this study was to explore the feasibility of developing an effective means of accessing networked services. To a large extent this aim has been achieved. A prototype which is based on a graphical user interface has been developed. The prototype consists of three applications: electronic mail, Bids and file manager/editor. The design of the prototype user interface is based on user interface guidelines. An evaluation has been carried out to study the effectiveness of the prototype.

8.1 GUIDELINES TO USER INTERFACE DESIGN

User interface guidelines were reviewed before and during the construction of the prototype. The referral to user interface guidelines were helpful even though many of the interaction mechanisms could be designed based on common sense and past experience with other software tools. Since it was the first time the author developed user interface software, there was a tendency to overlook some important aspects of the design. The guidelines also helped to reinforce some of the decisions in implementing the design.

Numerous guidelines to user interface design have been proposed in the literature. Only those guidelines which were applicable to the application to be developed were selected. Most of the guidelines were generally defined and required further interpretation when applied to specific tasks.

Some of the guidelines that were given greater emphasis in the design of the prototype user interface were:

- Simplicity - the concept of "progressive disclosure" is adopted so that the system appears simple but provides the advance features desired by advanced users.
- Consistency - consistency is enforced within an application and also among all applications of the prototype. Experiences gained from using an application can be applied when using other applications.
- Feedback - the system keeps users aware of what is going on. Different feedback mechanisms are used to indicate the status of the computation. Different pointer images are also used to indicate different types of modes on the interface.
- User control - the prototype windows are mostly non-modal windows and they provide a flexible interaction with the application. Users may work with more than one task simultaneously.

8.2 DESIGN OF THE PROTOTYPE

The prototype is completely independent of networked services. There is a clean separation between the prototype and application codes of networked services. The advantage offered by the separation is that, the prototype can be implemented without knowing the detailed implementation of the service and can be altered without having any adverse effect on the application codes.

Most interactions to networked services are through full-screen menu based interfaces. The advantage of this kind of interface is that it can be supported by low-cost terminals. Since the prototype is based on a graphical user interface, it runs only on powerful platforms that support bit-mapped screen displays. Advances in microelectronics technology today have made powerful hardware available at a reasonable price.

The prototype integrates different application programs into a single application. The integration provides users with a consistent method of accessing and using networked services. Consistency can be enforced not only within applications, but also in the whole environment.

The integration of services also provides means for the services to communicate with one another. Information obtained from a networked service can be manipulated by users according to their requirement. If they wish, they may copy and paste the information onto a simple file editor, edit it and then print. They may also incorporate part of the information into mail messages before sending it to their colleagues via electronic mail.

8.3 IMPLEMENTATION OF THE PROTOTYPE

The prototype has been developed using an object-oriented language, Smalltalk-80. An interface can be built more quickly because the Smalltalk class library supports a variety of objects such as menu, scroll bar, buttons and dialogue boxes which are available for reuse. The class library also contains a variety of types of view and controller classes. Some of these views were reused for the construction of the prototype user interface.

Existing views may not provide all the behaviour needed by the prototype. Using the Smalltalk inheritance mechanism, new subclasses are created to provide the prototype with the required behaviours (see 6.6.3). The inheritance mechanism permits rapid creation of new classes by refining existing classes.

Smalltalk has been recognised as being an effective tool for the construction of interactive graphical interfaces. The Smalltalk Model-View-Controller (MVC) methodology has been used as the framework for the construction of the prototype user interface. The methodology is based on three classes - Model, View, and Controller. These classes work together as components of an interactive system. Communication between the components is handled by passing messages and by the dependency mechanism of Smalltalk. (Dependency mechanism - see 6.3)

The prototype has proved that Smalltalk is also capable of communicating with different computers and applications in a networked environment. The communication is implemented by using UNIX pipes and forks, in that the output from the application is piped into the interface objects. Existing codes were reused for this purpose and much of the development time has been saved.

Features of object-orientation are effective in containing change. Changes can be made quickly without disrupting other parts of the application. Such a capability is essential because the prototype has to accommodate changes whenever there are changes to network configuration. During the development of the prototype several changes occurred in the network and networked services. For example, during the early stage of development access to BIDS was via the Spock machine. The machine was scrapped and access to the service was then rerouted via the Quipu machine. The procedures to access the service were also changed.

Smalltalk has been an effective environment for the construction of the prototype user interface to networked services. However, care must be taken to ensure that the development of the user interface of the prototype is not influenced by the Smalltalk interaction mechanisms and widgets. Some of them may not be appropriate for user interaction with networked services. For example, the Smalltalk scroll bar and prompter.

The Smalltalk-80 version 4 has been used for the construction of the prototype. This version is rather robust compared to the earlier version 2.5. The Smalltalk image of the prototype never crashed during its development. For future development, a later version of Smalltalk, i.e. version 4.1 or VisualWork 2.0 is proposed. These versions provide more mature objects which are needed for the construction of user interfaces. Some of the recommendations to the prototype such as to increase the size of scroll bar were not implemented, partly because they could be easily developed if the later version were to be used.

8.4 EVALUATION OF THE PROTOTYPE

The evaluation of the prototype with actual users was carried out to test the usability of the interface, and to identify good and bad features. The evaluation comprises two stages:

- Executing test tasks - Test users were requested to execute a list of tasks given to them. During this stage the facilitator observed and noted the way users used the system. Users were also encouraged to give their opinions about the prototype.

- Answering questionnaire - Users were then requested to rate the prototype by answering the questionnaire.

The evaluation was carried out after a review of literature that described user interface evaluation. Observing users executing test tasks and rating the system using questionnaire are common approaches for evaluating an interface and they are easy and cheap.

Observing users executing test tasks and users' opinions about the system were helpful because the advantages and disadvantages of each design aspect could be identified. The data obtained from this process was used to suggest further improvement to the prototype. A new version of the prototype was not reconstructed due to the time constraint. Had time permitted, iterative design and evaluation could have been carried out until a satisfactory interface was produced.

The result obtained from users' rating indicated that the prototype was usable and users generally liked the interface. The prototype incorporates an online help facility in its system. However, there were no conclusive results obtained from the evaluation regarding the effectiveness of the help facility. Some usability problems were identified during the evaluation. The online help facility requires further research.

The results of the evaluation were not comprehensive because all test users were novices. They were considered novices because it was the first time they used the prototype.

8.5 FUTURE DEVELOPMENT

BIDS provides access to various bibliographic citation databases. Searches can be made by selecting various fields such as title, author and journal. The prototype has provided all these functions within a graphical user interface environment. The results of the evaluation showed that users preferred the prototype user interface compared to the current menu-based interface.

The prototype has proved the effectiveness of employing database search in a graphical user interface environment. The prototype only provides a simple search to databases. Further research is needed to include advanced database search which is more complicated. Further research may include other techniques such as natural language processing and form fill-in to enhance its effectiveness.

The interface could be developed to allow developers or advanced users to customise the interface according to their requirements. This is due to growing requirement for users participation in interface design. Frequent and expert users may wish to have more controls of the interface. For example, users who do not usually use a carbon copy function may not like the function to appear on the message header of their electronic mail.

The prototype was developed on a Sun Sparcstation that runs Sun's operating system. Since Smalltalk is portable, the prototype should also run on other hardware platforms. The prototype was tested on the Macintosh LC. It worked fine but a few alterations were required. For example, the prototype windows had to be resized to suit the Macintosh LC small screen. Communications with the networked services could not be established because the prototype used the object Terminal for communications. The object uses Unix commands, thus it is not compatible with other platforms.

Further development to allow the prototype to run on the Macintosh was impeded by the hardware constraint. The four megabytes of RAM available during the time of development were just enough to run the prototype on the platform. An object which allows the prototype running on other platforms to communicate with networked services could be incorporated, thus allowing the prototype to be completely portable.

Visvalingam (1988) points out one of the challenges in user interface design is the need to accommodate changing user requirements. The features which were initially regarded as helpful became tiresome and irritating. Some useful approaches have been suggested and thus require further research.

The prototype can be developed further so that it include other networked services. Communications between these services could be automatically established by the prototype. For example, after carrying out searches using BIDS, users usually access the local library to check whether the journals that they wish to review are available. Instead of having users logging on to the library in order to carry out another search, the prototype may provide options to users if they wish the prototype to carry out the search for them.

The prototype is implemented in such as way that the interface keeps track of the position of the pointer and detects which mouse button is pressed. The information obtained is used to guide users using the system. For example, if users press the wrong button, a message appears informing that the button is not used by the system. This information can be stored and can be used as a source of information for research on adaptive user interfaces. Until now literature that describes adaptive systems in a graphical user interface environment is rare.

8.6 SUMMARY

This research has demonstrated the effectiveness of accessing networked services when the user interaction is aided by a well designed user interface. A prototype has been developed to help users interact with the services. The prototype provides access to electronic mail, Bids and file manager/editor and it adopts a graphical user interface style of interaction.

The design of the prototype user interface were based on the guidelines of user interface design. Even though user interface guidelines are generally defined, understanding of the guidelines helps produce a usable user interface.

Developing graphical user interface software is not an easy task. This research has shown that Smalltalk is an effective tool for the construction of such an interface. The Smalltalk Model-View-Controller framework is a powerful paradigm which separates interactive applications into three main components that can be linked together. The Smalltalk class library contains many objects which are needed for construction of the prototype user interface and the object for the prototype to communicate with networked services.

An evaluation followed the completion of the prototype. The results of the evaluation indicated that the prototype was usable and users generally liked the interface. The evaluation also has identified some difficulties faced by users when interacting with the prototype. The results of the evaluation have been used to recommend improvements to the interface.

BIBLIOGRAPHY

Allwood, C. M. (1986), "Novices on the Computer: a review of the literature", *International Journal of Man-Machine Studies* **25**(6), pp. 633-658.

Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H.F & Secret, A (1994), "The World-Wide Web", *Communications of the ACM* **37**(8), pp. 76-82.

Berry, R. E. & Reeves, C, J. (1992), "The Evolution of the Common User Access Workplace Model", *IBM Systems Journal* **31**(3), pp. 414-428.

Black, U. (1987), *Computer Networks - Protocols, Standards and Interfaces*, Prentice-Hall International Inc.

Brennan, R., Thompson, K. & Wilder, R. (1991), "Mapping the X Window onto Open Systems Interconnection Standards", *IEEE Network Magazine* **4**(2), pp. 32-40.

Brown, J. R. & Cunningham, S. (1989), *Programming The User Interface - Principles and Examples*, New York: Wiley.

Burbeck, S. (1987), "How to Use the Model-View-Controller : Application Programming in Smalltalk-80", Softsmarts Inc.

CCITT (1988), *Data Communication Networks Message Handling Systems*, CCITT Blue Book, Vol VIII.7, Recommendations X.400-X.420.

Carroll, J.M. & Mazur, S. A., (1985), "Lisa Learning", *Computer* **19**(11), pp. 35-49.

Davis, F.D. (1993), "User Acceptance of Information Technology: System Characteristics, User Perceptions and Behavioral Impacts", *International Journal of Man-Machine Studies* **38**(3), pp. 475-488.

de Souza F.L. & Bevan, N. (1990), "The use of Guidelines in Menu Interface Design: Evaluation of a draft standard", in Diaper,D., Gilmore, D., Cockton, G. & Shackel, B. (eds), *INTERACT '90 : Human-Computer Interaction*, pp. 435-440.

Ege, R. K. (1992), *Programming in an Object-Oriented Environment*, London: Academic Press.

Eglowstein, H. & Smith, B. (1994), "E-Mail From Afar", *Byte* **19**(5), pp. 122-160.

