Menu-Based User Interface Systems: Theory & Practice

by

Djilali IDOUGHI

A Thesis Submitted for the Degree

of

Master of Science

in the

Department of Computing Science

at

The University of Glasgow

October 1988

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. A.C. Kilgour for his assistance and advice during the last past year. Thanks also go to Kevin Waite and Cathy Wood for their help and their kindness and respect they showed to me and my family. My gratitude and all my respect must go to the Algerian government, without whose educational policy and encouragement this present study would not have been possible. Finally, my gratitude and affection go to my wife, Sadika and my daughter Schehinez, for their patience, assistance and support. during the last two years.

Dedication

To my wife Sadika
and our
darling daughter Schehinez

Summary

The thesis discusses the menu selection technique, which is one of the most commonly used interaction techniques in Human-Computer Interfaces, and continues to flourish because of its simple interaction format and its adaptability to the many diverse applications. The ease of use of the technique, particularly by novices, contributes significantly to the widespread acceptance of menu-based user interface systems, despite their inherent disadvantages and drawbacks. Chapter One surveys the issues concerning the design and use of menu-based interfaces, and addresses particularly the navigational problems encountered by users of menu selection systems, identifying various navigational aids which help overcome these problems. The chapter concludes with a comparison between menu-based interfaces and other interface styles (command language, natural language and form-filling).

Chapter Two describes the practical work of the thesis which consists of implementing a particularly demanding menu-based interface example involving multiple menu selections using four different dialogue specification systems. The implementation is discussed mainly from a menu system designer's view. Strategies to solve or address the multiple selection mechanism problem as well as some the navigational concepts discussed in chapter one are devised and used within each the four target systems. Also, some other related user interface design issues are reported in chapter two.

The principal aim of the work is to investigate the difficulties a dialogue designer may face in attempting to implement a common type of menu-based interface using various delivery systems, all of which claim in varying degrees to support menu-based interactive styles. In the final chapter conclusions are drawn from the practical work concerning desirable menu support features in user interface implementation systems, and issues requiring further investigation are identified.

Menu-Based User Interface Systems: Theory and Practice

Contents page

Chapter 1. **Menu selection systems**

Chapter 2. **Experimental studies in system use**

Chapter 1
Menu Selection Systems

1. Introduction

In the early development of the computer industry, effort, research and money were concentrated on the development and sophistication of the machine's internals and programming languages and to the efficient use of the cpu and storage media. Early users were necessarily computer technicians and professionals through whom other users had to go in order to access the remote computer. As computer technology has grown faster and become widely available, and costs have become lower, many areas such as the commercial, medical and educational spheres have exploited this technology for different purposes. The next stage has been marked by a closer move of the computers toward human society in which they occupy a big place nowadays. There has been a considerable growth in the number of users without formal training in programming or computer technology. These users are simply using the computer as a tool, and are not interested in becoming computer professionals or in understanding the details of their application systems. However, although most computer systems are designed to run essentially autonomously, most provide a means through which human users and the computer can communicate. This means is nowadays known as the *User Interface* .

So, human users and computers communicate through the user interface whose primary role is to support information exchange between users and computers. Many names have been assigned to the communication process. These include *Man-Machine*

Communication, *Man-Machine Dialogue*, *Human-Computer Dialogue* and finally *Human Computer Interaction*. This stage has also been characterised by the fact that despite the degree of sophistication of the machinery and the elaborateness of many computer systems, problems have arisen at the user interface which have seriously undermined the effectiveness of the computer as a tool for human problem-solving. Most of these problems are directly related to the underlying dialogue between the human user and the computer system, and have arisen principally because of the lack of attention paid by the system designers to effective human computer interfaces.

This can be considered as the starting point of a new era in which greater attention is paid to the issues which guarantee high quality user interfaces, and in which research effort is focussed on attempts to understand the complex interaction of humans and machines. Contributions to this research are required from different disciplines such as psychology, human factors, ergonomics and related fields, and taken together these constitute the area which is now known as the Human-Computer Interaction. Already considerable progress has been made and important findings reported in this new area.

All the HCI specialists were unanimous about the need for user interface improvements because of the crucial effects of the interface on user efficiency and the acceptability and therefore commercial potential of the computer system. One of their major findings was that the human user has to be taken into account as well as the computer system in the design process. Previously the emphasis was on hardware developments, but now the emphasis is

shifting toward human concerns. The result of this change of emphasis is that greater efforts are being made to make computers easier to use and program by providing better programming languages, better program development environments. These may include User Interface Management Systems (UIMS) which are intended to free the applications programmer from low-level details so as to be able to concentrate on higher applications specific aspects of the User Interface, i.e to separate the details of user interaction from the details of advanced applications [Buxton et al., 83; Bennett, 86]. Generally, a UIMS consists of a package of tools which support the implementation, debugging and evaluation of interactive human-computer dialogues [Buxton et al., 83].

The User Interface may be thought of as surface through which data are passed back and forth between computer and user, where the data displayed on the workstation provide a context for interaction [Bennett, 86]. The interaction part of the User Interface is important since it represents the communication path between the user and computer. Users' interests are not, in general, concerned with programming but with the utility of the end product. This will often depend on how easy the system is to use and is particularly influenced by its User Interface. There are many techniques commonly used for communication between humans and computers. They differ widely in their ease of learning and use and their general applicability [Brown, 82]. For example, the interface to Unix is very different from the interface to a Macintosh. The computer system interface imposes a certain dialogue style on the user. Among the important and commonly used communication techniques are the following:

- *Command Interfaces* : The user types instructions to the computer in a formally defined command language.
- *Natural language Interfaces* : The user's command language is a significant, well defined subset of some natural language such as English.
- *Form-Filling Interfaces* : The user issues commands by filling in fields in one or more forms displayed on the screen.
- *Direct Manipulation Interfaces* : The user manipulates through a language of button pushes and movement of a pointing device such as a mouse, a graphic representation of the underlying data.
- *Menu-Based Interfaces* : The user issues commands by selecting in sequence choices from among displayed alternatives. This is the form of communication or interaction which is the subject of this thesis.

The literature on the interaction between computers and human users is large and varied. Therefore, the scope of this thesis is limited to those elements which relate directly to the design and implementation of one particular type of user interface: the *Menu-Based User Interface* .

A broad survey of the characteristics of menu systems as well the underlying issues involved in a Menu-Based User Interface design are presented in the following sections of this Chapter.

2. Characteristics of Menu-Based Systems

The dialogue part of the Human-Computer Interaction represents the central aspect of any interactive system. For many dialogues, the exchange of information can be characterised in terms of its style, structure and content [Hammond et al., 84]. Only the

menu selection style is fully discussed in this chapter since it characterises any menu system. A menu-based system or menu selection system is a system where each user response is predicated on a set of choices provided by the system. The user is presented with a sequence of menus, each containing some descriptive text and a list of items (options).

The user responds by selecting one item, causing the system to perform an action associated with that item selection. Menu systems offer a simple interaction format that is adaptable to many diverse applications. They are primarily used to present information and to control the actions of computer systems. The user interface associated with or provided by these systems is said to be menu-driven in the sense that the user is guided and assisted in the decision making process or problem solving task. This form of interaction has several characteristics, including the following: (i) neither formal training nor memorisation of complex command sequences are required; (ii) it offers a simple selection mechanism via some pointing devices (mouse, keystroke); (iii) it simplifies choice by structuring the user's decision making, thus reducing the risk of making errors. Therefore, menu based systems appear to be more appropriate to novice or casual users, and menu interfaces have become increasingly popular during the last decade as a means of making the computer more accessible to those with little experience and/or those who use systems infrequently.

However, if a menu system is well and carefully designed, it can be appealing to experts as well. As the title of the thesis implies, the key word is "menu". It represents the central component of a menu system. Before considering menu system design issues in

detail, a brief description of the categories or types of menus is given.

2.1. Menu Categories

Menus can be categorised as either *explicit* or *embedded* [Koved & Shneiderman, 86]. The difference lies in the context in which the menu items are presented. Explicit menus are themselves subcategorised. Each of these categories is briefly discussed next.

2.1.1. Explicit menus

These are usually characterised by an explicitly enumerated list of items from which the user selects using one of the selection mechanisms available. Till recently, a linear organisation of the menu items was the assumed format in this category. Recently PIE menus, in which the items are arranged circularly, have been introduced [Hopkins et al., 87]. For linear menus, a variety of types may be distinguished, including

- *Pop-up and Pull down menus*, that is menus which appear below a fixed label on the screen (pull down), or anywhere within a fixed area, occasionally the whole screen (pop-up), in response to a click of a pointing device.
- *Permanent menus*, that is menus which are permanently displayed so always available to the user.

In general, linear menus are a linear row or column of items.

PIE menus are normally of the pop-up variety. The menu items are positioned in a circle around the menu centre. The direction in which the cursor is moved makes the selection, and the length of motion (i.e. the distance of the cursor from the centre of the

Pie) is available as a second input.

2.1.2. Embedded Menus

The menu items are embedded within the information being displayed on the screen to the user. In embedded menus, highlighted or underlined words or phrases within the text become the menu items and are selectable, using the commonly used touch screen, cursor and mouse methods. They are more appropriate in some situations where explicit menus are inefficient particularly in touch text, spelling checkers and language-based editors [Koved & Shneiderman, 86].

2.2. Menu Selection in the context of Hypertext

Hypertext is a concept, typically used within the electronic documentation domain, which allows non-linear organisation of the underlying material (text/graphics) of a document. It also provides a communication and thinking tool allowing authoring and design as well as reading and retrieving. A hypertext system has two main components: a database and a user interface to the database.

Hypertext systems use the menu selection technique as a fundamental mode of user control in navigating through the information space. A hypertext system may therefore be regarded as a menu selection system. However, the reverse is not always true. To qualify as a hypertext system, a menu system must exhibit the main hypertext features, which are the following:

- the database is a network of textual/graphical nodes
- windows on the screen correspond to nodes in the database on a one to one basis

- standard window operations are supported
- windows contain link icons which point to nodes in the database
- the user can easily create new nodes and new links to new nodes or to existing nodes
- the database can be browsed in three ways: link following, string searching and graphical browsing.

One of the most important characteristics of a hypertext system is its linking capabilities. Unlike selections in menu selection systems, links in hypertext systems can be of several functions and of different types. There are many systems which do not qualify as hypertext systems because of their lack of either the underlying database (eg. window systems) or the interface to the database (eg. DBMS).

Menu selection is a method of communication with a system, whereas hypertext is a tool using this method as its means of interaction. Finally, since a hypertext system is a menu system, therefore hypertext designers have to consider most of the design issues inherent in menu selection systems (discussed below) as well as those special to hypertext. When it comes to the use of any hypertext system, both writing and reading are allowed, but generally in separate modes normally called *author/writer* and *browser/reader* respectively. However, in considering the design and use of a menu-based system below, these two modes will be referred to as the *designer* and *user* modes respectively.

3. Menu System Design Issues

It is not yet known what are all the issues or factors that must be taken into account in order to achieve an effective menu

system, and less is known about what will guarantee the ease of learning and use of such system. However, many psychological, cognitive and human factors studies have been conducted in this direction and these have produced results which can be considered at this stage as guidelines. Many of these results are common to the design of interactive systems in general. Only those concerned with menu systems are discussed in this chapter. Design issues are considered in relation to the presentational, organisational and functional aspects of the interface.