Erlinger, M.A. (1993), "Networking - Centerpiece of Academic Computing", *IEEE Transactions on EDUCATION* **36**(1), pp. 90-93.

Gray, P. A. (1991), *Open Systems: a business strategy for the 1990s*, McGraw-Hill, Book Company.

Gray, P.D. & Mohamed, R (1990), *Smalltalk-80 : A Practical Introduction*, Pitman Publishing, UK.

Grimm, R. & Heagerty, D. (1989), "Recommendation for A Shorthand X.400 Address Notation", *Computer Networks and ISDN Systems* **17**(4), pp. 263-267.

Gould, J. & Grischkowsky, N. (1984), "Doing the same work with Hardcopy and Cathode Ray Tube (CRT) terminals", *Human Factors* **26,** pp. 323-337.

Hansen, W. J., Doring, R. & Whitlock, L. R. (1978), "Why an examination was slower on-line than on paper", *International Journal of Man-Machine Studies* **10**, pp. 507-519.

Henshall, J. & Shaw, S. (1988), "Message handling systems, X.400/MOTIS", *OSI Explained: end-to-end computer Communication Standards*, Chichester: Ellis Horwood, pp. 158-200.

Kahn, R. E. (1994), "Viewpoint: The Role of Government in the Evolution of the Internet", *Communications of the ACM* **37**(8), pp. 15-21.

Krasner, G.E. & Pope, S.T. (1988), "A Cookbook for Using The Model-View-Controller User Interface Paradigm in Smalltalk-80", *Journal of Object-Oriented Programming* **1**(3), pp. 26-49.

Kille, S.E. (1989), "PP - A Message Transfer Agent", in Stefferud, E., Jacobsen, O.J. & Schicker, P. (eds), *Message Handling Systems and Distributed Applications*. pp. 115-128, Amsterdam: North-Holland.

Lebeck, S, K. (1989), "Implementing MHS: 1984 versus 1988", in Stefferud, E., Jacobsen, O.J. & Schicker, P. (eds), *Message Handling Systems and Distributed Applications*, pp. 101-114, Amsterdam: North-Holland.

Marcus, A. (1993), "Human Communications Issues in Advanced UIS", *Communication of the ACM*, **36**(4), pp. 100-109.

Manros, C. (1989), *The X.400 Blue Book Companion : CCITT X.400 MHS 1988 : ISO/IEC MOTIS*, UK: Technology Appraisals.

Monk, A., Wright, P., Harber, J. & Davenport, L. (1993), *Improving Your Human-Computer Interface- A Practical Technique*, Prentice-Hall International.

Nielsen, J., Frehr, I. & Nymand, N.O. (1991), "The Learnability of HyperCard as an Object-Oriented Programming System", *Behaviour and Information Technology* **10**(2), pp. 111-120.

Osborne, D.J. & Holton, D. (1988), "Reading from Screen versus Paper: There is no Difference", *International Journal of Man-Machine Studies* **28**(3), pp. 1-9.

Pangalos, G.J. (1992), "Consistency and Standardization of User Interfaces", *Information and Software Technology* **34**(6), pp. 397-403.

Potosnak, K.(1988), "Do Icons Make User Interface Easier to use?", *IEEE Software* **5**(3), pp. 97-99.

Prasad, K. V. K. K. (1990), "On the User Interfaces for Electronic Mailsystem", *Telematics and Informatics* Vol **7**(2), pp.145-149.

Raymond, D.R. (1992), "Flexible Text Display with Lector", *Computer* **25**(8), pp. 49-60.

Reenskaug, M.H.T. (1981), "User-Oriented Descriptions of Smalltalk Systems", *Byte* **7**(8), pp. 75-81.

Samuelson, B. (1993), "Smalltalk Benchmarking Revisited", *The Smalltalk Report* **2**(8), pp. 16-21.

Schicker, P. (1989), "Message Handling Systems, X.400 (An Overview)", in Stefferud, E., Jacobsen, O.J. & Schicker, P. (eds), *Message Handling Systems and Distributed Applications*, pp. 3-41, Amsterdam: North-Holland.

Shan, Y (1990), "MoDE: A UIMS for Smalltalk", *ECOOP/OOPSLA '90 Proceedings*, pp. 258-268.

Plaisant, C. & Shneiderman, B. (1992), "Scheduling Home Control Devices: Design Issues and Usability Evaluation of Four Touchscreen Interfaces", *International Journal of Man-Machine Studies* **36**(3), pp. 375-393.

Six, H.W. & Voss, J. (1991), "User Interface Development:Problems and Experiences", in H. Maurer (Ed) *Lecture Notes Computer Science* 555, pp. 306-319.

Simmonds, C. (1993), "Searching Internet Archive Sites with Archie - Why, What, Where, and How", *Online* **17**(2), p.50.

Smith, S. L. & Mosier, J.N. (1986), *Design Guidelines for Designing User Interface Software. Technical Report* MTR-10090, The MITRE Corporation, Bedford, USA.

Steele, D. (1992), *Human Computer Interaction*, British Standards Institution, London.

Sweeney, M., Maguire, M. & Shackel, B. (1993), "Evaluating User-Computer Interaction: A Framework", *International Journal of Man-Machine Studies* **38**(4).

Thimbleby, H. (1994), "Formulating Usability", *SIGCHI* **26**(2), pp. 59-64.

Tognazzini, B.(1992), *Tog on Interface*, Massachusset: Addison-Wesley.

Tarouco L.M.R. (1984), "User Friendly Interface for Messaging Systems", in Smith, H.T. (eds), *Proceedings of the IFIP WG 6.5 Working Conference on Computer-Based Message Services*, Nottingham, England, pp. 167-175, May 1984.

Ulich, E., Rauterberg, M., Moll, T., Greutmann, T. & Strohm, O. (1991), "Task Orientation and user-oriented dialogue design", *International Journal of Human-Computer Interaction* **3**, pp. 117-144.

Westlake, D.R. (1992), *Geac: A Guide for Librarians and System Managers*, (2nd ed), Ashgate Publishing Limited.

Wilbur, S. (1991), "Impact of Electronic Mail on Organisational Structure", *IEE International Conference on Information Technology in the Workplace*, pp. 27-29.

Wilson, P.A. (1984), "Structures for Mailbox System Applications", in Smith, H.T. (eds), *Proceedings of the IFIP WG 6.5 Working Conference on Computer-Based Message Services*, Nottingham:England, pp. 149-166, May 1984.

Wirfs-Brock, A. (1991), "Getting to the Mainstream", *Journal of Object-Oriented Programming* **4**(2), pp. 51-54.

Wirfs-Brock, A. (1991), "Surviving in the Mainstream", *Journal of Object-Oriented Programming* **4**(3), pp. 64-66.

Wright, P. & Lickorish, A., (1983), "Proof-reading texts on screen and paper", *Behaviour and Information Technology* **2**(3), pp. 227-235.

REFERENCES

Apple Computer, (1992), *Macintosh Human Interface Guidelines*, Massachusset: Addison-Wesley.

Apple Computer, (1991), *Macintosh Reference: System 7.*

Apple Computer, (1987), *Apple Human Interface Guidelines: The Apple Desktop Interface*, Massachusset: Addison-Wesley.

Benbasat, I & Todd, P. (1993), "An Experimental Investigation of Interface Design Alternatives: Icons vs. Text and Direct Manipulation vs. Menus", *International Journal of Man-Machine Studies* **38**(3), pp. 369-402.

Berry, R.E. (1992), "The designer's model of the CUA Workplace", *IBM System Journal*, **31**(3), pp. 429-458.

Blankenberger, S & Hahn, K. (1991), "Effects of Icon design on Human-Computer Interaction", *International Journal of Man-Machine Studies* **35**(3), pp. 363-378.

Booth, P. A. (1989), *An Introduction To Human-Computer Interaction*, Hove : Erlbaum.

Bourne J.E. (1992), *Object-oriented Engineering: Building Bngineering Systems Using Smalltalk-80*, Richard D. Irwin, Inc.

Brindley, L.J. (1990), "The Electronic Campus and the Challenges of the 1990s", *University Computing* **12**(4), pp. 154-158.

Buxton, B. (1993), "HCI and the Inadequacies of Direct Manipulation Systems", *SIGCHI Bulletin* **25**(1), pp. 21-22.

Card, S. K., Robertson, G.G., & Mackinlay, J.D. (1991), "The Information Visualizer: An Information workspace. *Proceeding of ACM CHI'91, Conference* (New Orleans, LA, pp. 181-188, April/May 1991

Che Mee, I. (1990), *A Smalltalk-80 Electronic Laboratory Simulation Program for Computer Assisted Learning*, Phd Thesis, University of Bradford.

Chen, J. (1990), "Providing intrinsic support for user interface monitoring", in Diaper,D., Gilmore, D., Cockton, G. & Shackel, B. (eds), *INTERACT '90 : Human-Computer Interaction,* pp. 415-420.

Clark, I. A. (1981), "Software Simulation as a Tool for Usable Product Design", *IBM System Journal* **20**(2), pp. 273-295.

Coats, R. B. & Vlaeminke, I. (1987), *Man-Computer Interfaces: An Introduction to Software Design and Implementation,* Oxford : Blackwell Scientific.

Colomb, R. M. (1991), "Use of a Personal Workstation to Access Open Network Services", *Australian Computer Journal* **25**(1), pp. 7-13.

Comer, D. (1988), *Internetworking with TCP/IP - Principles, Protocols and Architecture*, USA: Prentice-Hall Inc.

Cook, S. (1991), "A new Smalltalk", *EXE Magazine* **5**(8), pp. 43-48.

Cox, B.J. (1984), "Message/Object Programming: An Evolutionary Change in Programming Technology", *IEEE Software* **1**(1), pp. 50-61.

Dern, D.P. (1994), *The internet guide for new users*, McGraw-Hill.

Diederich, J. & Milton, J. (1987), "Experimental Prototyping in Smalltalk", *IEEE Software* **4**(3), pp. 50-64.

Dillon, M., Jul, M., Burge M. and Hickey, C., (1993), "Assessing Information on the Internet: Toward Providing Library Services for Computer-Mediated Communication", *Internet Research* **3**(1), pp. 64-69.

Dixon, T. (1993), "Obstacles on the Road to Global Networking", *Computer Networks and ISDN Systems* **25**(1), pp. 9-15.

Dodani, M. H., Hughes, C. E. & Moshell, M. J. (1989), "Separation of Powers", *Byte* **14**(3), pp. 255-262.

Duffy, T.M., Palmer, J.E. & Mehlenbacher, B. (1992), *Online Help: Design and Evaluation*, New Jersey: Ablex.

Elkerton, J. (1988), "Online Aiding for Human-Computer Interface" in Helander, M. (ed), *Handbook of Human-Computer Interaction*, pp. 345-364.

Erickson, C. (1993), "USENET as a Teaching Tool", *SIGCSE Bulletin* **25**(1), pp. 43-47.

Gaines, B. R. (1981), "The Technology of Interaction - Dialogue Programming Rules", *International Journal of Man-Machine Studies* **14,** pp. 133-150.

Gardner, J. (1994), *A Dos User's Guide to the Internet: email, netnews, and file transfer with UUCP*, New Jersey: Prentice-Hall.

Gay, B. (1933), "Object-Orientation, User Interfaces and Reflective Computation", unpublished paper, Department of Computer Science, Aston University.

GEAC (1991), *GEAC Implementation at Aston University Library Information Services,* available from Aston University Library.

Goldberg, A & Pope, S.T. (1983), "Object-Oriented Programming is not Enough", *American Programmer* **2**(7&8), pp. 46-59.

Goldberg, A & Robson, D. (1983), *Smalltalk-80 The Language and its Implementation*, Massachusset: Addison-Wesley.

Gosling, J., Rosenthal, D.S.H. & Arden, M.J. (1989), *The News Book, An Introduction to the Network/Extensible Window System*, New York: Springer-Verlag.

Graham, I. (1991), *Object-Oriented Methods*, Massachusset: Addison-Wesley.

Grudin, J. (1993), "Interface - An Evolving Concept", *Communication of the ACM* **36**(4), pp. 110-119.

Grudin, J. (1989), "The Case Against User Interface Consistency", *Communication of the ACM* **32**(4), pp. 72-81.

Hahn, H. & Stout, R.(1994), *The Internet Complete Reference*, London : Osborne-McGraw-Hill.

Hayes, F. & Baran, N. (1989), "A Guide to GUIs", *Byte* **14**(7), pp. 250-257.

Houghton, R.C. (1984), "Online Help Systems: A Conspectus", *Communications of ACM* **27**(2), pp.126-133.

Hix, D. & Hartson, R. (1993), *Developing User Interfaces: Ensuring Usability through Products and Process*, New York: Wiley.

Huan-Chao, K. (1991), *Comprehensive Support for Developing Graphical, Highly Interactive User Interface Systems*, PhD Thesis, Oregon State University, USA.

Hutchins, E.L., Holland, J.D., & Norman, D.A. (1986), "Direct Manipulation" In Norman, D.A. & Draper, W.S. (eds): *User Centered System Design: New Perspectives in Human-Machine Interaction*, London: Lawrence Erlbaum.

Johnson, P. (1992), *Human Computer Interaction - Psychology, Task, Analysis and Software Engineering*, London : McGraw-Hill.

Kaehler, C. (1988), *HyperCard Power: Techniques and Scripts*, Massachusset: Addison-Wesley.

Kappel, G. & Min Tjoa, A (1992), "State of Art and Open Issues on Graphical User Interfaces for Object-Oriented Database Systems", *Information and Software Technology* **34**(11), pp. 721-730.

Karat, J. (1988), "Software Evaluation Methodologies" in Helander, M. (ed), *Handbook of Human-Computer Interaction*, Amsterdam : North-Holland, pp. 891-903.

Krivine, F. (1989), *The Aston Information Strategies*, available from Aston University Library.

Lalonde, W.R., Pugh, J.R. & Thomas, D.A. (1985), Smalltalk : Discovering The System, SCS-TR-80, School of Computer Science, Carleton University, Ottawa.

Library & Information Services (1993), *Anybody's Networking Book - A Simple Guide to Networked Services at Aston University*, available from Aston University Library.

Lodding, K. N. (1983), "Iconic Interfacing", *IEEE Computer Graphic and Application* **3**(11), pp. 11-20.

Maguire, M.C. (1990), "A Review of Human Factor Guidelines and Technique for the Design of Graphical Human-Computer Interfaces", in Preece, J. & Keller, L. (eds), *Human-Computer Interaction*, Hempstead : Prentice-Hall-Open University, pp. 161-184.

Mandelkern, D. (1993),"Graphical User Interfaces: The Next Generation-Introduction", *Communication of the ACM* **36**(4), pp. 36-39.

Marcus, A. (1992), *Graphic Design for Electronic Documents and User Interfaces*, Massachusset: Addison-Wesley.

Margono, S. & Shneiderman, B. (1987), "A Study of File Manipulation by Novices using Commands vs. Direct Manipulation", *26th Annual Technical Symposium, Chapter of the Association of Computing Machinery*, Washington D.C., pp. 57-62, June 1987

Microsoft Corporation (1991), *User's Guide: A Microsoft Window.*

Morrow, T (1992), "Bids ISI - A New National Bibliographic Data Service for the UK Academic Community", *Computer Networks and ISDN Systems* **25**(4-5), pp. 448-453.

Morse, A. & Reynolds, G. (1993), "Overcomming Current Growth Limits in UI Development", *Communication of the ACM* **36**(4), pp. 72-81.

Myers, B.A., (1985), "The Importance of Percent-Done Progress Indicator for computer-human interfaces", *Proceeding of ACM CHI'85 Conference,* San Francisco, pp. 11-17, April 1985.

Myers, B. D. (1992), "Demonstrational Interfaces: A Step Beyond Direct Manipulation", *Computer* **25**(8), pp. 61-73.

Nielsen, J. (1987), "Classification of Dialog Technique", *ACM SIGCHI Bulletin* **19**(2), pp. 30-35.

Nielsen, J. (1993a), "Noncommand User Interfaces", *Communication of the ACM* **36**(4), pp. 82-99.

Nielsen, J. (1993b), *Usability Engineering*, UK: Academic Press Limited.

Nielsen, J. & Levy, J. (1994), "Subjective User Preferences versus Objective Interface Performance measures", *Communication of the ACM* **36**(4), pp. 66-73.

Nielsen, J. & Levy, J. (1994), "Measuring Usability - Preference vs Perfomance", *Communication of the ACM* **37**(4), pp. 72-81.

Nielsen, J & Richards, J.T. (1989), "The Experience of Learning and Using Smalltalk", *IEEE Software* **6**(3), pp. 73-77.

Paap, K. R. (1988), "Design of Menu", in Helander, M. (ed), Ed. *Handbook of Human-Computer Interaction,* Amsterdam: Elsevier, pp. 205-235.

ParcPlace (1992), *Objectworks for Smalltalk-80, Release 4.1, User's Guide,* ParcPlace Systems, Inc.

Peek, J. D. (1991), *MH & xmh: E-Mail for Users and Programmers*, O'Reilly & Associates, Inc.

Pinson, L.J. & Wiener, R.S. (1988), An Introduction to Object-Oriented Programming and Smalltalk", Massachusset: Addison-Wesley.

Plattner, B. & Lubisch, H. (1989), "Electronic Mail Systems and Protocols. Overview and Case Study", in Stefferud, E., Jacobsen, O.J. & Schicker, P. (eds), *Message Handling Systems and Distributed Applications*, pp. 55-100, Amsterdam: North-Holland.

Powell, J.E. (1990), *Designing User Interfaces*, Microtrend Books, Slawson Communications, Inc.

Ravden, S. J. & Johnson, G. I.(1989), *Evaluating Usability of Human-Computer Interfaces : a Practical Method*, Chichester : Ellis Horwood.

Reinhardt, A. (1993), "Smarter E-Mail is Coming", *Byte* **18**(3), pp. 90-108.

Reisner, P. (1990), "What is Inconsistency", in Diaper, D., Gilmore, D., Cockton, G. & Shackel, B. (eds), *INTERACT '90 : Human-Computer Interaction*, pp. 175-181.

Rogers, Y. (1989), "Icons at the interface: their usefulness", *Interacting with Computers* **1**(1), pp. 105-117.

Root, R. W. & Draper, S. (1983), "Questionnaires as a Software Evaluation Tool", *Proceeding of ACM CHI'1983 Conference* , Boston, MA, pp. 83-87, December 1983.

Sadowsky, G. (1993), "Network Connectivity for Developing Countries", *Communications of the ACM* **36**(8), pp. 42-47.

Sellen, A. & Nicol, A. (1990), Building User-Centered On-Line Help. In Laurel, B. (ed). *The Art of Human-Computer Interface Design.*, Massachusset: Addison-Wesley.

Shneiderman, B. (1983), "Direct Manipulation: A Step Beyond Programming Languages", *Computer* **16**(8), pp. 57-69.

Shneiderman, B. (1992), *Designing the User Interface: Strategies for Effective Human-Computer Interaction*: (2nd ed), Massachusset: Addison-Wesley.

Smith, D.C., Irby, C., Kimball, R., Verplank, B. & Harslem, E. (1990), "Designing the Star User Interface", in Preece, J. & Keller, L.(eds), *Human-Computer Interaction*, Hempstead : Prentice-Hall-Open University, pp. 238-259.

Smith, S.L. (1986), "Standards versus Guidelines for Designing User Interface Software", *Behaviour and Information Technology* **5**(1), 47-61.

Sun Microsystems, inc., (1991), *OpenWin : User's Guide*, Massachusset: Addison-Wesley.

Sun Microsystems, inc., (1990), *OPEN LOOK Graphical User Interface Application Style Guidelines*, Massachusset: Addison-Wesley.

Sun Microsystems, inc., (1989), *OPEN LOOK Graphical User Interface Functional Specification,* Massachusset: Addison-Wesley.

Tanenbaum, A.S. (1989), *Computer Networks* (2nd ed), New Jersey: Prentice-Hall

Thomas, J.C. & Kellogg, W.A. (1989), "Minimizing Ecological Gaps in Interface Design", *IEEE Software* **6**(1), pp. 78-86.

Thomas, D. (1989), "What 's in an Object?", *Byte* **14**(3), pp.231-240.

Thimbleby, H. (1990), *User Interface Design,* New York : Addison-Wesley.

Tolhurst, W.A. (1994), The Internet Resource : Quick Reference, Indianapolis : Que Corporation.

Tullis, T. S. (1981), "An evaluation of Alphanumeric, Graphic, and Color Information Displays", *Human Factors* **23** pp. 541-550.

Tullis, T. S. (1988), "Screen Design", in Helander, M. (ed), *Handbook of Human-Computer Interaction,* Amsterdam: Elsevier, pp. 377-408.

Udell, J. (1990), "Smalltalk-80 Enters the Nineties", *Byte* **14**(7), pp. 138-142.

Verplank, W. L. (1988), Graphic Challenges in Designing Object-Oriented User Interfaces, in Helander, M. (ed), *Handbook of Human-Computer Interaction,* Amsterdam: Elsevier, pp. 365-376.

Visvalingam, M. (1988), "User Interface Design: Differing Requirements of Novices, Occasional and expert users", *University Computing* **10**(2), pp. 80-85.

Winblad, A. L., Edwards, S. D. & King D. R. (1990), *Object-Oriented Software,* Massachusetts: Addison-Wesley.

Ziegler, J.E. & Fahnrich, K.P. (1988), "Direct Manipulation" in Helander, M. (ed), *Handbook of Human-Computer Interaction,* Amsterdam: Elsevier. pp. 123-133.

Appendix A1

EVALUATION OF THE PROTOTYPE

Many thanks for your willingness to participate in the evaluation of this prototype. The purpose of the evaluation is to assess the system (prototype) and not the participant.

The prototype consists of three applications. The three applications and their respective functions in brief are as follows:

- Electronic Mail - allows one to send and receive electronic messages.
- Bath Information Data Service - allows users to search for articles by titles, authors' or journals' name.
- File Editor - provides users with a simple word processing capability.

The prototype is based on a windowing environment and the interaction is mainly by using the mouse pointer. Only the left and the middle mouse pointer will be used by the system.

During the evaluation session, you will be asked to perform several specific tasks and will be given some time to explore the system further, if desired so. Please 'talk aloud', i.e. to say what you are doing and why you are doing it. You may criticise and suggest improvements to the system. No doubt, your criticisms, suggestions, and opinions are invaluable that will help the facilitator/designer to understand more about the system being evaluated.

Again, you are reminded that the facilitator is not evaluating you. Your performance in carrying out the tasks will be used to measure how well the system serves the users. Please complete the questionnaire to rate the system after you have finished using it. If you wish, you may review the system while you are answering the questionnaire.