3.1. Presentational issues

These issues are concerned with all the presentation aspects of the interface, that is how text, options and graphics should be presented to the user. Therefore, attention is focussed on the constituents of a menu.

3.1.1. Titling

Choosing a title for a menu is as difficult as choosing a title for a book [Shneiderman, 86]. Different menus need different titles, therefore choosing a consistent title for a menu becomes a serious issue to consider. The importance of this issue has been demonstrated by several studies. Titles can be used to help the user understand the context of the menu, and to indicate the distance (level) from the main menu, so reducing the disorientation problem in deep menus and enhancing the user's confidence. In a recent study on the effects of the presence/absence of menu titles (showing the path of previous selections) on the search time and accuracy, Gray [86] found that the subjects searched more accurately with

titled than with untitled menus, but titles gave no benefit in search time. Previous selections as titles could also be of great benefit from the navigation point of view [Apperley & Spence, 83]. Besides the contextual and navigational help aspects of menu titles, title placement is also an important parameter to consider. For example, left justification has been found to be preferable with slow display rates.

3.1.2. Menu Items

Menu items should fit logically into categories and have readily understood meanings so that users are confident in making their selections, and have a clear idea of what will happen when they make a choice. The design issues concerned with menu items are very important because the overall design of the menu system itself is based on them. The issues involved range from phrasing the menu items to sequencing and selecting them.

•Phrasing menu items

Menu items should be written such that comprehensibility, clarity and non-ambiguity are assured. This is not as simple as it appears to be. However, following some appropriate guidelines such as using familiar and consistent terminology, distinguishing between items and using consistent and concise phrasing may help lead to better results in user performance. A consequence of bad phrasing of the menu items is ambiguity, which is a major drawback in menu systems.

•*Sequencing menu items*

The second issue concerning menu items is the presentation issue, in other words how should the items be presented to the user. Should they be in alphabetical, logical (functional) or random order? If the items have a natural ordering sequence, the design decision is straightforward, but in other cases, the designer needs to choose between the major ordering sequences (alphabetical, functional and random). The importance of this issue has led to the investigation of the effect of item ordering on search performance. Card [1982] reported that people performed better with alphabetical arrangements than functional which in turn was better than random. Snowberry et al [83a] also found evidence that a categorical arrangement results in a more accurate and rapid search performance than a random arrangement. In contrast Schultz [87] found no significant overall advantage of alphabetical over random ordering of menu selections apart from during the initial blocks of trials, and then only when a deep structure was presented. The issue of sequencing or organising the menu items is directly relevant to the semantic organisation of the user's task.

•*Selecting menu items*

After the phrasing and ordering of menu items comes the issue of item selection, that is what kind of selection mechanism is suitable or appropriate for the menu items in question. This represents the central aspect of the menu system for most users. The major existing selection mechanisms are on-screen direct pointing (touch panel), off-screen pointer manipulation (mouse) and typed identification (keyboard). The most commonly used selection

technique is still the keyboard, despite the growing availability of the mouse on most recent workstations. Therefore, choosing the most appropriate selection technique for the task at hand becomes an issue. For systems using the keyboard as a means of item selection, the menu designer has to decide between different alternatives such as sequential numbering or lettering the items. Each of these has advantages as well as disadvantages depending on the task at hand and the user who is going to carry out this task. Perlman [84] studied the effects of type selector on the selection times (user think times) and found that compatible letters (a compatible letter is the first letter of the menu item it is paired with eg. p for print) were the best selectors followed by compatible numbers (a compatible number is the ordinal alphabetical position of the initial letter of the menu item eg. 4 for Debug) whereas for incompatible selectors, the trend was just reversed. Another advantage of compatible lettering is to permit typeahead selections (below). However, it was found that compatible lettering selectors were useful only if the designer has full control over the contents of menus (static menus). In other cases (dynamic menus) compatible lettering could lead to the worst case. Therefore incompatible (nonmnemonic) letters and numerical selections are preferable for dynamic menus.

With recent workstations, there is a tendency to use selection techniques other than keyboard, in particular the mouse, which has become the most used pointing device. This widespread use of a mouse might be expected to be motivated by the best selection performance. Surprisingly, however, Karat et al [86] have just proved the opposite. They found that the touch panel technique led to better performance, followed by the keyboard, and the mouse gave the

poorest performance and was the least preferred device. In Menu systems using the mouse as a pointing device, only a single selection mechanism, that is a rigid sequence of single selections have to be made before seeing a menu at lower levels of the menu structure. However, with the keyboard as a means of the selection technique some features that facilitate speed in a menu system can be used such as:

- *typeahead* : to go directly to a desired menu by typing in a sequence of type selectors (characters or numbers). This technique is also known as the *BLT* approach [Shneiderman, 86].
- use of menu names
- *macros* , which allow regularly used paths to be recorded and used as a single option when invoked.

Highlighting the menu items is another issue to consider in the menu system design process, but too little work has been done on the effects of different highlighting techniques.

3.1.3. Menu Layout

Beside the issues previously discussed, another important consideration in the presentation layer of the interface is the menu layout, that is how many items or how much information should be presented to the user and how menus are related together. As with most interactive systems, the screen (menu) display is a key component of successful design. Dense or cluttered displays can provoke anger, and inconsistent format can inhibit performance. Menus should be designed such that the information displayed provides cognitive assistance to the user. Since a screen is the predominant element of the user interface that a user comes in

contact with, many user activities are involved such as reading, visual scanning, remembering and recalling. Therefore all these processes become part of the screen design process. Information and layout considerations are design functions that impact on the ability of the user to scan and digest the screen content, and poor design contributes to user frustration and fatigue, and can inhibit performance [Hodgson et al., 85]. Screen layout design is a difficult task because the demands of each task and user community are so varied and difficult to measure. However, there are experimental findings and guidelines which can lead to sensible and acceptable design. The major principles are visual clarity and memory load optimisation. These two principles are both involved in the menu size issue. The effect of menu size on user performance has been demonstrated by several studies. Miller [81] and Snowberry et al [83a] found that search time and accuracy increased if the menu size was increased. Perlman [84] also found that menu size has a linear effect on the time it takes to find an item and this effect is larger if the list is random. There is also an effect of menu size on response time [Norman, 87]. And finally, the effect of menu size on the menu structure is also important (see next section).

3.2. Organisational issues

Unlike the menu system components previously discussed, which the user sees and deals with directly, the following components are not necessarily visible to the user. However, the issues within this "inner part" of the interface are as important as the presentational issues. The major issues are the way the menus (or rather the information composing the menus) are structured, and the

way they are accessed and navigated.

3.2.1. Menu Structures

In some situations, the task domain may need only single menus with one or more screens which consist mainly of some items of instructions to choose from, as in online quizzes and document processing packages, for example. However, even with these simplest menus, some of the presentational issues (discussed earlier) are still under the designer's consideration. An application requiring the user to make one decision at a time, such as selecting the print parameters in a document printing package, may need a linear sequence of menus to guide the user through this decision-making process. Other organisational issues relevant to such cases are concerned mostly with movement through the sequence of menus, for example moving backward, or forward and giving a clear sense of progress within this sequence.

For a relatively more complex task, where neither single menus nor a linear sequence of menus are appropriate, a more suitable way of guiding the user through the problem solving task is through the use of menu trees or hierarchically structured menus. Menu trees are primarily used to offer or provide a step-by-step guidance to the user. The menu structure can have one or more menu levels, each consisting of a set of items from which the user selects to proceed to the next level, and repeating the process till the user's goal is met. It is obvious that structuring the menus in a hierarchical manner needs great consideration of the presentational issues discussed earlier in order to assure better user performance and optimum use of the hierarchical structure.

One of the very important issues in designing hierarchical structures is the question of depth (number of levels) versus breadth (number of items per menu), i.e how many items each level should have for a given task. At least three effects can be attributed directly to the depth/breadth tradeoff. These are visual scanning, memory load and disorientation problems (which themselves represent important issues to consider in menu design). Many studies have demonstrated the importance of the depth versus breadth issue and have studied its effects on user performance. In two different studies which consisted of assessing user performance in retrieving items from four configurations (64 items with 1 level, 8 items with 2 levels, 4 items with 3 levels, 2 items with 6 levels) of a tree structured menu system containing 64 target items, Miller [81] and Snowberry et al [83a] found that the goal acquisition times were faster with the intermediate levels of breadth and slower with the extreme ones, while the accuracy decreased when the depth was increased, that is the deepest structure was the least accurate. These results suggest an advantage of a broader structure over a deeper one. However, it is not always appropriate to choose a broader structure for some applications where the depth alternative is not only an issue but a task requirement. In these cases it is necessary to provide additional support to reinforce the semantic grouping at all menu levels in order to facilitate performance accuracy.

Although tree structures are very appealing because they are the most natural structures for organising levels of abstraction, and the command-language for navigating them is simple, they suffer from the disadvantage that the tree structure is a function of the few specific criteria that are used to creating it [Conklin, 87], and it is

often necessary to force a hierarchical organisation upon a task or domain which does not fit logically and naturally into such an organisation. One solution is to allow the information elements or task components to be structured into multiple hierarchies and allow cross-references between them, resulting in a network structure. However, network structures may introduce new problems not found with hierarchical menus. The complexity of menu network structures may make the understanding or modification of the overall menu system virtually impossible [Brown, 82]. There may also be disorientation and lack of flexibility in the order in which the information is received by the user.

Some approaches have been devised to overcome the complexity problem of menu networks. Brown [82] adopted the approach called *structured subgraphs* and which is inspired by the top-down structured programming methodology. Part of this approach is discussed in Chapter Two when considering KMS, since it a typical example of a menu network system. Arthur [85] proposed an approach which is based on partitioning the conventional, monolithic frame (menu) network into a set of hierarchically structured, disjoint networks that preserve the original network topology while reducing its overall complexity and size.

There is no perfect menu structure that matches every person's knowledge of the application domain. The initial design of the structure can be motivated by some the principles discussed above, and can be improved over time to meet the user's and task requirements.

3.2.2. Menu Drawbacks

Although a menu system is relatively easy to write and implement compared with other interactive systems, it does not necessarily follow that this kind of system is the easiest to learn or interact with. Poor design in a menu system can lead to bad user performance because of the many problems that can be encountered by the user. These problems are likely to be of two main categories, namely problems caused by a poorly designed presentation layer, and problems of menu structure. A poorly designed presentation layer may involve cognitive mismatches caused by the organisation and categorisation of the information. The following problems form principally this category:

- ambiguity in choices or selections
- overlapping categories
- extraneous items
- conflicting classification in the same menu
- unfamiliar jargon
- generic items
- weak association between descriptor terms (higher levels) and target words (lower levels)
- visual scanning and memory load problems.

The problems related to traversal or movement in the menu structure may include:

- uncertainty in the users about their current position and about how to move to another state,
- artificially imposed hierarchies, that is hierarchical relationships between menu items where no real hierarchy exist
- inflexibility

3.2.3. Navigation Aids and Techniques

Two main solutions have been proposed to the problems mentioned above. To avoid problems due to cognitive mismatch, Shneiderman [86] suggested guidelines which are useful for semantic grouping in menu structures. These are:

- create groups of logically similar items
- form groups that cover all possibilities
- make sure that items are not overlapping
- use familiar terminology.

The other type of solutions consist mostly of a set of navigation aids and techniques that have been the results of many experimental studies. One suggestion that might solve some of the problems mentioned earlier is to increase the amount of information per menu [Miller, 81; Snowberry et al., 83a] but not to the extent of increasing the visual scanning and memory load problems and the response time. This is particularly relevant in menu systems with large and deep menu structures.