The session should last about one hour.
Drinks and biscuits will be provided after the session.

Appendix A2

USER DETAILS

Please tick (__) the following which is/are relevant to you

I am a student

_____ Undergraduate student

_____ Postgraduate student

I am staff

_____ (Please specify : ______________________________)

Please describe your computer/application experience by using the following scales:

0- never used
1 - used but not thoroughly familiar
2 - familiar

A) Hardware

_____ Sun workstation

_____ Macintosh

_____ IBM/IBM Compatibles

_____ Others (Please specify: ___________________________)

B) Applications

_____ Word Processing
(Please specify which package(s): _______________________)

_____ Electronic Mail Services
(Please specify which package(s): _______________________)

_____ Bath Information Data Services

Please comment on the following basing on your experience.

- Word Processing ______________________________________
- Electronic Mail Services ________________________________
- Bath Information Data Services ___________________________

Appendix A3

TEST TASKS-1

(user's copy)

1-12 are the test tasks to be evaluated during this session.

# Using Mail, send a short message to *abdullah@aston.sun.*

# Activate Electronic Mail from the Launcher.

1) Read new mail - the message that you have sent.

2) Create an alias name for your email address.
If you don't have an email address - use *abdullah@aston.uhura.*

3) Send a short message to your address OR to *abdullah@aston.sun.*

4) Create a new folder.

5) Refile the message into the new folder.

# Activate Bath Information Data Services.

6) Search for articles that contain a title 'user interface'.

7) Save the articles in a file.

8) You are connected to the Science Citation Index Database for the year 1994.
Change the connection to the year 1993.

9) Search for all articles by author whose surname is 'Colomb' (i.e. for 1992).

# Activate File Editor.

10) Open a file 'evaluation' . The file is in a folder named 'experiment'.

11) Copy articles found in (9) and paste to the file 'evaluation'.

12) Delete a folder named 'temporary' together with its content.

Appendix A4

TEST TASKS-2 (facilitator's copy)

1) Read new mail - the message that you have sent.
2) Create an alias name for your email address.
3) Send a short message to your address OR to *abdullah@aston.sun.*
4) Create a new folder.
5) Refile the message into the new folder.
6) Search for articles that contain a title 'user interface'.
7) Save the articles in a file.
8) Change the connection to the year 1993.
9) Search for all articles by author whose surname is 'Colomb' (i.e. for 1992).
10) Open a file 'evaluation'. The file is in a folder named 'experiment'.
11) Copy articles found in (9) and paste to the file 'evaluation'.
12) Delete a folder named 'temporary' together with its content.

Task	User's actions and comments	Facilitator's notes
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		

Appendix A5

QUESTIONNAIRE

1.	Strongly Agree
2.	Agree
3.	Not decided
4.	Disagree
5.	Strongly disagree
N/A	Not Applicable

1.	Learning to use the system is easy - - - -	{ }
2.	Commands and information are well presented on the screen -	{ }
3.	The system keeps users informed about what it is doing. -	{ }
4.	Messages provided by the system are helpful - - -	{ }
5.	Help system is good - - - - - -	{ }
6.	The prototype is reliable - (e.g. users' input did not cause the system to hang). - -	{ }
7.	The system is protected from problems due to operating system or network - - - - - - -	{ }
8.	The Email application is better than the package I usually use -	{ }
9.	The BIDS application is better than the package I usually use -	{ }

Please write any comment about the system in the space below, if you have any.

____

____

____

____

____

____

____

____

Appendix B1

AVERAGE RATING OF THE PROTOTYPE

No.	Statements	Average	Std.Dev.
1.	Learning to use the system is easy	2.0	0.9
2.	Commands and information are well presented on the screen	1.7	0.8
3.	The system keeps users informed about what it is doing.	1.9	0.9
4.	Messages provided by the system are helpful	1.8	0.8
5.	Help system is good -	2.2	0.9
6.	The prototype is reliable - (e.g. users' input did not cause the system to hang)	1.5	0.7
7.	The system is protected from problems due to operating system or network	1.2	0.4
8.	The Email application is better than the package I usually use	2.4	0.9
9.	The BIDS application is better than the package I usually use	1.4	0.5

Appendix B2

GROUPING OF TEST USERS

	Familiar with Sun	Not Familiar with Sun
Familiar with Application Domain	Group 1 (13 test users)	Group 3 (8 test users)
Not Familiar with Application Domain	Group 2 (5 test users)	Group 4 (4 test users)

RATING GIVEN BY DIFFERENT GROUPS OF TEST USERS

Group	Stmt1	Stmt2	Stmt3	Stmt4	Stmt5	Stmt6	Stmt7	Stmt8	Stmt9	Aver-age
Group1	1.85	1.54	1.77	1.92	2.1	1.56	1.1	2.42	1.2	1.77
Group2	2	2	1.8	1.4	3	1.67	1.33	2	1.5	1.84
Group3	2.25	1.75	1.88	2	2.3	1.38	1.25	2.43	1.5	1.86
Group4	2.25	1.5	2.5	2	1.75	1.75	1.5	2	2	1.96

Appendix B3

USERS' COMMENTS

- The system is really robust provided that one is familiar with it. An interesting training right before its use will facilitate the user.

- I found it a bit hard to hit the button (may be because I'm so used to Macintoshes). I think the system would be quite easy to use given more practice.

- Larger font and scroll bar for people with poor eyesight. Use of middle button for pop-up menu slightly unexpected. A good feature is the amount of button so that users need not remember too many command.

  With the Bids system, I would like to see slightly more sophisticated search window. I would like to see help on start-up.

  On mail I would like to see feature which permits the editing on e-mail header. mail message undelete is handy.

  On the file editors, the "--new file --" message should disappear when you start typing. A drag and drop zone would be useful. More help buttons instead of pop-up menus. Good system overall.

- Too many windows opening lead to confusion- need more sequence.

- The idea of using the middle button for help is not intuitive unless you are a Smalltalk users. Perhaps an extra help button on each screen would be better. My usual email is simple UNIX mail. This version is much better than the openwindow mailer.

The BIDS application is also an improvement. BIDS has some unusual restrictions such as the way author names are presented to the system. An extra help button for that window is really needed.

In general I found that this package more friendly than most UNIX interfaces.

- There are a few spelling mistakes. The word "Retrieve" on the button is confusing. I thought the function of the button was to carry out search.

  The word "Preference" should be changed to "Data-base" which is more appropriate.

  Pop-up menu should stay on the screen all the time.

- Excellent Potential and I like it very much. It makes an extremely unfriendly system usable.

  I like the appearance of the interface. It is simple. Not like the CD-ROM interface- I am irritated by its colours.

- The system is easy to use and useful. I think it is nicer and easier if commands are visible on the screen.

- Done is word that means that I am finished doing something. Close should be a better word - this means that you just want to close the window, whether you have done anything or not. Another suggestion is to add another button labelled Cancel.

  A message at the bottom of the window is good - it gives a clue about the interface

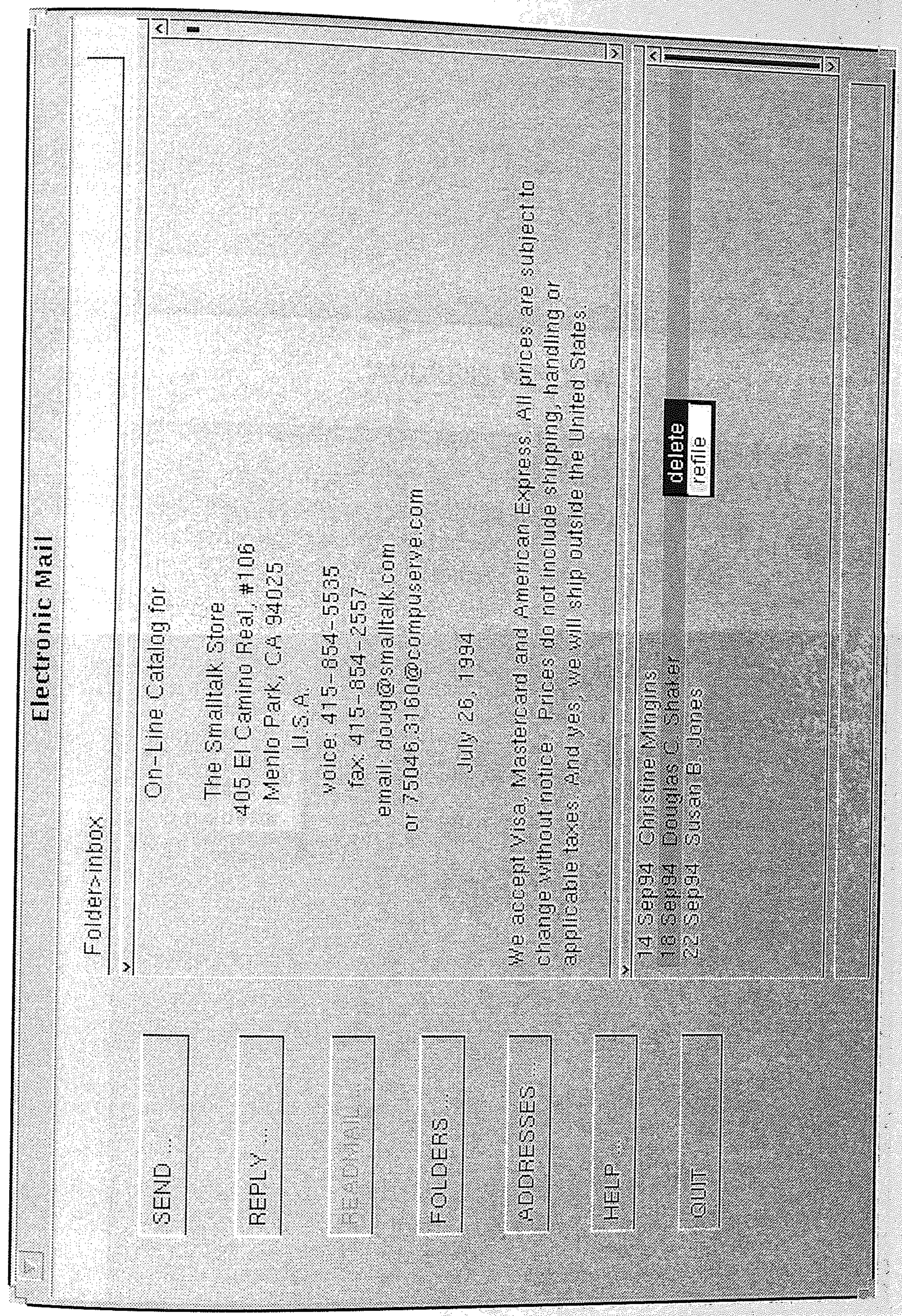
- There is not much different between the read-only views, buttons and write views (where you have to input your information)

- I didn't see the message at the bottom of the window. It is better to have the message just beside the buttons

- The name send is slightly confusing. The is no clear relation between the command send and the opening of window for you to write your message. I thought send will send my mail.

Appendix C1

ELECTRONIC MAIL WINDOWS



On-Line Catalog for

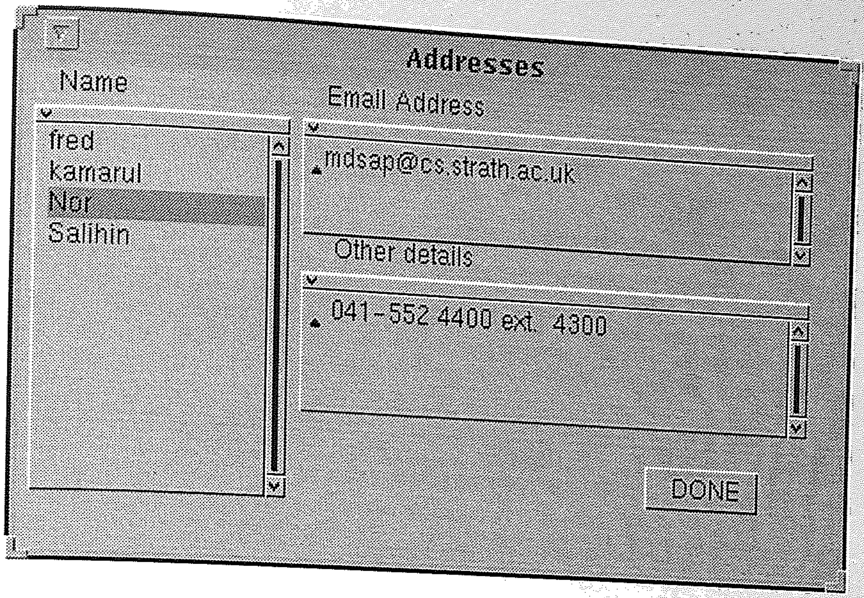
The Smalltalk Store
405 El Camino Real, #106
Menlo Park, CA 94025
U.S.A.
voice: 415-854-5535
fax: 415-854-2557
email: doug@smalltalk.com
or 75046.3160@compuserve.com

July 26, 1994

We accept Visa, Mastercard and American Express. All prices are subject to change without notice. Prices do not include shipping, handling or applicable taxes. And yes, we will ship outside the United States.

14 Sep94 Christine Mingins
18 Sep94 Douglas C. Shaker
22 Sep94 Susan B. Jones

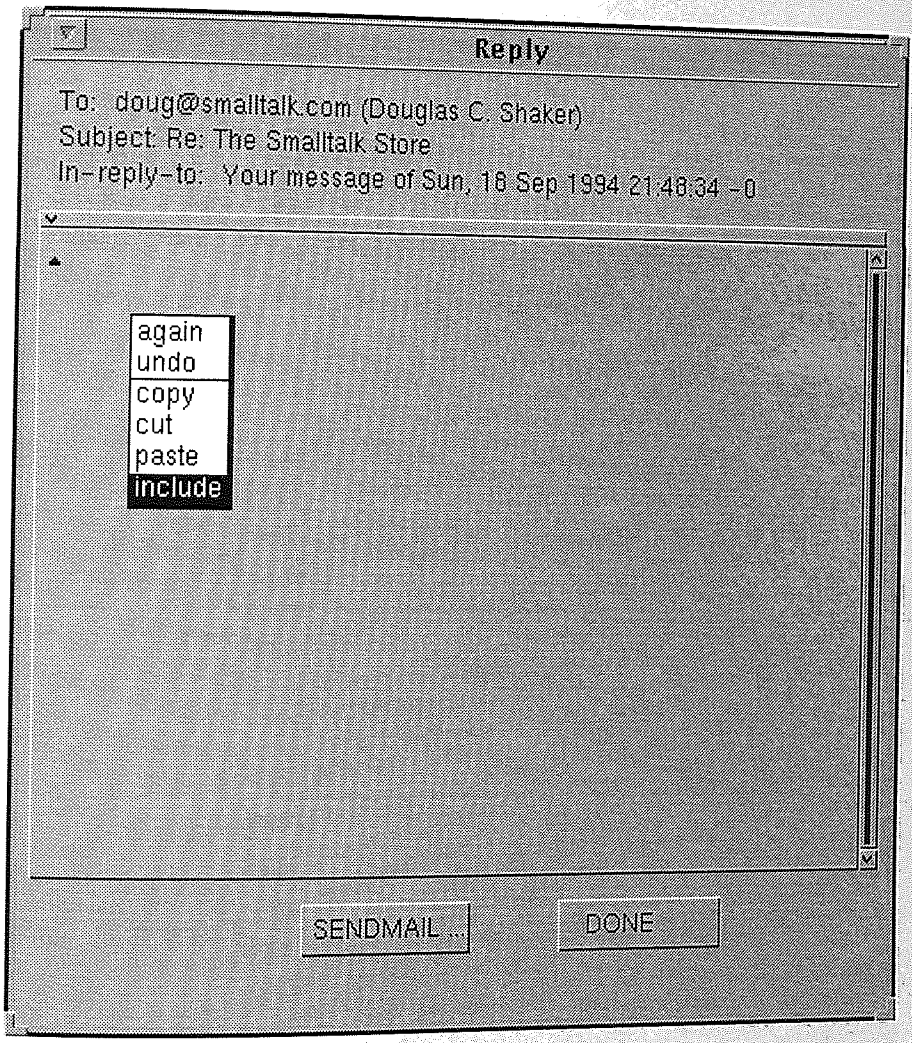
ELECTRONIC MAIL PRIMARY WINDOW



mdsap@cs.strath.ac.uk

041-552 4400 ext. 4300

ADDRESS WINDOW

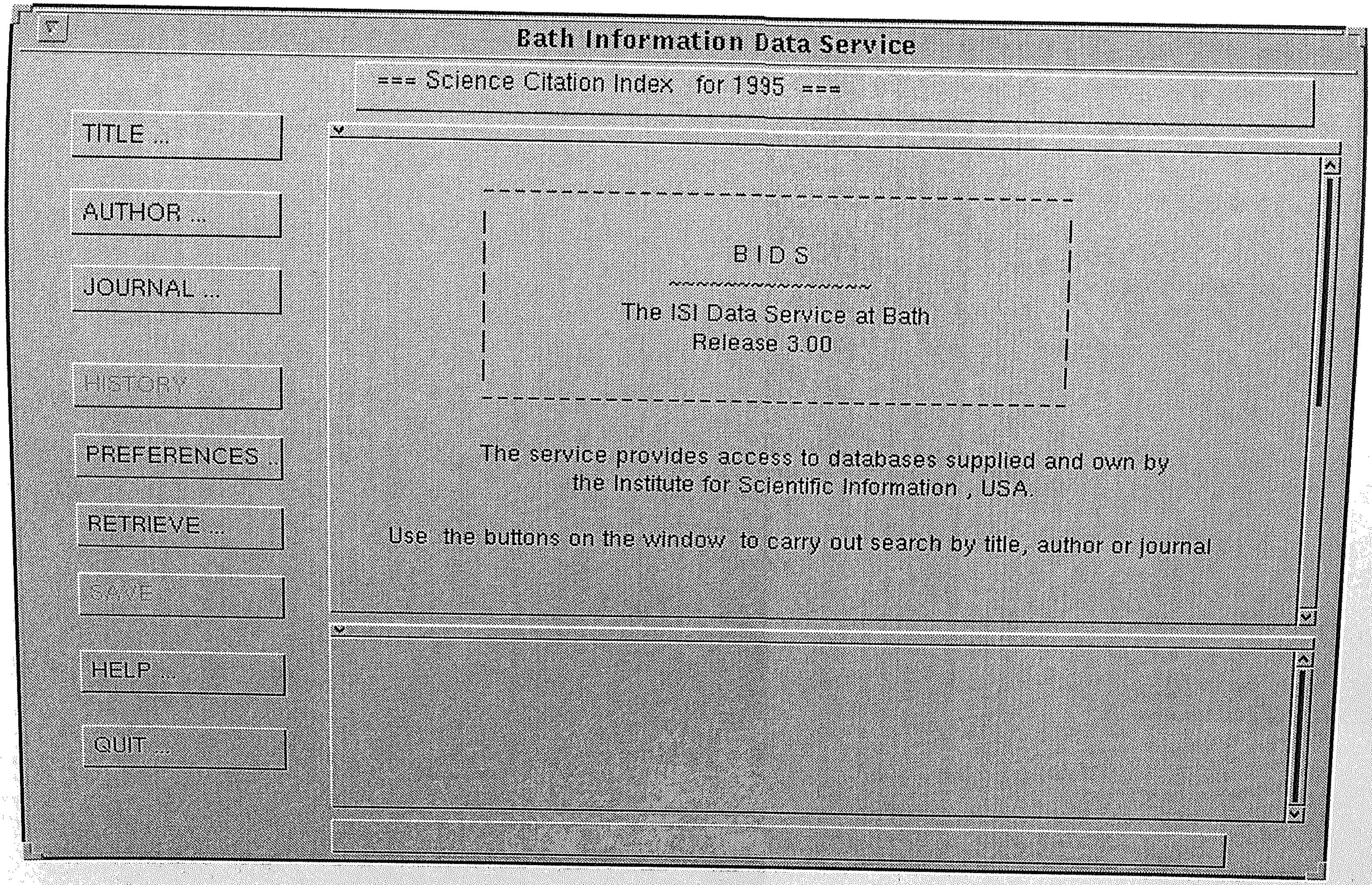


To: doug@smalltalk.com (Douglas C. Shaker)
Subject: Re: The Smalltalk Store
In-reply-to: Your message of Sun, 18 Sep 1994 21:48:34 -0

REPLY WINDOW

Appendix C2

BIDS WINDOWS



=== Science Citation Index for 1995 ===

B I D S

~~~~~~~~~~~~~~~~

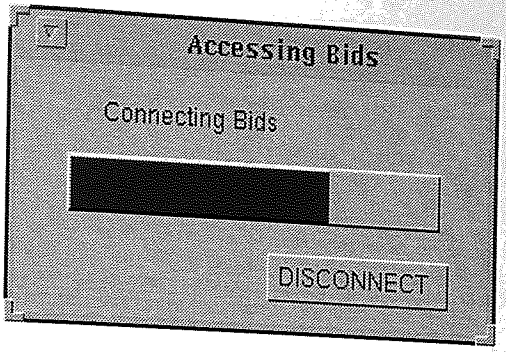
The ISI Data Service at Bath

Release 3.00

The service provides access to databases supplied and own by the Institute for Scientific Information , USA.

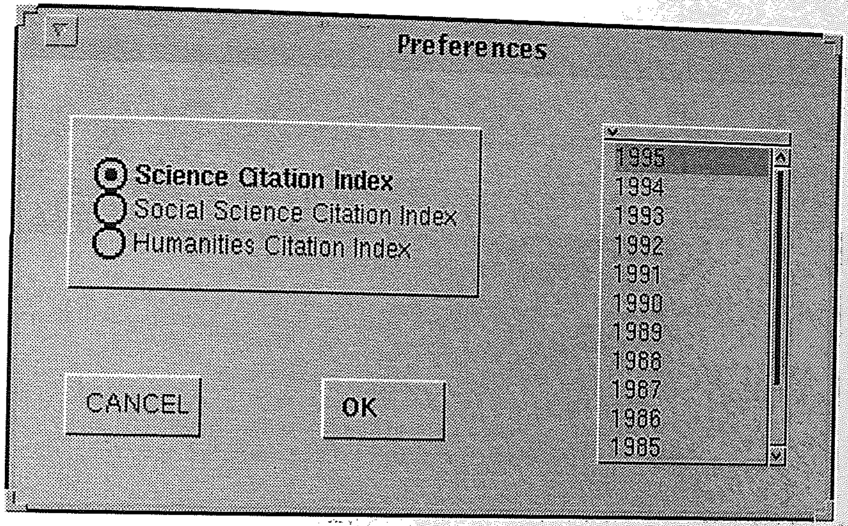
Use the buttons on the window to carry out search by title, author or journal

BIDS PRIMARY WINDOW
~~~~~~~~~~~~~~~~