As the depth of a menu system structure grows and becomes larger, it becomes increasingly difficult for the user to maintain a sense of position in the menu structure and the risk of getting lost increases. Many menu systems have adopted different alternatives to overcome these problems. Some have adopted the method of an index such as: Prestel whereas some other systems use a map to show the underlying menu structure. Bellingsley [82], in a study on the effects of providing a map of the hierarchical structure and a semantically organised index, found that the presence of a map of the overall structure helped users develop a mental model of the underlying structure and led to a better performance over the index

method. An advantage for the map over other forms of training in menu learning has also been reported by Shneiderman [86]. It seems that offering a spatial map can help the user develop a better mental model and can thus assist in overcoming many of the problems above. However, other menu systems and their designers rely on other means and strategies developed to this end. Apperley et al [83] proposed some navigation techniques based on the following concepts.

- *stability*, that is the user should be given the possibility to cancel any choice and return to the state prior to its use by making the selections bistable (active and inactive).
- *awareness of state*, that is the current choices as well as the choices which led to the current ones (previous choices) should be displayed to the user allowing him/her to cancel, back up, and select again (incremental and selective retreat).
- *parameter nodes*, which permit a set of choices which are merely parameter definitions and which all lead to the same subsequent node to be replaced by a single parameter node. This is particularly convenient where the number of choices is large, and it also assists in avoiding artificially imposed hierarchies.

Other techniques proposed by Hepe et al [85] include:

- *instantiation*, which consists of displaying all subordinate nodes of each label after the label of the current node. Snowberry et al [85] called this upcoming selections. It is not only useful to increase the user's understanding of the category label but increases search accuracy as well.
- *sideways viewing*, which consists of displaying not only the upcoming selections (lower levels) but also selections from the

superior node to the other nodes at the same level (nearby levels). This enhances the user's confidence in selection, since it gives a better perception of the user's location within the menu structure.

3.3. Functional or Computational Issues

The attractiveness and acceptability of a menu system depends heavily on the speed at which users interact with the system, that is the pace of interaction. This is characterised by the system response time, and the display rate. These two factors are very important in menu system design because they influence other design issues such as user expectations, speed of task performance and error rates. Novice users prefer slower interaction together with more informative and complete displays, whereas more experienced users would prefer rapid interaction and less disruptive information. Rapid interaction can increase productivity and user performance but may also increase errors in consequence. Therefore, how can the designer choose the most appropriate interaction pace when each variable affects the other? In any case such a decision must be made to maximise user performance and satisfaction [Norman, 87]. The effects and the relationships between the different variables have been the subject of many different studies.

4. A descriptive/prescriptive model for menu-based interaction

Interactive system design is a very difficult task in general because it involves so many factors which the designer cannot pin down by an algorithm or a systematic method. Moreover, the interface requirements that support user interaction increase in sophistication and complexity, making the human-computer

interaction process even more difficult to understand. However, people who are concerned with designing and building interactive systems and have a lot of experience in this field have produced a number of ideas and suggestions which are purely the results of their long and rich experience. Guedj [80] suggested to setting up of guidelines intended to improve the quality of interactive systems, and since then many others have followed up this suggestion.

Another approach which offers better control over the interaction process is the use of models which can be formally specified in order to allow the precise description of the external behaviour of the system regardless of its internal implementation [Jacob, 83; Arthur, 86].

For menu-based user interface systems, Arthur [86] proposed a model that characterises menu-based interaction. It is designed to provide a basis for achieving understanding of the capabilities and the limitations of menu-based interaction systems. Arthur suggests that any menu system is minimally characterised by:

- a finite set of frames each consisting of a sequence of options
- a set of user responses
- a mapping from each frame/response pair to another frame.

Systems displaying only these three characteristics are said to be *information systems* or *information retrieval* systems, because their main function is to provide the users with information. Examples include most the videotext systems such as: Prestel and Ceefax. If these three characteristics are extended with a set of actions associated with frame item selection, then such systems are said to be *task-oriented* systems. Menu systems that provide user response facilities such as response reversal and item selection

histories need another discriminating element which is the incremental history sequence. Therefore, five discriminating elements have been identified to characterise menu systems. These five elements represent the model components. Any menu-based interaction can be modelled and specified by the following 5-tuple : M = (F, R, A, H, T) where,

- F is a finite set of frames
- R is a finite set of user responses
- A is a finite set of actions that support system and task-oriented operations
- H is a set of all sequences over $F \times R \times A$, that the set of all possible history sequences
- T is a transition that maps $F \times R \times H$ into $A \times F \times H$ as follows:

  let h be an element of H and define :

  app : $H \times (F \times R \times A) \rightarrow H$

  app (h, y) is the sequence obtained when the 3-tuple y is appended to H. $\forall$ f, f' $\in$ F, r $\in$ R, a $\in$ A and h, h' $\in$ H then

  T (f, r, h) = (a, f', h'), where h' = app (h, (f, r, a)).

This model represents a basic framework within which some important menu system concepts can be described such as :

- user movement within a menu system
- the incremental history sequence
- the current state of the menu system

The menu systems used in the practical work described in chapter 3 could all be described within this framework, but such a description is not presented here, since the main purpose of the practical work was to explore empirical properties of the various systems which are not captured by Arthur's model.

5. Menus vs other Interfaces

5.1. General

This section discusses the merits of three different kinds of interface (command language, natural language and form-filling) relative to the menu-based interface. Different communities of users with different needs may have different objectives in using the computer. The way the computer is used or exploited depends on the task, type and knowledge of the user. Different human-computer interfaces are needed for different groups of users. For example, text editing and interacting with an operating system are usually most appropriately achieved via command language interface, because of the wide range of capabilities and operations required by this type of application. Menu or natural language interfaces would not be appropriate for these applications. One of the most appropriate domains for a natural language interface is querying databases, where the query language consists mainly of a subset of a given natural language such as English.

5.2. Menus vs Command languages

Generally, command language interfaces are used by experienced and knowledgeable users in a task domain such as interaction with an operating system. Users can specify their operations directly simply by typing the names of the commands along with their parameters, and are therefore offered a vast range of possible intentions that can be realised, as well as freedom in accomplishing their goals within the capability of the system. With menu interfaces on the other hand, possible processing goals are completely prespecified in advance, and users need only select a

permissible sequence to accomplish their goals. Command language interfaces require users to learn and memorise several commands and there is usually no online reminder of the set of possible actions. This leads to many known problems and difficulties with these interfaces. Possible techniques for overcoming problems associated with the command language interface (such as the memorisation and learning problems) include :

- commands as prompts : this approach is close to but more compact than a standard numbered menu, and preserves screen space for task-related information.
- command menus : a list of descriptive items that can be selected by single letter presses. This is known as a hierarchical command language and analogous to the typeahead (BLT) approach to menu selection.
- pop-up or pull-down command menus : the menu items are commands which are selected via a pointing device, a mouse. The Apple Macintosh interface is a typical example.

From this perspective, a menu-based interface is an interface in which commands are presented via menus. This is a menu-driven interface. Norman [87] reported five attributes on which the comparison between these two types (menu-driven and non menu-driven) of interface could be based. These are as follows:

- *speed of use* : slow for large and hierarchically organised menu interfaces, but faster with command language interfaces.
- *prior knowledge required* : too much demanded with CL interfaces, whereas menu interfaces are in principle self-explanatory.
- *ease of learning* : high in menu interfaces because they involve

recognition rather than recall, and facilitate exploration and discovery of system options. In CL interfaces, on the other hand, learning is harder because of the numerous names and syntax to be memorised and recalled, and there is no simple way for exploring the system.

- *errors* : erroneous actions are difficult to determine and recover from in menu interfaces, whereas errors in illegal commands are easy to detect and correct.
- *most useful for* : menu interfaces are more suitable for beginners and infrequent users while CL interfaces are useful for expert users.

5.3. Menus vs Natural languages

It might be expected that communicating with the computer in a natural language such as English would be the most natural, simple and powerful human-computer interface. Many natural language interfaces have been designed and built, but applied only to specific domains where the results are not as satisfying as anticipated. A common application of natural language interfaces is in querying databases in which the query language consists of a subset of English that is translated by grammars to a formal query language such as SQL [Simmons, 86]. In a Natural Language Interface (NLI), the users are assumed to be knowledgeable about the task domain but intermittent about the syntactic details of the query [Shneiderman, 86]. However, like most human-computer interfaces such as menu and command language interfaces, NL interfaces present many problems and difficulties as well. Tennant et al [83] reported the following problems that NLI suffer from:

- typing and formulating questions in a way that the system can understand is necessary
- high failure rates which often frustrate users
- users often do not use features of the system because they are unaware of them
- systems are expensive to build and require a large amount of memory.

Simmons [86] associated the following additional problems with NLI:

- lack of feedback for misformulation of the queries
- user expectations are poorly met
- lack of understanding of human intentions

A possible solution to some of these problems is proposed by Thompson and which consists of the adoption of menu control [Simmons, 86], where users select whether to formulate an enquiry or to supply data. A menu then shows how a command may begin, and selecting an option causes a new menu to appear showing choices for possible continuations. This method keeps the user in the English subset and ensures that the user's queries remain in the semantic and pragmatic bounds of the system. Moreover, selection by mouse gives the added advantages of largely eliminating typing problems and ensures error free-use, accompanied by a satisfactory feedback showing the user the resulting translation to a simple formal language. This hybrid form of interface is called by Tennant et al [83], a menu-based natural language interface. In Tennant's comparison between conventional and menu-based natural interfaces the advantages of the menu-based approach over the conventional one are summarised as follows:

conventional	menu-based
10-15% failure rate	0% failure rate
typing required	selection through pointing
possible spelling errors	no spelling errors
hard to create a sentence	easy to recognise a sentence
1-30 man/months per application	1-30 man/hours per application
large memories	small memories

However, this does not mean that this approach will always be preferred, or will replace the conventional one in all circumstances because conventional natural language interfaces can cover more design possibilities within an application domain than are possible with menu based interfaces. Also there are many applications which either cannot be done with menu interfaces, or long and complex menu search requires more effort than typing.

5.4. Menus vs Form-filling

For some tasks, requesting the user to type in various values in various fields of a single display may be more appropriate than the use of menus. An interface that allows the use of a keyboard as a means for its input and the display of various fields in which the values and options are specified and entered is called a *form-filling interface.*

Menus and forms are both input mechanisms. The difference lies in the way input is used. Forms are integrators of information while menus are displays of discriminating alternatives [Perlman, 84]. A form can be viewed as a menu with random access of fields via cursor movements. Unlike values in menus, which are assumed to

be valid prior to selection, values in forms are validated. As with menu selection interfaces, form-filling interfaces also have their associated design guidelines [Shneiderman, 86].

6. Examples of menu systems

The best examples of menu systems are the systems known generically as of *videotext* or *viewdata*, in which the TV screen is used to display data or information organised into frames or pages. They are on-line information retrieval services. A typical version of videotext is given next under the name of Prestel.

6.1. Prestel

A Prestel database contains many thousands of information pages, where the key to information retrieval is the indexing method. A Prestel page is the smallest item of information which a user can address directly. A page is a screenful with up to ten links to any other page. Index pages are called routine pages which lead to end pages which contain information rather than routing choices. But each page can be extended over up to 26 additional display screenfuls, each called a frame. A frame is identified by its parent page number plus a following alphabetic character (a to z). Frames can only be reached via their parent pages. There are no jump or reverse procedures for finding frames. Frames permit a logical topic to be extended over more than the capacity of a single screenful. Prestel uses also the combined printed directory with the numbered choices approach.