Accessing Bids

Connecting Bids

STATUS WINDOW


Preferences

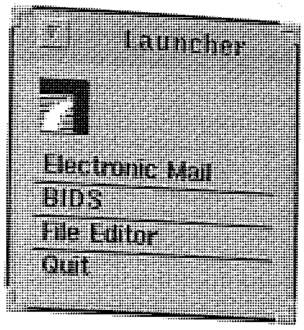
(●) Science Citation Index
( ) Social Science Citation Index
( ) Humanities Citation Index

1995
1994
1993
1992
1991
1990
1989
1988
1987
1986
1985

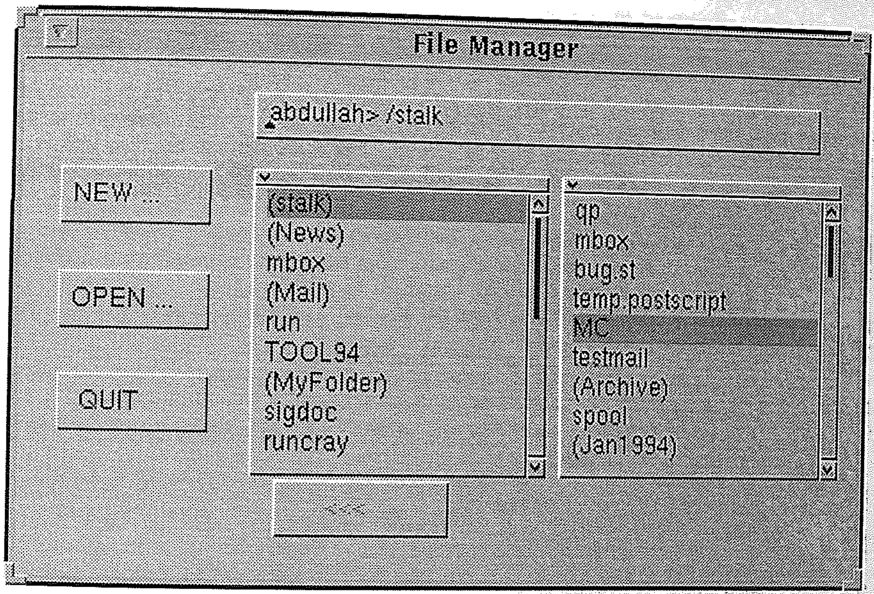
PREFERENCES WINDOW

Appendix C3

MAIN MENU AND FILE MANAGER



MAIN MENU



FILE MANAGER

Appendix D

CONFERENCE PAPER

Implementing an Interface to Networked Services

Abdul Hanan Abdullah[1] & Brian Gay[2]
Aston University, Birmingham, UK.
Email - [1]abdullah@aston.ac.uk; [2]bgay@aston.ac.uk

Abstract

This paper highlights the general problems and difficulties in using networked services. A prototype has been developed to help user interact with networked services. General design principles which arise in implementing a prototype user interface to networked services are discussed. The construction of the prototype is based on an object-oriented approach. The way it communicates with networked services and a help facility are also described.

1.0 Introduction

Networked services are a rapidly growing environment and they facilitate communication among users and encourage the creation and dissemination of information. Even though the services have undergone rapid changes, the development and implementation of methods of describing and accessing information are still lagging behind (Dillion *et al*., 1993). User friendly software tools are rare. As a result, locating, accessing and using information provided by networked services can be quite complicated even for computer experts.

An aspect of networked services that requires research attention is the user interface. Sadowsky (1993) points out that the current state of networked services is relatively primitive and tools to guide users to find items of significant interest are still

inadequate. A well designed user interface is required if an effective use of the service is to be achieved. The interface not only enables users to access various networked services, but also enables them to navigate the services.

A number of networked services provided in a university environment have been studied. The services provided can be divided into two: local and remote services.

Local services are provided by local hosts in the university. Users may access these services using computers connected to the university network from their own department. These services include electronic mail and access to the local university library. Electronic mail facilities allow users to send and receive electronic messages. By means of remote access users may use library services, such as accessing the catalogue, placing books on reserve and renewing loans. They are no longer required to visit the library building every time they require such services.

Remote services are provided by remote hosts via the internet. Similar to local services, users may access remote services from their department. Access to the services are usually via a specified local host. In the UK, such services include Bath Information Data Services (BIDS) and National Information Services and Systems (NISS).

The BIDS ISI Data Service is owned by the Institute for Scientific Information (ISI), USA. It provides access to four bibliographic citation databases. These databases contain details of articles drawn from over 7000 selected journals worldwide, and details of Scientific & Technical Proceedings of over 4000 conferences per year (Morrow, 1992).

NISS was established in 1984. NISS provides computer disseminated information for the UK academic community. NISS services are freely accessible for JANET users. The NISS gateway provides menu-driven access to a large number of online information services worldwide. For example, OPACs (Online Public Access Catalogues) is a service that gives users access to many library catalogues in the United Kingdom.

2.0 A Review of User Interface Problems

Some of the services were developed at the time when relatively little attention was paid to the design of the user interface. Most of the user interface designs were based on page screen display or scroll mode. Since the user interface part is embedded in the program code, reconstruction of the user interface is almost impossible.

Hence, users have to accept the difficulties in order to use the system.

Some networked services are old-fashioned menu-based interfaces even though they may be presented within a window of a graphical user interface. Menu systems are generally too modal. Users have to move from one menu to another in the hierarchy of menus in order to acquire the needed information. Users are not in control of the interface. Since most of the interactions use scroll mode, lengthy information may scroll up and pass the window.

To use networked services, users must first know on which machine the service resides. Then they must log on to the machine by invoking its unique name. Once logged on, different procedures need to be carried out to be connected to different services. BIDS users are usually required to log on to a special host in their local institution that supports a connectivity service to BIDS. Having logged on to the host, users have to log on to the X.29 (PAD) service. The PAD service functions as an interface between the users machine and X.25 public network. At the PAD prompt users have to enter the address of BIDS. BIDS will then request users to enter a valid identification and password before they can be connected to the service.

Graphical user interfaces (GUI) have been developed for some networked services. The use of the modern GUI technology alone is not enough if consideration of other aspects of user interface is neglected. For example, users must be given some kind of notification such as a status indicator or transformation of the pointer to indicate that the system is currently busy and not ready to accept any input from users.

Most of the developments of user interfaces to networked services are carried out independently by different developers. Thus, inconsistencies between the interfaces for different services are unavoidable. For example, most menu systems require the users to select their choices by typing their choice and then pressing return or enter, but in the case of NISS and Tin (a Usenet news reader), users just type the selection and it is automatically accepted by the system. As a result of inconsistencies, users have to keep on learning different styles of interaction every time they use a new service.

Inconsistencies among the user interfaces include the use of many different terms to refer to the same action. For example, different services use different terms to mean quit from the system. Examples of terms include exit, q (or quit), end, bye and logout. Thus, it is not surprising to see users leave the services without quitting after using them. They simply do not know how to quit.

Not all services support a help facility. The services which support the facility may not provide any indication that such a facility is available and the ways to access the facility are not the same from one service to another. As a result not many users consult the help facility even though they are having difficulties in using the services. They would rather consult their colleagues or attempt to use the services by trial and error or abandon the services completely.

3.0 Design Issues for User Interfaces and Network Services

Networked services are highly interactive applications and they require effective and efficient user interfaces. It has been established that properly designed graphical user interfaces are effective for interactive applications as they support both an intuitive presentation of information on the screen as well as easy interaction with the underlying system (Min Tjoa & Kappel, 1992).

This prototype has been developed to help users interact with networked services. It is a UNIX client that gives the user an easy and consistent interface with various networked services. Development was done on a Sun Sparcstation that runs Sun's operating system version 4.1.2.

The sections that follow discuss interaction mechanisms with the networked services adopted by the prototype.

3.1 Prototype Windows

Users interact with the prototype through windows. Windows create a sense of perceived stability because they provide a standard way for users to view and interact with different kinds of data (Apple, 1992). Since more than one window is usually required for the interaction and each one of them may behave differently, a proper management of windows is essential.

The prototype windows are mostly non-modal. Non-modal windows provide a flexible interaction with the application. Users can open more than one window and may work with more than one task simultaneously.

Windows of the prototype can be classified into two types: primary and secondary windows. A primary window is the main window opened by users when they invoke an application from the main menu. It is through the primary window that the main interaction between users and applications takes place. The prototype supports only one primary window and the interaction may be assisted by a number of secondary windows.

Figure 1 shows the primary window and one of the secondary windows of BIDS, one of the services currently supported by the prototype. The secondary window is opened when users invoke the preferences command by clicking one of the buttons in the window.

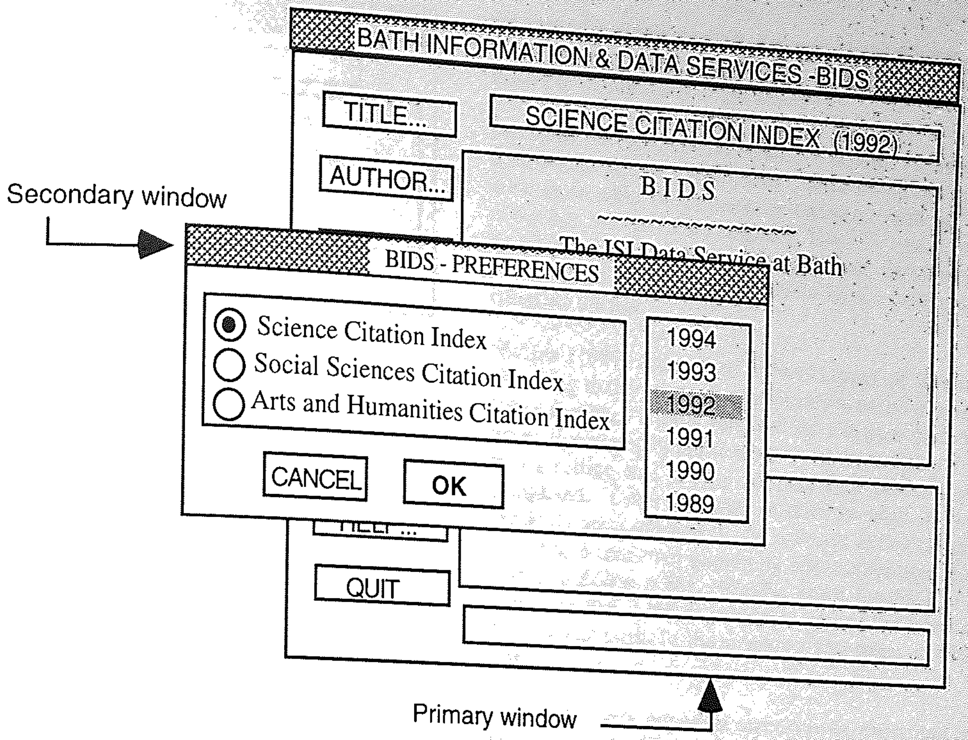

BATH INFORMATION & DATA SERVICES -BIDS
TITLE...
SCIENCE CITATION INDEX (1992)
AUTHOR...
BIDS
The ISI Data Service at Bath
Secondary window
BIDS - PREFERENCES
Science Citation Index
Social Sciences Citation Index
Arts and Humanities Citation Index
1994
1993
1992
1991
1990
1989
CANCEL
OK
QUIT
Primary window

Figure 1 : BIDS Windowing System

A secondary window is a window generated by a primary window. The relation between the main menu, primary window and secondary window is hierarchical, i.e., a primary window is generated from the main menu and a secondary window is generated from a primary window. The relation is shown in figure 2.