As mentioned before, hypertext systems are also menu systems, and apart from the systems which are discussed in chapter two, the

following is also a typical hypertext and menu system.

6.2. TIES or HyperTIES

This is a typical example of a menu system which uses the hypertext approach. The University of Maryland Interactive Encyclopaedia System [Conklin, 87] is an information retrieval system which allows users to explore information resources in an easy and appealing manner. The basic units in the system are short articles which are interconnected by any number of links (selection). The links are highlighted words or phrases in the article text (embedded menus). The user activates the links by touching them by a finger or using the arrow keys to jump to them. Activating a link causes the article about that topic to appear in its own window on the screen.

The major purpose of this work and the thesis as a whole, is to investigate the functionality and limitations of a number of dialogue specification systems, from the point of view of a designer wishing to build a menu-based interface. Mostly the work involves implementation of a particularly demanding example involving multiple menu selections. And in order to get a deeper insight into the properties and limitations of these dialogue specification systems, practical work was carried out implementing sample problems and concepts. This work is described in the following chapter.

Chapter 2

Experimental Studies In System Use

1. Introduction

This chapter discusses the practical work which was carried out as part of the investigation of menu systems. The work consisted mainly of implementing some practical examples using four different menu-based dialogue specification systems. Each of these incorporates some important concepts and principles which give the underlying system its own type and style. However, the differences in type or style focussed on in this thesis are those relevant to menu-based user interface systems or hierarchically organised dialogues.

The practical examples to be implemented were chosen with the aim of highlighting the relationship between the underlying systems and hierarchically based systems. Additional aims were to discover whether the multiple menu selection mechanism adopted by some menu-driven systems such as the *Dining Out In Carlton* system described in [Hepe et al., 85] was achievable or not, to see whether the implementation of certain important navigational concepts was possible or not, and finally to investigate the extent to which the hypertext concept may influence the design of menu selection systems. A wide range of issues involved in the design of user interfaces arose in the course of implementing the examples using the target systems and these issues are discussed together with the difficulties and deficiencies encountered during those experiments.

The chapter is partitioned as follows: section 2 gives an outline of the examples as well as their special properties which represent the different sample problems to be implemented using the target systems, section 3 gives full details of each of the four experiments, including a detailed description of the different target systems used.

2. Sample problems

In chapter One, some drawbacks were mentioned which many menu systems suffer from and which represent the main disadvantages of such systems. Chapter One also described proposed solutions to these problems, several of which have been successfully applied in many applications [Hepe et al., 85; Apperley and Spence, 83; Apperley and Field, 84]. Typically, these techniques relate to the navigation around a given dialogue structure. In order to illustrate the importance of these techniques, the task of designing and prototyping a user interface to two menu-based examples was carried out. The examples chosen for this purpose were *Dining Out In Carlton* [Hepe et al., 85] and the *On Line Library* based on the CR classification scheme [CR, Acm press, 88].

The *Dining Out In Carlton* example gives scope for use of all the navigation aids and techniques discussed in Chapter One as well as a multiple selection mechanism, and the principal reason for choosing it was to investigate how easily these navigational techniques as well as its multiple selection property can be implemented in each of the selected dialogue specification systems. The descriptions of the two examples are given below.

2.1. Description of the "Dining Out In Carlton" example

It is a hierarchically organised information system. It was originally devised to provide information about restaurants for the experimental viewdata system described by Apperley and Field [84]. It operates at three levels :

- a menu of restaurant attributes,
- a list of available restaurants,
- the restaurant's information page.

The first menu consists of a set of attributes where the user's choice is a combination of selection of three attributes. This multiple menu selection scheme represents a special property of the example and which makes a challenge for conventional menu specification systems. The attributes used are *cuisine*, *location* and *price range*. This selection leads to the corresponding alphabetically ordered menu (page) where only one item is chosen or selected. Thereafter, the corresponding third level menu which represents the information page of the specific restaurant chosen is displayed. A valuable feature is also included allowing the user to bypass making a decision for all parameters in order to browse the available target space. This facility is known as the *Skip-to-target-level* option. One other valuable option named *any* which gives added freedom to the user who has a specific value of an attribute in mind but does not care about the other parameters in order to skip to the target level is also added. Attribute selection is achieved by the user pointing to the attribute name with a light pen. The selected attribute name is then highlighted.

As well as providing information, another purpose was to provide a working demonstration of most of the techniques employed to remove the inherent disadvantages of classical menu

based information systems. In the implementation discussed here, only the sideways viewing technique has been omitted (See [Hepe et al., 85] for more details).

2.2. Description of the On-Line library example

This example is typically based on the CR classification scheme [CR, Acm press, 88]. This scheme is mainly aimed to classify and structure all the information contained within the computing field. The classification scheme consists of two parts:

- a numbered tree containing unnumbered subject descriptors,
- a general terms list

The tree and subject descriptors

The tree consists of eleven first level options and one or two more numbered levels under each of these.
The set of children of all first and second level options begins with an option named "General" and ends with an other named "Miscellaneous". The first level options have letter designations (A through K) with numerals used for the second and third levels. A set of subject descriptors is associated with most leaves of the tree. These are essentially fourth level options intended to subdivide the subject area denoted by the leaves into subareas. Cross-references between the options within the tree structure are also supported in this scheme.

The General Terms list

Typically many areas of the computing field share a common set of General Terms. Therefore grouping reviews in CR

according to the General Terms is another way of organising the information retrieval task. Examples are: algorithms, design, etc... This general Terms list represent the keywords within this example.

3. The Systems

The four dialogue description systems considered for practical work in this thesis were: Chisl [Wood et al., 88], Guide [Brown, 86], KMS [Akscyn et al., 88] and HyperCard (released by Apple and developed by Bill Atkinson, [Apple Macintosh HyperCard User's Guide, 87]).

The very first step relating to the use of each system was to acquire and understand all the underlying features, concepts and mechanisms concerning the design of an eventual menu-based system. The respective outcomes of this step as well as the description of each system are discussed in each subsection of this chapter.

The idea behind the objective of carrying out the task of designing and prototyping a menu-based user interface to the examples chosen was not the use of the end products themselves, but rather the investigation and consideration of the underlying concepts which compose each of the target system and the way the design and implementation of the above examples are achieved. Each experience is discussed from both the designer's and the user's perspectives.

3.1. Chisl

1. Features of the Chisl specification.

Among the key features of Chisl are the following:

i. Chisl is a graphical dialogue specification language which allows:

- the creation of hierarchically organised dialogues,
- the dynamic reconfiguration of a dialogue specification without requiring recompilation.

ii. A dialogue consists of sequence of dialogue units hierarchically structured. Each dialogue unit is specified and stored in a separate file in a human readable form. The filename is used to identify the dialogue unit. A dialogue unit consists of a set of options which are selectable either by the user, the application program or Chisl itself.

iii. An option consists of:

- an option name: identifies the option and holds information about the option type,
- a location: the initial coordinates of a selectable screen object, but optional,
- a condition: a boolean expression such that if is evaluated as true, the option is selected,
- an action sequence: a list of actions carried out when the option is selected.

A dialogue unit may also have an entry action which will be carried out once when the dialogue unit is first activated.

The syntax of an option is as follows:

<selection condition>[<location>]<option name><action sequence>;

An option is either local or global.

- local: when declared or defined in a DU (Dialogue Unit), a local option is selectable or legal only in the DU in which it is declared. If while in a lower level dialogue unit a local option is chosen from further up the hierarchy, then the dialogue will back up to the chosen level of the selected option.
- global: a global option is exported to each DU called from the DU where the option is declared even if that DU is deactivated or exited. A local option, on the other hand, is selectable only as long as the DU in which it is declared remains active.

iv. The interpretation and execution of a dialogue specified in the Chisl language is performed by the Chisl interpreter *Chip*. The *execute* function of Chip is called recursively each time an activated dialogue unit is encountered within the selected option.

v. Chip uses a condition satisfier to evaluate the selection conditions of the options which are tested in the following order:

1. global option exported to the current level;
2. local options at the current level,
3. local options at successively higher levels along the activation path, back to the root.

The Chisl system can be classified as a hierarchically based dialogue system and appears to be well suited for the implementation of menu based user interface systems.

2. Experience of using the Chisl System.

This section describes the task of designing and implementing a menu based user interface to the *Dining Out In Carlton* example using the Chisl system. Two approaches are devised for this purpose which consist of using the Chisl specification language as well as a preprocessor.

2.1. Design and Implementation

This section discusses the design and use of the *Dining Out In Carlton* example using the Chisl system. The attribute values considered here for illustration are:

- Cuisine: French, Italian
- Location: Carlton, Abbeywell
- Price: 3-10, 10-15

Only a few attributes are considered in this example in order to generate small dialogue units.

2.1.1. Using Chisl specification language

The menu of attributes should be displayed first in order to allow the user to select three attributes in any order achieving therefore the special property of the *Dining Out In Carlton* example.

This is how it is done when using the specification language (See Appendix A for more details on the Chisl syntax). First of all, the display should be done by the "*Root*" dialogue unit, let's call this dialogue unit: "*Root*". The attribute values or items are considered as local buttons or options and identified by the attribute values themselves. So, part of the contents of this dialogue unit in the present example would be:

```
{B_French}        X0 Y6      B_French          assign(reg97,French);
{B_Italian}       X0 Y7      B_Italian         assign(reg97,Italian);
{B_Carlton}       X20 Y6     B_Carlton         assign(reg98,Carlton);
{B_Abbeywell}     X20 Y7     B_Abbeywell       assign(reg98,Abbeywell);
{B_3-10}          X40 Y6     B_3-10            assign(reg99,3-10);
{B_10-15}         X40 Y7     B_10-15           assign(reg99,10-15);
{B_quit}          X0  Y0     B_quit%           quit();
{B_Show-List}     X10 Y10    B_Show-List%      assign(reg91,View);

{(reg97=French) AND (reg98=Carlton) AND (reg99=3-10) AND (reg91=View)}
View    reset(reg91)      D1[];

{(reg97=Italian) AND  (reg98=Abbeywell) AND (reg99=10-15) AND (reg91=View)}
View    reset(reg91)      D2[];
```

The execution of the Root dialogue (above) by the Chisl intrepreter *Chip* issuing the following command: "*Framex Root* " will generate the display of figure 1.1.

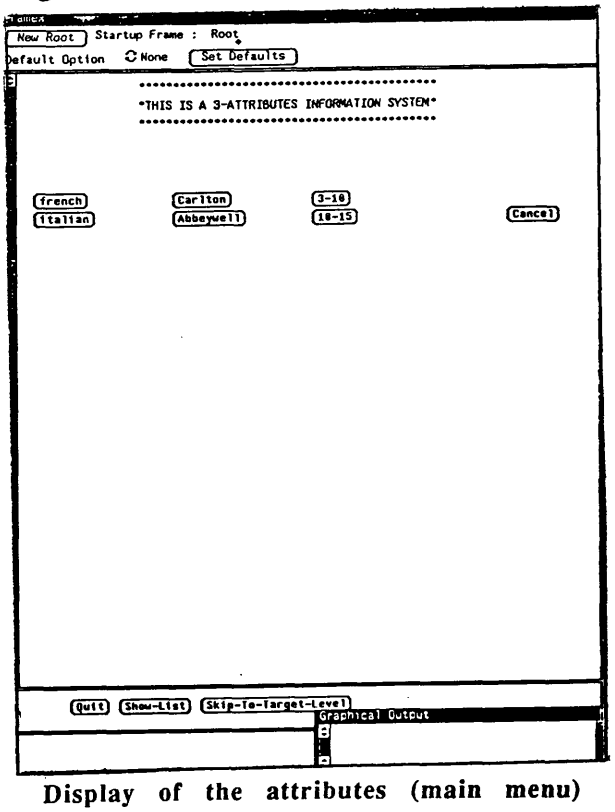


figure 1.1. Display of the attributes (main menu)

For simplicity, we suppose that only two selections (a combination of 3 options) have associated menus of available restaurants. So, two

dialogue units could be called or activated whenever the corresponding selection condition becomes true, which means:

(1) if (reg97=French) AND (reg98=Carlton) AND (reg99=3-10) AND (reg91=View) then the dialogue unit D1 is activated, or

(2) if (reg97=Italian) AND (reg98=Abbeywell) AND (reg99=10-15) AND (reg91=View) then the dialogue unit D2 is activated.

These two selection conditions illutrate perfectly the multiple menu selection property of the *Dining Out In Carlton* example.

Each dialogue unit is specified in the same way as the Root dialogue unit. Only the dialogue unit D1 is considered here for illustration. So, D1 may look like:

```
{B_1.French1-Carlton-3-10}          X0  Y9     B_1.French-Carlton-3-10
                                               assign(reg1,item1);
{B_2.French2-Carlton-3-10}          X0  Y10    B_2.French2-Carlton-3-10
                                               assign(reg2,item2);
{B_Show-Info-Page}                  X20 Y0     B_Show-Info-Page%
                                                     assign(reg90,OK);
{(reg1=item1) AND (reg90=OK)}        OK        reset(reg90)      D11[ ];
{(reg2=item2) AND (reg90=OK)}        OK        reset(reg90)      D12[ ];
```

This means, that the dialogue unit D1 displays a menu of two local options (items) and one global option. The execution of this dialogue unit together with the root dialogue unit by the Chisl interpreter will generate the display of figure 1.2. This execution can be achieved without necessarily issuing explicitly the execution command. This can be done simply by selecting the "*New Root* " option (figure 1.1) which invokes the Chisl interpreter to execute the updated dialogue in consequence.

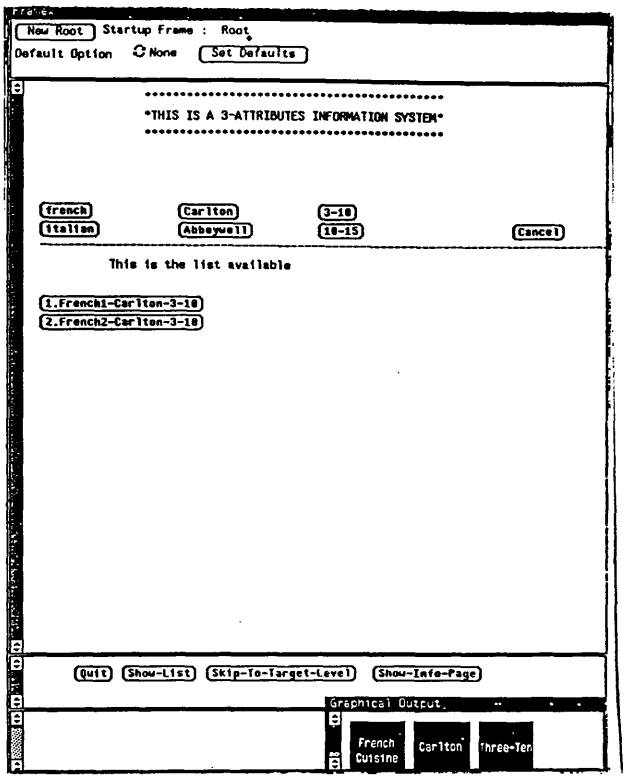


figure 1.2. Display of the available list

So, if item1 and the global option are selected (identified by registers "reg1" and "reg90") then the dialogue unit *D11* is activated, the information page corresponding to the item chosen (1-French-Cardlton-3-10) is displayed, or if the item2 is selected then the dialogue unit *D12* is activated instead.

Let's consider the dialogue unit *D11.D11* will display the information page where the target information is always retrieved. Usually, the information page which is the main concern of the information system provider contains a large amount of information. There are many ways of presenting or displaying this page on the screen providing better layout and greater clarity. So, doing this using the Chisl specification language will lead to larger dialogue units and require great attention to writing a more accurate dialogue specification.

The dialogue unit *D11* could be specified as follows:

```
ENTRY   message(0,15,13,"THIS IS THE INFORMATION PAGE")
        message(1,15,14,"_____")
        message(2,10,15," CHEZ MAXIM(****)")
        message(3,10,16,"_____ ")
        message(4,5,17,"SOUP ")
        message(5,5,18,"~.~.~")
        ..
        ...
        ....
{B_Dummy}    X0 Y13  B_Dummy  assign(reg92,Dummy);
```

The execution of this dialogue unit together with the two dialogue units already specified above by *Chip* will generate the display of figure 1.3.

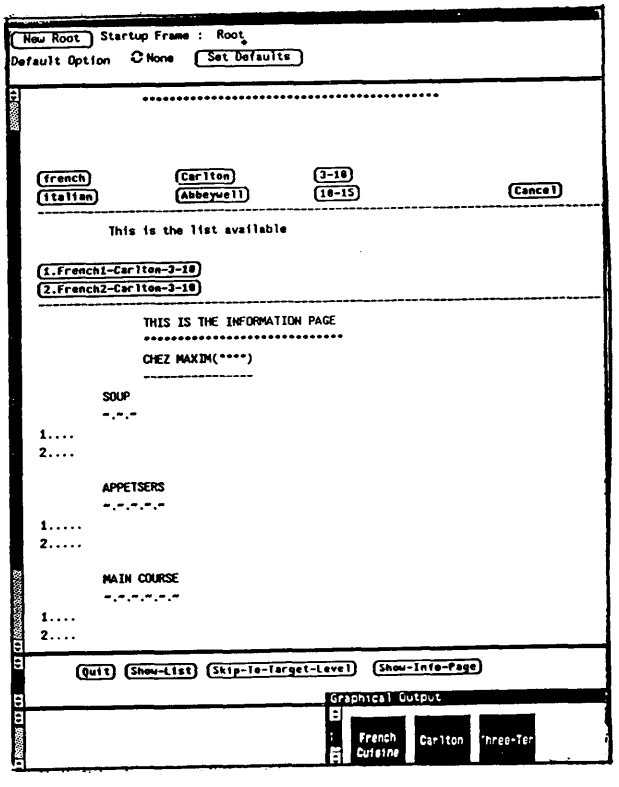


figure 1.3. Display of the information page

As we can see from this dialogue unit (*D11*), all the layout of the information page should be done explicitly by the means of a pre-defined routine *message* which takes as arguments: the message identifier, the (X,Y) coordinates of the first character of the text and the text to be displayed. Finally, the dummy option is added so that the dialogue unit can be exited when selecting an option from the upper level in the hierarchy (a deficiency in the present version of Chisl).

As has been demonstrated in this exercise, the multiple menu selection property was possible using Chisl. Moreover, options from three different levels of the dialogue (figure 1.3) are made available to the user, which illustrates the instantiation or upcoming selection technique (chapter 1). Finally, since the first level options are always available, therefore, the parameter node concept is also possible.

After having specified all the dialogue units, the dialogue is hierarchically organised (figure 1.4). The hierarchical nature of this dialogue structure arises from the fact that the dialogue units are called from within an action sequence.

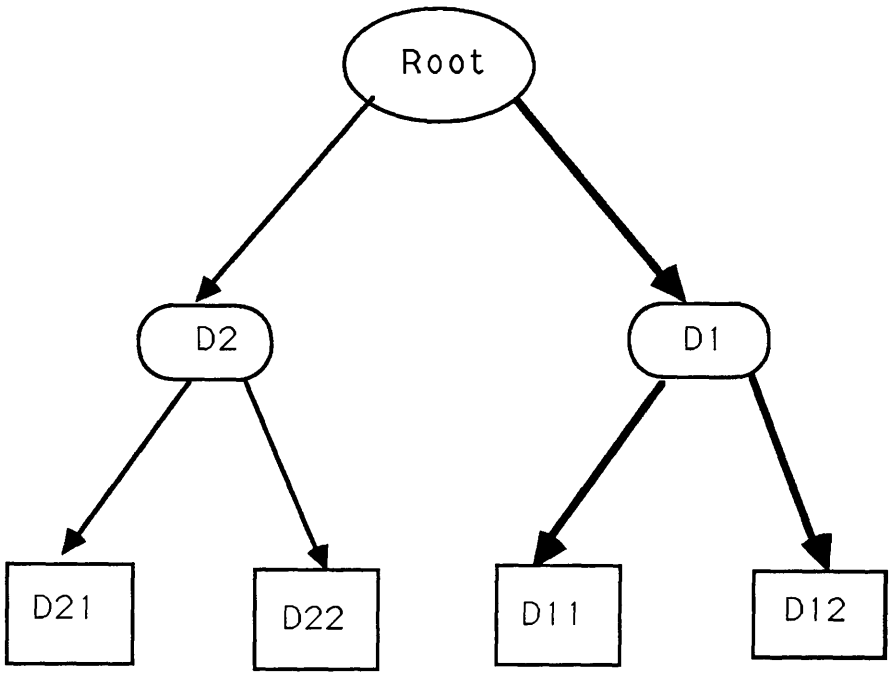


figure 1.4. **Hierarchical Dialogue structure.**

As consequence of experience in using Chisl, an auxiliary goal was formulated. The new objective was to provide a means of specifying a dialogue without necessitating the learning of a formal specification language, the aim being to avoid the misleading impression of the system's functionality given by the Chisl specification. This new goal led to the construction of a preprocessor: the *Chisl preprocessor*. The preprocessor specifications are given in appendix A. The design and implementation of the same dialogue or example using the preprocessor are discussed next.

2.1.2. Using the preprocessor

With this method, the user, instead of specifying the dialogue in terms of dialogue units and the Chisl specification language, has to specify the dialogue in terms of ordinary text files called the *PreChisl DU files* (See Appendix A). These files are translated into the

Chisl specification language for later interpretation and execution by the Chisl interpreter *Chip*. Three types of files have to be specified or created because there are three levels in the hierarchical structure of the example.

i. *The attributes file*

This file contains all the information related to the options to be displayed (attributes) at the first level. Let's call this file F. In the present example, its content would be:

French
Carlton
3-10
F1
D1
icon1
icon2
icon3
Italian
Abbeywell
10-15
F2
D2
icon4
icon5
icon6

- French, Carlton, 3-10, Italian, Abbeywell, 10-15 represent the options names (attributes).
- F1, F2 are PreChisl DU files, containing textual information about the list of restaurants which will be displayed at level 2.
- icon1, icon2, icon3, icon4, icon5, icon6 represent the names of files containing the icons to be displayed upon a selection of the corresponding option in order to indicate the selected state of the option since this facility is not available in the current version of

the Chisl system.