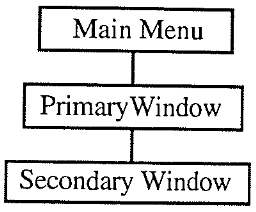


Figure 2 : Relation between Main Menu, Primary and Secondary Window

Even though the windows are hierarchical, user interaction is not modal. Users working with any non-modal window can simultaneously access any primary or secondary window. Users may also quit from any point in the system without having to back up through previous windows.

Some kind of coordination has to be established between dependent windows. Shneiderman (1992) suggests an automatic opening and closing of dependent windows. In the case of the prototype a number of secondary windows are defined as dependent windows. The Preferences Window in figure 1 is an example of a dependent window. This type of window closes with the primary window whenever users quit the application, i.e. by clicking the Quit Button on the primary window.

In some cases, a relationship between the window and application has to be established. In a text editing environment, for example, warning must be issued if users try to close the window without saving changes to the text. Another example is the window that contains a running process, such as a process that checks for any new mail. The process must be forced to terminate whenever users close the window.

Browsing through a collection of items with hierarchical structure can be made using a combination of two or more coordinated views or windows (Shneiderman, 1992). The Smalltalk-80 System Browser contains four scroll list views to help users browse through the class categories, classes and methods supported by the system. A scroll list view is a special view that holds a list from which users may select items by clicking on them.

The prototype uses a combination of two scroll list views to help users browse through the hierarchy of folders. The folder browser is shown in figure 3. The arrow buttons below the views help users to browse through the different levels of folders. The view at the top displays the pathname of a selected folder. Such a coordination of views may be used to review information arranged in a hierarchy such as the Usenet news.

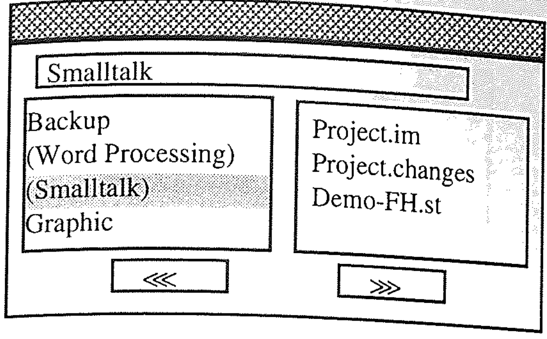


Figure 3 : A Folder Browser

3.2 Reversibility

Reversibility is widely used especially for text-editing and graphics by executing undo. The command reverses the effect of the user's previous operation. Such mechanisms for allowing users to reverse their action will encourage users to learn to use the application by exploration. For example, users may wish to experiment with various features supported by a text-editing application such as changing font size and using different page formatting. If by accident they delete a paragraph, they simply execute undo to recover the deleted paragraph.

An example of reversibility where users are given the opportunity to return to the state just before they activate the command is the confirmer (Hix & Hartson, 1993). This is a simple prompt from the interface that requests users to confirm their action. Figure 4 shows a confirmer before users send a mail message. Users have the choice, either to select 'yes' to confirm their action or press 'no' to reverse their action.

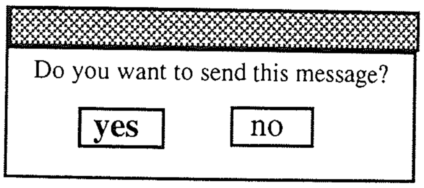


Figure 4 : A Confirmer

A confirmer is also used to warn users if they choose a command that may cause irretrievable data loss. This facility is essential because activating commands is so easy that users may accidentally choose a destructive command.

3.3 Feedback

Provision of immediate feedback is required in order to keep users aware of what is going on. Smith *et al.* (1990) state "It is disastrous to the user's model when you invoke an action and the system does nothing in response". Since no immediate feedback is provided by the system, some users of Openwin (Sun Microsystem, 1991) click an entry from the main menu several times. They keep on repeating their request because they are not sure whether the system has heard them or not. This situation results in multiple unwanted windows.

Feedback provided should be simple enough for users to understand (Apple, 1992). Most users would not know what to do when they encounter messages such as "The computer unexpectedly crashed. ID=13". It is more helpful if the message is well explained, for example, "Not enough memory was available for the computer to complete the task".

Nielsen (1987) categorises different types of feedback according to their degrees of persistence in the interface. Some feedback is only relevant for a certain period of time. For example, transformation of the image pointer into a rolling ball is only relevant until processing is completed. Other feedback is relevant until the user explicitly acknowledges it. An example in this category would be a confirmer requesting whether the user really wants to delete a file. Finally, some feedback is so important that it has to remain as part of the interface. An example might be the remaining free space on a hard disk.

Feedback is very important especially for operations that require a longer time to complete. After a long delay, the user may have forgotten to which action the machine is responding (Smith & Mosier, 1986). Card *et al.* (1991) provide the basic guideline regarding response time and the user's feeling and behaviour. The time limit versus the user's behaviour is summarised in table 1.

Time Limit (in sec)	User's Feeling/Behaviour
0.1	the system is reacting instantaneously
1.0	flow of thought stays uninterrupted
10	attention still focuses on the dialogue

Table 1 : Response Time Vs User's Behaviour

Myers (1985) suggests that percent-done progress indicators should be used for operations which are longer than 10 seconds. The advantages of using progress indicators are: (Nielsen, 1993)

- They indicate that the system is currently working on the user's problem.
- They indicate approximately how long the user is expected to wait.
- They provide something for the user to look at, thus making the wait less painful.

BIDS is accessible via JANET (the UK Academic Network) computer network and it takes more than two minutes to establish a connection to the service.

Since the waiting time is considered long, the prototype displays a status window on the screen when users activate BIDS. The window holds a status bar that displays the estimated length of time to complete the operation. The window also informs users from time to time about the different stages to establish the connection. Figure 5 shows BIDS progress indicator.

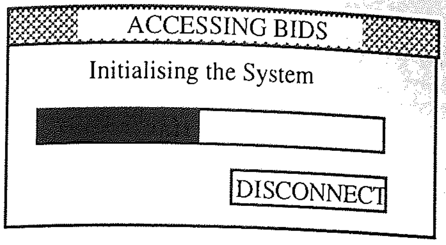


Figure 5 : BIDS Progress Indicator

BIDS may not be available at certain times of the day. This could be due to system maintenance. During a peak time, users have to be in a queue before the connection to BIDS can be established. The status window prompts a special message informing that the system is currently out of service or the users have to wait in a queue. The window provides users with the Disconnect Button to allow users to quit the service and try it later.

Using the percent-done progress indicator for operations which require between 2 and 10 seconds would violate the principle of display inertia (Nielsen, 1993). The indicator displays itself for a short while and then disappears. The display happens so fast that it causes irritation to the user.

3.4 Progressive Disclosure

Smith *et al.*(1990) state "Typically, a trade-off exists between easy novice use and efficient expert use. The two goals are not always compatible". Networked services are inherently complex, resulting in a user interface which is also complex. However, as Alan Kay (1977) suggests, user interface can be designed in such a way that simple tasks should be easy for the user, and complex tasks should be possible.

Sun Microsystems (1991) also suggests the concept of "progressive disclosure" so that the system looks simple but provides the advanced features desired by the advanced users. The concept is derived from the design of consumer appliances and stereo systems. For example, a stereo system has its complex controls placed behind panels.

Apple (1992) implements progressive disclosure by presenting the most common options in the dialogue box when it initially appears on the screen. The box holds a button named More Choices and when the users click the button, the box expands to display more information and the button name changes to Fewer Choices.

The prototype hides the complexity of the interface by delegating a group of related functions to a secondary window. For example, invoking a button named Folder causes a folder window to open. The window holds commands which are related to folders such as activating, creating and deleting folders.

3.5 User Guidance

Good user guidance results in faster task completion, fewer errors, greater user satisfaction and it permits the accomplishment of tasks which would otherwise be impossible for novice users (Magers, 1983). User guidance includes the provision of error messages, alarms, prompts, and labels as well as other instructional materials to help guide a user's interaction with a computer (Smith & Mosier, 1986).

An issue related to user guidance is choosing the most appropriate location to display messages. Smalltalk-80 version 4.0 displays messages on the Transcript Window. For example, if users try to search for a class which is not available in the library, a message 'nobody' is displayed on the window. There are a number of problems with the implementation. First, the message is not appropriate. Second, the Transcript Window is usually placed at a corner of the screen which is slightly away from users' attention. Third, if the window is closed or overlapped, users will not be aware of the message.

The prototype places the user guidance messages on the note view at the bottom of the window. The main task of this view is to help users using the interface. The note view can be broadly categorised into the following five functions:

- Describes the function of command buttons in brief.
- Helps users to complete a task.
- Advises/Navigates.
- Checks for errors.
- Indicates status.

The note view briefly describes the function of the button currently indicated by the pointer's position. Users can review the function of other buttons by moving the pointer onto the buttons. Every button is labelled according to its function. The label is usually a single word which does not precisely describe its functions. A label 'TITLE' may refer to a title of an article, a title of a book, a title of a conference, or a combination of any of them.

Completion of some tasks may require the user to execute more than one command in the right sequence. For example, the creation of an alias for electronic mail requires more than one step. After users have entered an alias, the next step is to include the real electronic mail address. Since the sequence of actions may not be clear to some users, the note view has to describe the next action in the sequence.

A similar interface design is implemented by Excel version 4. The software contains a panel at the bottom of the screen that describes different commands on the pull-down menus and guide users in using the system. Some tasks, such as building a graph, require the users to carry out more than one instruction. After users have selected the graph icon on the interface, the panel instructs users to drag the pointer to obtain the result. Such an instruction is a valuable piece of information for novices.

and infrequent users because it encourages them to explore the interface with confidence.

The note view displays errors made by users. For examples, users may try to delete folders that are not empty or create files using previously used names. The note view on the primary window displays a special message if users invoke a mouse button which is not used by the system. In a graphical user interface environment, invoking an inappropriate mouse button can be regarded as an error.

The note view is also used as status indicator. To retrieve information from a networked service may take a considerable length of time. The note view displays a counter showing the number of articles that have been retrieved. From the counter, users may estimate how much longer they have to wait.

4.0 Online Help

Sellen and Nicol (1990) list five main reasons why users avoid using help facilities: difficulty in finding information; failure to obtain relevant information; difficulty in switching between the help and the working context; complexity of the help interface; the quality and layout of help information.

When users need help, they must switch from a problem-solving mode to a learning mode. Clark (1981) found that the mode switch sometimes causes users to forget why help was requested in the first place. Houghton (1984) suggests that, one way of making help messages less disruptive is to place them on the screen simultaneously with the problem.

The solution to the problem is to develop a context-sensitive help, whereby help messages are displayed within the user's working context (Apple, 1992; Shneiderman, 1992). This type of help allows users to see problem and solution simultaneously, and the user's application remains active or in control. Therefore, users are not bothered by the mode change and are not required to memorise help messages before returning to their application.

Duffy *et al.* (1992) point out that users may not even realise that online help exists unless the means of access is explicitly spelled out on the screen. One of the buttons on the primary window of the prototype is labelled 'HELP'. The button reminds users to consult help whenever they encounter any problem. Activating this button causes a help window to be displayed. The window explains to users how to obtain the online help and provides an overview of the application software that aids users in effectively using the system.

When users press the middle mouse button on any button on the primary window, a single entry pop-up menu labelled 'HELP' appears. If users release the mouse button without moving the pointer away, a context-sensitive help window is displayed on the screen. Figure 6 shows the behaviour of the button when users press the middle mouse button.