- D1, D2 are the Chisl DU files into which F1, F2 are translated respectively.

ii. *The PreChisl DU files type1*

F1, F2 are of this type. Only F1 is considered here. F1 is:

1.French1-Carlton-3-10
F11
D11
2.French2.Carlton-3-10
F12
D12

- F11, F12 are PreChisl DU files of type2, files containing detailed textual information about a specific item at level 2.
- "1.French1-Carlton-3-10", "2.French2.Carlton-3-10" are the two items displayed upon the selection of the three attributes (French, Carlton, 3-10), that is the local options at level 2.
- D11, D12 are the Chisl DU files into which F11, F12 are translated respectively.

iii. *The PreChisl DU files type2*

F11, F12 are files of type2. Only F11 is considered .

This file is an ordinary text file which can contain any text. No special format is required for this type of file, since the content of this file represents the information page. This file is exactly displayed as it is written. So, by this means, it is much easier to modify or add items of information. F11 may look like:

THIS IS THE INFORMATION PAGE

CHEZ MAXIM(****)

SOUP
~.~.~

1.....
2.....

APPETISERS
~.~.~.~.~.~.~

1.....
2.....

MAIN COURSE
~.~.~.~.~.~.~.~

1.....
2.....

DESSERTS
~.~.~.~.~.~

1.....

As can be seen from this example, the file structure (figure 1.5) is equivalent to the dialogue structure (figure 1.4) which is being built using the preprocessor.

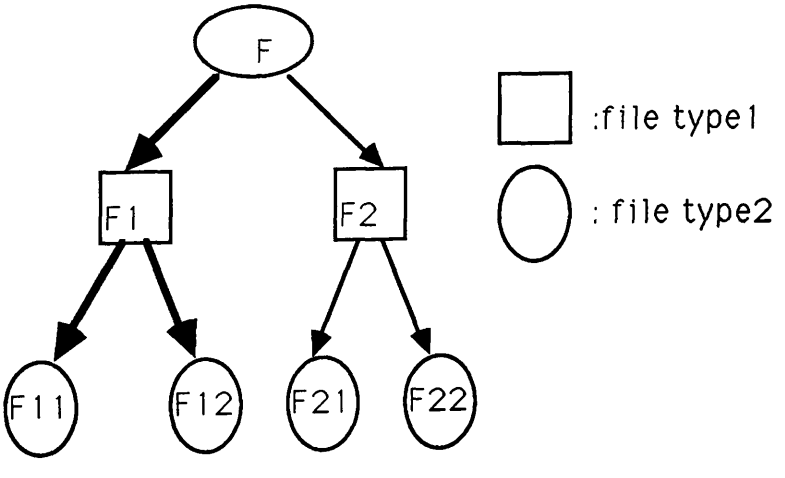


Figure 1.5. Hierarchical PreChisl DU file structure

2.1.3. Invocation of the preprocessor

When all the textual files are created, the dialogue is built and subsequent communication between the user and the system is via the user interface generated by the Chisl system, as illustrated by the figures above. The command: "*PreChisl F Root* " takes two file names (*F* and *Root*) as arguments. *F* is the name of a file containing textual information about the attributes (see Appendix A) to be displayed as the main menu options. *Root* is the name of a file which will become the Chisl root dialogue unit. The information contained within the file *F* will be translated into the Chisl specification language within the dialogue unit *Root*. Thereafter, issuing the command: "*Framex Root* " will invoke the Chisl interpreter to execute the prespecified and translated dialogue. This will result in the displays of the figures 1.1, 1.2 and 1.3.

The user makes his choice of parameters, after which a list of available restaurants is displayed. Thereafter the user can selectively retreat to change any of the three parameters. This results in an updated list based on the new value and the other (unchanged) attributes values. At the third level of the hierarchy, the user can also either return to the first or second level in the hierarchy by either selecting one option from the main menu options (displayed by the root dialogue unit) or an option displayed by one of the second level dialogue units.

The current implementation of the Chisl system does not provide an implicit way for displaying a history of the selected items or the current path. However, this is achieved in an explicit way by displaying an icon for each attribute upon its selection in a separate graphical window (figure 1.2).

3. Discussion

The two versions of the "Dining Out In Carlton" example (the version described in [Hepe et al., 85] and the one prototyped in Chisl) are discussed in terms of differences and improvement.

The major difference resides in the way in which the techniques discussed earlier are illustrated and exploited, and the user interface supported or generated for each version.

In the first version described in [Hepe et al., 85], most of the navigational aids are used apart from the sideways-viewing one which is quite difficult to achieve within the conventional display used. Moreover, the user interface is organised in such a way that:

- At level 1, the user is presented with the display of figure1.6 from which he selects a combination of three options (attributes) leading him to level 2.

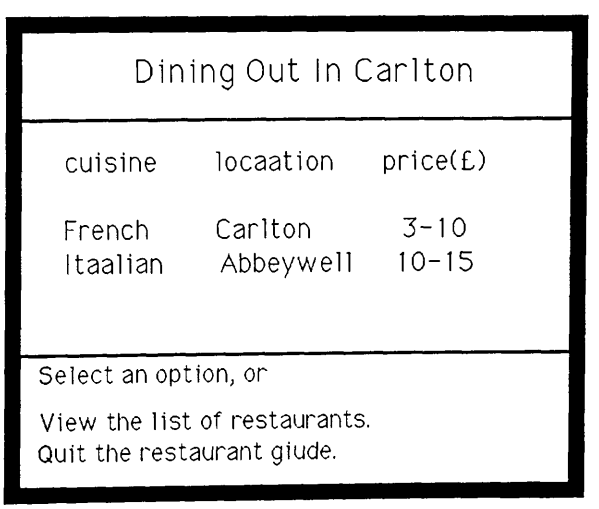


Figure 1.6. Display of option menu at level 1.

- At level 2, the user is presented with the display of figure1.7 which replaces the first display. At this level, the user can either select an option which will lead him to the third level or return back to the option menu.

Figure 1.7. Display of a list of restaurants at level 2.

•At level 3, the user is presented with the information page of a specific restaurant chosen at level 2 together with the options allowing him to return to either of the previous levels.

It is clear that the user is presented with only one display at a time where the backtracking option is necessary for navigating or moving through the system hierarchy. In a hierarchically organised system where the backtracking option is the only means for navigation, it is hard for the user to see and understand the efficiency and the powerful navigational aids provided by those techniques. However, in the version prototyped using Chisl, some of these techniques, such as *parameter nodes* and *selective retreat*, are automatically supported or provided by the Chisl system. This is due to the more flexible way in which local and global options are handled as already

demonstrated. Moreover, the *instantiation* technique is better illustrated when using the Chisl system since the user can see instances from all the three levels simultaneously (see figure 1.3). By this means, the user can navigate more accurately and rapidly through the system hierarchy. More important is the fact that the user is given the opportunity to cancel any doubtful choice and change his choice, since the option menu is always available to him (see figures).

Finally, the *sideways viewing* technique which has been omitted in the original or first version, could be easily included in the second version. This could be achieved for example by displaying all the nearby menus (level 2) in a second interaction window. So, if an option is selected from that window, the menu to which this option belongs could be displayed in the principal interaction window. But, and unfortunately, the present version of the Chisl system (still in the process of development) does not handle or support the case of displaying and selecting from another window apart from the control panel window. This has prevented the realisation of the idea above.

4. Difficulties and Deficiencies in Chisl.

This section outlines some difficulties and problems encountered during the above experience in using Chisl.

1. Only one string of characters is allowed to represent an option.
2. An option is identified only by the string, so no identical strings are used.
3. A dialogue unit must have at least one local option in order to be exited.

4. Nonexistence of a primitive or a function which allows the removal of a button as for example for a message.

5. The option or button selected should remain highlighted as long as it is activated.

6. Sometimes, an infinite loop situation could happen, when for example in a parent dialogue unit one or more test conditions are found always to be true when calling another dialogue unit where no test condition is true. This may be due to the misuse of the registers.

7. Sometimes, the error messages displayed by the Chisl interpreter do not seem to be very explicit.

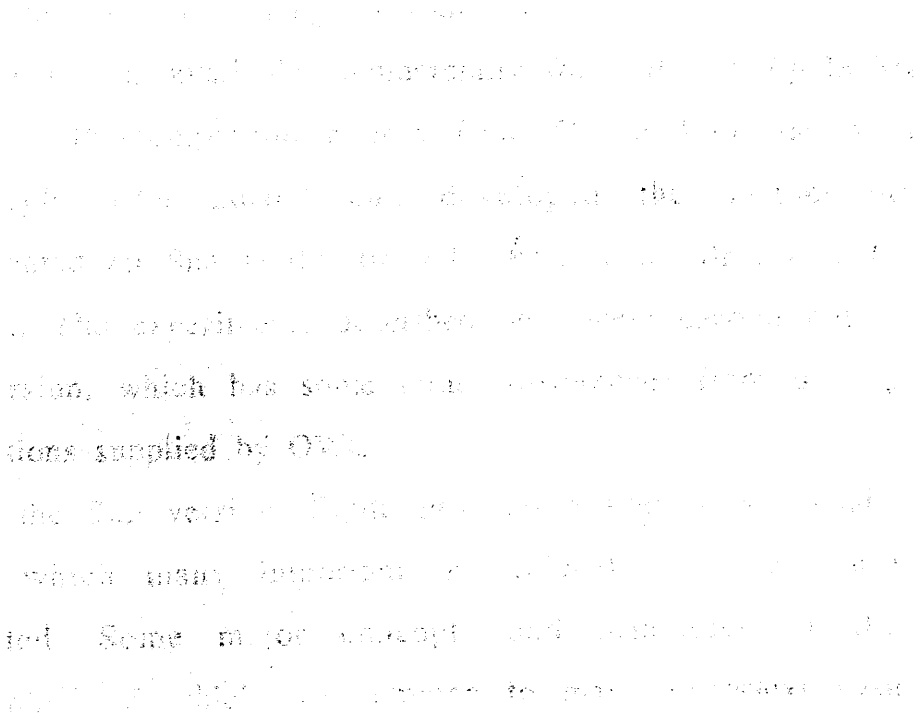
8. The number of the registers manipulated is limited. So in a very large dialogue the situation of lack of resources could happen where for example more registers are required.

However, some of these problems have been solved in later versions of the Chisl system.

5. Summary

One of the major difficulties in the Chisl system is the hierarchical structure of the dialogues and the specification language itself. In the beginning it was quite difficult to map the user interface design requirements onto the Chisl specification language, but, after a period of time using the system a better perception of Chisl was acquired. The use of the Chisl system would highlighted the privilege of one class of users (knowledgeable) from another (novice or casual). Nevertheless, many of the techniques mentioned in chapter One have been implemented in the chosen example and some important concepts are well handled by the Chisl specification language.

In the course of carrying out this exercise, the need became clear for an additional tool to simplify the creation and editing of menu structures of the type required by the exercise, and a preprocessor for this purpose was constructed. This had the added advantage of relieving the dialogue designer from the need to have a detailed understanding of Chisl syntax (The long-term aim of the Druid project, whose work produced the Chisl language, is to provide high-level graphical tools for editing all aspects of a dialogue specification).