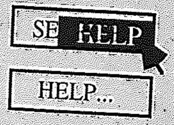


Figure 6 : Accessing HELP

5.0 Communication Channel

The prototype has to communicate with both the network services and the users. Hence, the interface itself can be divided into two parts;

- the *user interface* that manages the display of output and accepts user's input. The user interface also has to translate user's actions into commands that are understandable to networked services and transform the output from the communication channel before it is displayed on various views on a window.

- the *communication channel* that sends and receives data from networked services through the computer port. The channel has to read data at an appropriate speed, otherwise, some of the data would be lost.

A communication channel is created whenever users invoke a networked service. More than one communication channel is created if many services are accessed at a time. A communication channel is a child process that runs concurrently with the user interface process. It is created using a fork system call that executes the UNIX C Shell program.

Figure 7 shows the communication between the user interface and communication channel process. The implementation is carried out using an object-oriented approach where both the user interface program and communication channel are objects. Objects communicate by sending messages. The communication channel provides two methods for receiving and sending messages. The methods are:

- sendAll:(arg) - accepts arguments(user input) provided by the sender and sends it to the networked service.

- getData - The information received from the networked service is stored in one of the instance variables of the communication channel. The getData message returns the information to the sender.

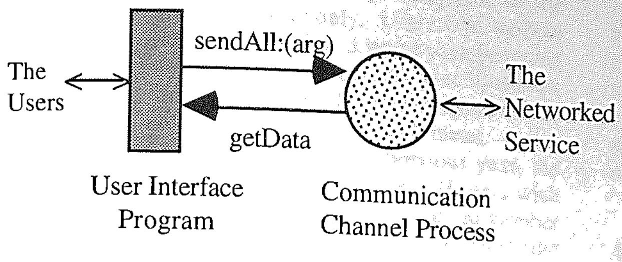


Figure 7 : Communication between User Interface Program and Communication Channel

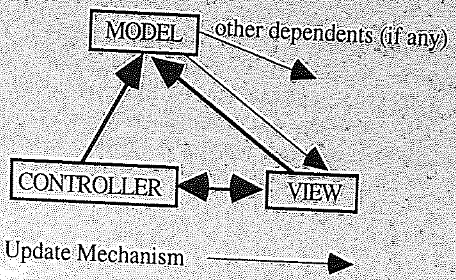


Figure 8: The Model-View-Controller

6.0 Object-Oriented Approach and Smalltalk-80

The prototype is developed using an object-oriented language, Smalltalk-80 (Goldberg & Robson, 1983). Smalltalk is not just a language, but a complete programming environment. Users work within an object-oriented framework of browsing the class library, deleting objects, experimenting and testing them out and incorporating them into their applications. More importantly, Smalltalk forces users to adopt the object-oriented approach (Thomas, 1989).

As far as reusability is concerned, the different Smalltalk systems are better than any other object-oriented languages. Smalltalk provides an extensive class library which permits rapid creation of new classes by inheritance (Bourne, 1992). An interface can be built more quickly using pre-existing library objects such as menu, scroll bar, buttons and dialogue boxes to construct an application. Objects created for one application can be reused as building blocks for other applications.

The prototype utilises the Smalltalk-80 Model-View-Controller (MVC) framework for the construction of its interface. The methodology provides a powerful tool that allows an easy and systematic development of an interactive application. The model deals with the underlying functionality, the view presents the model on the display screen and the controller manages the interaction of the system with the user.

Communication among the components of MVC is shown in figure 8. The communication is handled by sending messages and by the dependency mechanism of Smalltalk. The view and controller are tightly coupled. The view stores an instance variable that points to its controller and vice versa. The view and the controller also store an instance variable pointing to the model. This means that both the view and the controller may directly access information about each other and the model.

The Smalltalk MVC methodology is a powerful tool that allows an easy and systematic development of an interactive application. Most of the components required to build an application based on this methodology are available in the Smalltalk library. The same methodology is also used by Smalltalk itself for implementing the tools of the programming environment, for example, the System Browser. The System Browser consists of five views, and its information model is the library of Smalltalk classes.

Reusability of software components is one of the major benefits of structured and object-oriented programming (Graham, 1991). Initially, programmers have to invest some time reading and testing existing code. They may reuse this code in the construction of their application rather than writing program from scratch. They may also create new classes by refining some of the existing classes. The amount of programming by refinement is proportional to the amount of difference between the desired behaviour and the behaviour of existing classes that are being refined (Goldberg & Pope, 1989). In mature object-oriented systems such as Smalltalk, the amount is quite small.

7.0 Discussion

The implementation of the prototype is completely independent of networked services. There is a clean separation between the prototype and application codes of networked services. The prototype does not need to know the detailed implementation of the service and can be altered without having any adverse effect on the application codes.

The prototype provides an easy access to various networked services. The prototype displays most of the networked services available on the main menu and users may invoke any of the applications simply by selecting the right item. The prototype carries out the task of establishing the connection to the service. This simplifies the learning process because users have only one process to deal with, i.e., selecting an application, as opposed to keying in a variety of commands to establish the connection. Furthermore, mistakes are greatly reduced because the users are no longer required to key in a command.

The prototype can solve some of the limitations of the networked services without changing their application code. For example, BIDS users are connected to the database for the selected year only. Users who wish to search for all articles for the last 5 years have to repeat the current search for the previous year 4 times. Repeating the current search for the previous year can be carried out by selecting **z** from the search menu. Every time users carry out a search for the previous year, the current year will be set to the year before. If users wish to make another search for all articles for the last number of years, they have to reset the current year to 1994 (the current year) and keep on repeating the search a number of times. The option menu can be used to change various settings, including switching the current year.

The prototype allows users to define the range of years for which the search is effective. Users may specify the range by using the Preference Window (see figure 1). They may select the range by using the mouse. The prototype automatically carries out the search for the years that are specified by users.

The prototype integrates different application programs into a single application. The integration provides users with a consistent method of accessing and using the services. For example, the existing library and BIDS services do not provide a consistent way of accepting input when users carry out a search. The library users may specify the surname of the author if the initials are unknown, but BIDS users have to specify the surname and append it with ' _*'. The prototype ensures a consistent input to these services by scanning user input to BIDS. If the characters '_*' are not appended by the users, the prototype appends it to the input before executing the search.

An evaluation was carried out on the prototype. Users were given a number of tasks to execute. When they were asked to search by author whose surname was "Colomb" using the BIDS services, most users simply input the name without bothering to check the input syntax. Since typing the surname alone is acceptable by the library service, they just assume it is the same for BIDS.

Integrating the services provides the opportunity for services to communicate with one another. The BIDS services display a list of articles in different journals if a search is successful. Users then have to log on to the library services if they wish to check whether the journals listed are available in the library. They have to search for the journals one by one. If the services are integrated, the search can be carried out automatically by the prototype. This facility is not yet implemented by the prototype.

Smalltalk is not easy to learn. This statement is based on the authors' experience and it is in agreement with other Smalltalk users (Diederick & Milton, 1987; Nielsen & Richard, 1989; Hix & Hartson, 1993). It takes about 3 months for programmers who are familiar with a procedural approach to become proficient in Smalltalk. Once learnt, the flexibility and expressive power of the language as well as a large class library of predefined classes make the development of the prototype easier and faster.

Features of object-orientation are effective in containing changes. Changes can be made quickly without disrupting other parts of the application. Such a capability is essential because the prototype has to accommodate changes whenever there are changes to network configuration or networked services.

Acknowledgements

The authors would like to thank anonymous SIGDOC'94 reviewers, Mary Finean and Sharon Penfold for helpful comments on earlier drafts of this paper. Abdul Hanan Abdullah's research is sponsored by the Government of Malaysia.

References

Apple Computer, (1992), *Macintosh Human Interface Guidelines, Addison-Wesley.*

Card, S. K., Robertson, G.G., & Mackinlay, J.D. (1991), "The Information Visualizer: An Information workspace", *Proceeding of ACM CHI'91 Conference,* New Orleans, pp. 181-188.

Diederich, J. & Milton, J. (1987), "Experimental Prototyping in Smalltalk", *IEEE Software* 4(3), pp. 50-64.

Clark, I. A. (1981), "Software Simulation as a Tool for Usable Product Design", IBM System Journal, **20**(2), p.2.

Dillon, M., Jul, M., Burge M. and Hickey, C. (1993), "Assessing Information on the Internet: Toward Providing Library Services for Computer-Mediated Communication", *Internet Research* **3**(1), pp. 64-69.

Duffy, T.M., Palmer, J.E. & Mehlenbacher, B. (1992), *Online Help: Design and Evaluation,* Ablex.

Goldberg, A & Pope, S.T. (1983), "Object-Oriented Programming is not Enough", *American Programmer* **2**(7&8), pp. 46-59.

Goldberg, A & Robson, D. (1983), *Smalltalk-80 The Language and its Implementation*, Addison-Wesley.

Graham, I. (1991), *Object-Oriented Methods*, Addison-Wesley.

Hix, D & Hartson, R (1993), *Developing User Interfaces: Ensuring Usability through Products and Process*, Wiley.

Houghton, R.C. (1984), "Online Help Systems: A Conspectus", *Communications of ACM* **27**(2), pp. 126-133.

Magers, C.S. (1983), "An Experimental Evaluation of online HELP for non-programmers. *In Proceeding of*

CHI'83 Human Factors in Computing Systems, pp. 277-281.

Min Tjoa & Kappel, G. A. (1992), "State of Art and Open Issues on Graphical User Interfaces for Object-Oriented Systems", *Information and Software Technology* **34**(11), pp. 721-730.

Morrow, T. (1992), "Bids ISI - A New National Bibliographic Data Service for the UK Academic Community", *Computer Networks and ISDN Systems* **25**(4-5), pp. 448-453.

Myers, B.A. (1985), "The importance of Percent-Done Progress Indicator for computer-human interfaces", *Proceeding of ACM CHI'85 Conference.*, San Francisco, pp. 11-17.

Nielsen, J. (1987), "Classification of Dialog Technique", *ACM SIGCHI Bulletin* **19**(2), pp. 30-35.

Nielsen, J. (1993), *Usability Engineering*, Academic Press.

Nielsen, J. & Richards, J.T. (1989), "The Experience of Learning and Using Smalltalk", *IEEE Software* **6**(3), pp. 73-77.

Sadowsky, G. (1993), "Network Connectivity for Developing Countries", *Communications of the ACM* **36**(8), pp. 42-47.

Sellen, A. & Nicol, A. (1990), Building User-Centered On-Line Help. In Laurel, B. ed. *The Art of Human-Computer Interface Design.* Reading, MA: Addison-Wesley.

Shneiderman, B. (1992), *Designing the User Interface: Strategies for Effective Human-Computer Interaction*: 2nd ed. Addison-Wesley.

Smith, D.C., Irby, C., Kimball, R., Verplank, B. & Harslem, E. (1990), "Designing the Star User Interface", in Preece, J. & Keller, L.(eds), *Human-Computer Interaction*, Prentice-Hall-Open University, pp. 238-259.

Smith, S. L. & Mosier, J.N. (1986), *Design Guidelines for Designing User Interface Software. Technical Report* MTR-10090, The MITRE Corporation, Bedford, USA.

Sun Microsystems, inc. (1991), *OpenWin : User's Guide,* Addison-Wesley.

Thomas, D. (1989), "What 's in an Object?", Byte **14**(3), pp. 231-240.