3.2. Guide

1. Description of Guide

Originally, Guide was designed typically for electronic or interactive documentation purposes and first applied to the Unix documentation [Brown, 86]. It is an interactive computer-based document system, whose user interface exploits hypertext concepts, eg. links (Chapter One). Guide may thus be considered to be a "hypertext computer-based document" system. It allows users to build their own documents interactively by providing a simple way for selective display of information and for creating material that can be so displayed. Guide, as a tool and as a hypertext system, can be used for: storing, cataloguing, cross-referencing, structuring, prototyping and retrieving information

Guide is available commercially for both the Apple Macintosh and IBM PC-compatible micros from Office Workstations Ltd. of Edinburgh, who ported and developed the system originally implemented on Sun workstations by Prof. Peter Brown of University of Kent. The experiments described here were carried out using the Sun version, which has some minor differences from the Apple and PC versions supplied by OWL.

In the Sun version, Guide provides a special command dialogue within which many important hierarchical structure concepts are embedded. Some major concepts and principles of the Guide philosophy and which are common to many hypertext systems are discussed.

2. Concepts and Principles

2.1. Buttons

One of the most important features of Guide is the notion of a button (which in the hypertext terminology introduced in chapter 1 is simply called a link). Guide offers two major types of button or link:

- *replace-button* : causes the button to be completely replaced by the text and/or picture pointed to by the button when it is selected. There are three kinds of replace-buttons:

  definition-button: the replacement associated with the replace-button applies not only to the button itself but can also be employed by other usage-buttons and/or glossary-buttons (See below) that match the same name.

  local-button: the replacement applies only to the button itself.

  usage-button: the replacement is created dynamically (eg, using the definition or the result of running a shell-script).

A group of replace-buttons may be organised such that all the buttons are replaced by one button's replacement. These buttons form an *enquiry*. The replacements of this kind of buttons are displayed within the principal frame-of-view (below). These buttons are made emboldened when created.

- *glossary-button*, the replacement of a glossary-button is called a definition. Whenever a glossary-button is selected, its associated definition is displayed in a separate area called a glossary-view (See below) and the original button still remains. This is the difference between these two types of button. A Guide document may contain several occurrences of the same glossary-button. Moreover, several different glossary-buttons may share the same name, that is they have different definitions. A glossary-button is underlined when created.

2.2. Views

Unlike many other hypertext systems, Guide does not support heavy use of windows that have one-to-one correspondence with nodes in the database (chapter 1). Instead Guide has adopted the 'split screen' display concept and generates different, independent areas called views or frame-of-views.

In the Sun implementation, a Guide screen consists mainly of one Sun View window which may be divided into different views (see figures). It is screen-based, that is it provides a convenient user interface by displaying a whole screenful of information, menus, etc...together with a scrolling mechanism for each view.

2.3. Editing

Guide allows the capability of editing by providing the user with some facilities in order to manipulate the material to be edited. Guide provides two types of editing :

- *Textual editing*, the usual way of editing.
- *Structural editing*, only possible in author or design mode where the underlying structure is made visible. This allows the author to identify the types of buttons where each structure (button and its replacement) is delimited by special characters. Buttons (structures) can only be created in author mode using an additional menu which consists of a set of commands (see Appendix B). The way the buttons and replacements are constructed is made invisible to the reader (user). Guide distinguishes an ordinary text file from a source file in such a way the former does not contain any structuring (no underlying structure visible to the reader).

2.4. Replacements

Unlike many other interactive systems, Guide provides three useful mechanisms whereby buttons are replaced automatically on loading. Each mechanism meets a different user need, but only two of these are worth considering in the present discussion. These concern the automatic selection of the buttons with specific properties or unasked replacements. They are:

•*Asking-level and User-level*

Each replace-button has an asking-level (a digit between 0 and 3). It is set to 1 by default at creation. The asking-level can be changed by the end-user. Associated with each user is a user-level which is set to 1 by default. The user can change his user-level by specifying it in the command which is used to load the source file. This mechanism implies that all the replace-buttons for which the asking-level is less than the current user-level are automatically replaced (without asking the reader). It is mainly used to control some buttons such that the end-user or reader may not be aware of. This mechanism can be regarded from the designer's point of view as one of many important techniques for accommodating different communities of users with different needs and requirements.

•*Preset replacement*

Unlike the first mechanism, this one is principally useful from the user's perspective since it gives the opportunity to have some sort of control over individual local and definition replace-buttons. Presetting a button means not only is the button itself replaced but all the replace-buttons of the same name are automatically replaced too. This can also be preplanned by the author. More details about Guide and its concepts can be found in [Brown, 87].

2.5. The command dialogue

All the structural editing and authoring are achieved via a special set of menu commands which represent the menu specification language of Guide. The description of these commands is given in appendix B. The underlying principles of the menu specification language language are explored by considering the implementation of an example.

3. Experience of using the Guide system

A different example was chosen for examination instead of the *Dining Out In Carlton* example previously discussed in the Chisl section, because of the unsuitability of Guide for that application. This inappropriateness arises mainly because the notion of buttons and their replacements does not fit well with the requirements of the main menu in the example, which consists of a set of attributes that can be selected in any order and in any number (1, 2 or 3). This means that neither the different types of buttons nor their combination can be used to achieve the multiple attribute selection property of the Dining Out In Carlton example. However, if the main menu is considered as a multi-level menu attribute, the notion of buttons may apply but still in a rather inappropriate manner. In principle the example could be implemented in a purely hierarchical fashion, although this would impose an unnatural constraint on the attribute selection scheme, and would lead to a combinatorial explosion in the overall structure. This is the main reason why another example had to be considered instead. The example used was the On-Line Library already described in chapter 1.

The following section discusses the major points involved in the design and implementation of a Guide interface to the *On-Line Library* example, and outlines the important steps in the

implementation.

3.1. Design and Implementation

This section illustrates Guide from the designer's perspective, in particular how the information handled within the On-Line library example is structured and presented to the end user, and how the menu specification language provided is exploited for such purpose.

3.1.1. Entering the design (author) mode

Guide uses the end user (reader) mode as its default mode, where only a set of menu commands (see figure 1b in Appendix B) are available. Therefore selecting the author option from this menu switches to the author mode making available an extra set of options (see figure 2b in Appendix B).

3.1.2. Authoring and Design

This has much to do with structuring and representing the material to be displayed and accessing the information to be retrieved. The hierarchical organisation of the menu items implied by the CR classification scheme should be displayed in the principal view accordingly, that is the four menu levels of the On-Line Library example should be displayed such that whenever a menu item is selected, its corresponding lower level options are displayed within the principal view. To meet this requirement, This menu item should be created as a local replace-button, and its lower level options as its replacements. All the four level menus are created in the same manner. In order to make the display clear, the menu items are displayed such that the hierarchical structure of the menus is well reflected (see figures) on one hand. On the other hand, menu items not already selected (emboldened) are distinguished from the menu

items already selected (plain text) which themselves appear within the displayed replacement (See [Brown, 87] for more details on the creation of local replace-buttons). Figure 2.1 illustrates the first level menu (main menu) of the example in author mode.

All the options of the fourth level (subject descriptors) should lead to the display of their respective target information when selected. Instead of displaying the target information within the principal view which may render it clutter and inadequate for visual scanning and reading, it is displayed in a different view (glossary-view). To this end, all the fourth level options are created as glossary-buttons and their respective target information is created as their definitions. These definitions are created in special definition-file called the *glossary.guide* file (See [Brown, 87] for the creation of glossary-buttons and their definitions). There are some other menu items which do not have any further associated options such as: "General" and "Miscellaneous". These options are also created as glossary-buttons (see figure 2.2). So far, only hierarchical organisation is illustrated. Since cross-references exist in the CR classification scheme and in order to distinguish them within the prototype, references are put between brackets (see figures). A cross-reference means jumping from one node to another node. In this example, a cross-reference is represented by a button, when selected, brings up a set of options of an already existing node within the tree structure. Therefore, to meet this requirement, cross-references are created as usage-buttons, because a usage-button uses a definition of an already existing button (See [Brown, 87] for the creation of usage-buttons).

This is how all the information is structured and presented to the user. In author display, the underlying structures are made visible to the designer helping therefore the authoring and the

design of the prototype. The figure 2.1 shows the main menu in author display. Each structure is delimited by two special characters, in this case B and its mirror image for a button.

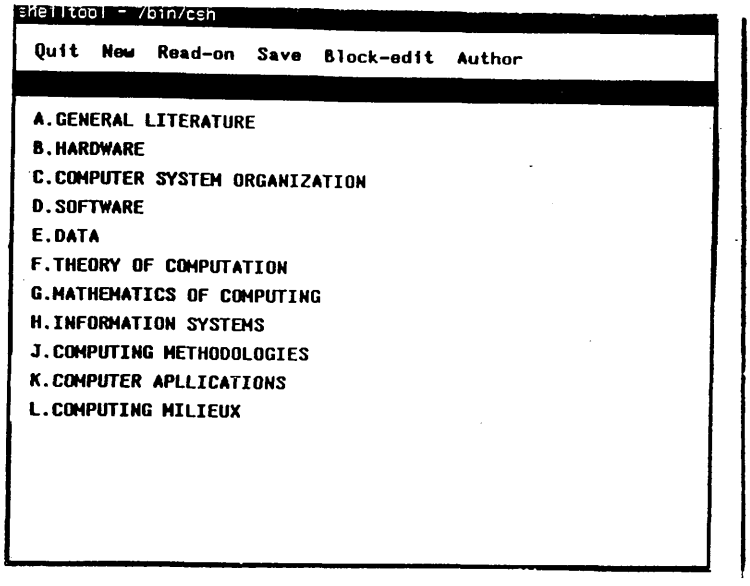


figure 2.1. main menu in reader display

The possibility of switching to the reader display while prototyping the user interface to the On-Line Library example is a very helpful and useful facility allowing the designer to see the prototype as the reader would see it.

3.1.3. Saving the prototype

After having built the prototype, this has to be saved. It can be saved either as a source file or as an ordinary text file. In this case it is saved as a source file with all its underlying structures. The name of the source file in which the four level menus are saved is *library.gu* , and the definitions are saved in the *glossary.guide* source file.

3.2. Using the prototype example

This focuses mostly on the reader's perspective, in particular how users move around the information space to reading and finding information.

3.2.1. Entering the user's (reader) mode

There are different ways to enter the reader mode. But only two are considered in the present discussion

i. *by starting a Guide session*

The user loads the source file by issuing the following command: *guide library.gu* . Therefore, the display of figure 2.1 appears on the screen. The default mode is reader mode as said before.

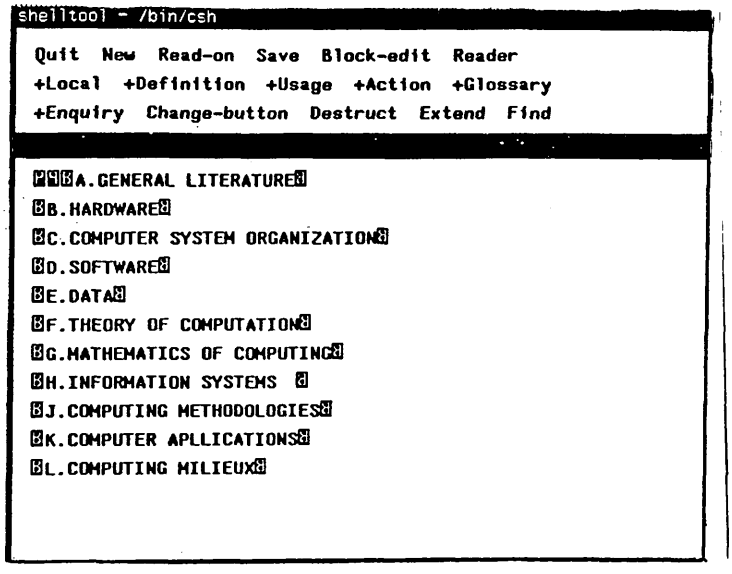


figure 2.2. First level of menus (main menu) in author display

ii. *within a Guide session*

During a Guide session, switching to reader mode (if not already in) is by selecting the reader command from the menu of figure 2b in Appendix B.

The first case is likely to be the normal and usual way of entering reader mode.

3.2.2. Reading and Retrieval

Retrieving information is the main purpose in using the prototype. Retrieving all the books covering a specific topic in the computing field or finding all the books written by a given author both are examples of information retrieval task that a user is likely to be carrying out. Guide provides two strategies or ways for information retrieval task, these are:

i. *link following or item selection*

The information seeking process can start from the main menu (see figure 2.1) by selecting the appropriate menu items till the target information is found.

Let's consider the following example, selecting the menu item labelled "H.INFORMATION SYSTEMS" from the main menu (figure 2.1) will cause an extra menu items to be displayed as in figure 2.3.

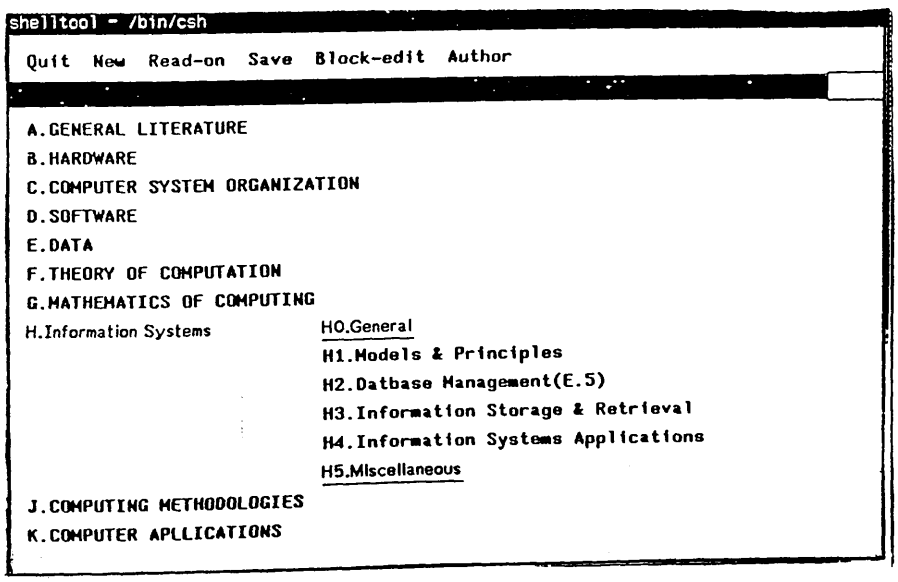


figure 2.3. Second level of menus

Therefore selecting for example "**H1.Models and Principles**" leads to the display of figure 2.4.

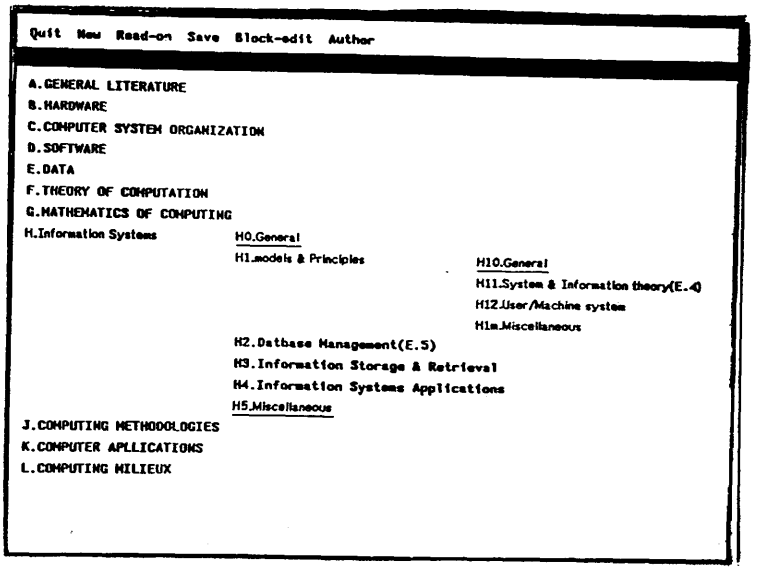


figure 2.4. third level of menus

Finally selecting for example "**H10.General**" will lead to the display of figure 2.5.

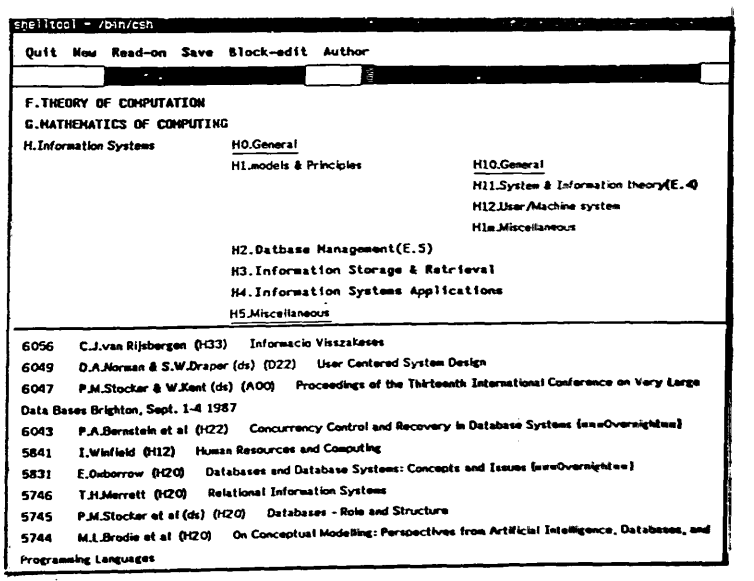


figure 2.5. Display of the target information

As it is illustrated by the figures displayed above, some of the navigation techniques mentioned in chapter One are well handled. The figures highlight the availability of more than one level of menu items at a time, illustrating therefore, the instantiation or upcoming selections technique on one hand. On the other hand, nearby menu items are also made available and selectable, thus illustrating the sideways viewing technique. Moreover, these two techniques could fully and completely illustrated if the user sets his user-level to the highest level, and all the menu items are automatically replaced, thus the whole structure is made available and visible. Finally, the parameter node concept would have no sense in this example.

ii. *string searching*

In this case the information seeking process can be restricted to a string search. This implies that the string to be searched or found within the information space has to be specified. However, there are different ways of doing so in Guide. Typically this is expressed by the fact that the find command can be invoked differently. This method is more appropriate for searching for general terms which are the keywords within the information space of the example. This is achieved by selecting the *find* command from the command dialogue (See figure 2b in AppendixB), and by typing in the string to be searched for in a prompt frame-of-view provided for this purpose.

4. Discussion

The previous sections have been mostly on the Guide tool concepts and principles and the design and implementation of a particular application encompassing these principles and highlighting the underlying specifications of the prototype built. The discussion in

this section will principally focus on the major design issues raised during this particular experience. These issues concern typically the following points.

4.1. The command dialogue

The menu commands provided have been used to construct the menu-based prototype as shown in the previous sections. Therefore, the concepts embedded within this command language seem to be attractive for hierarchically structured systems, but non-hierarchical structures are also supported. Moreover, it enables a number of features that overcome many of the objections to hierarchically organised systems such as instantiation and upcoming selections (Chapter 1) to be realised or achieved. It can be considered as a menu specification language embedded within the run-time environment. This helps increase the flexibility and efficiency of the prototype creation and use. There is no particular specification syntax to learn. However, some negative effects due to the misuse and mishandling of those concepts may occur. Some of the merits of command menus are also discussed in (chapter 1).

4.2. Structure and navigation concepts

Unlike many other systems, Guide does not include the concept of a browser which is usually used to give a global view of the structure or a part of it as a means for traversing the structure and especially when it grows more complex, but instead it uses the scrolling mechanism.

Instances are made selectable at any time. Moving up and down the menu structure are straightforward. Although Guide does not provide explicit *Goback* , *Goto* and *Cancel* as in other systems, upper level menu items are available, and upon selection, the user moves

up the hierarchy thus performing the Goback action as in Chisl. Also, the Goto action is catered by the fact that usage-buttons can just do that (cross-references). Moreover, Cancelling a menu item is simply done by undoing its replacement, therefore returning to a state prior to its use.

However, some negative effects may become important issues when considering the information space as a whole and the movement around it. In effect, if the information to be displayed to the user in not well laid out even for simpler hierarchical structures which are the most natural way of organising the information, it will be difficult to grasp and understand the overall structure, let alone the navigation aids and concepts embedded within that structure. Sometimes, organisational links may point to pieces of information which when combined together form a hierarchical structure which is not visible at all to the user when displayed because of the linear display of the information. It is only the display which is linear but not the underlying structure. This does not help the user develop a suitable mental model of the underlying structure. Therefore, the getting lost problem known with many other systems becomes an issue. Furthermore, by traversing down through the levels of menus and moving around the information space the user may forget the original context in which the material was retrieved, because there is no way for providing cues or displaying selected records (history selection). The approach used in KMS for such a purpose is to assign an asterisk (*) for a previously selected item such that when you go back up a level, you easily recognise the item previously selected, therefore avoiding to selecting it if another search path is required.
In Guide you have to rely on your memory in order not to follow the same path again. This makes the memory overload problem another issue, but it is not as severe as is found in other systems supporting a

heavy use of windows (or frames in KMS and Cards in HyperCard).

4.3. The User Interface

From the user's point view, any user interface created with the Guide tool is characterised primarily by its simplicity and ease-of-use. This is due mainly to the "frame-of-view" concept and the selection mechanism used (click on a mouse button).

From the designer's point of view, however, Guide does not provide enough facilities to help the menu designer to conduct and design a well and efficient menu-based user interface. Guide is lacking techniques which might help increase the visual scope of the user and which addresses the problem of cognitive layout of user interfaces [Norman et al., 86]. This may result from the limited text editor (highlighting facilities not available) used and also from the concept of replace-buttons and their replacements which do not allow much freedom in the way the information (surface layout) is presented. This issue concerns typically the way in which the user views and cognitively processes information presented in the different views which may compose the user interface. Therefore the designer has to consider very carefully the surface layout from which the user's mental model (cognitive layout) is derived. A broken visual scope of a Guide display may cause confusion, disorientation and difficulty in locating needed information on the display.

Unlike many other menu systems, the number of menu items which can be generated becomes a less important issue because of the scrolling mechanism used. Extended menus [Shneiderman, 86] may also benefit from this scrolling capability, thus speeding usage. Finally, I believe that more functionality and appropriate techniques

are needed to be added to those already supported in order to generate more flexible, consistent and efficient menu-based user interfaces despite their simplicity and ease-of-use.

4.4. Reconfigurability

Most of the prototyping is carried out during the design process. The behaviour of the Guide interface and more exactly the way the information is made accessible and displayed to the user may be more or less modified dynamically and tailored to meet the different needs of different users exploiting the Ask-level and User-level concepts discussed above. Since structural and textual editing are the only operations that are involved in the prototyping process, then the behaviour and the interface and the changes made to it are rather restricted and limited. End users as well as designers may be involved in the modification and reconfiguration. However, the interface designer has the possibility to protect the interface from being modified and changed. Moreover, there is no way of changing the internal specification of the interface nor can the command language used to build it be extended or respecified. This point is common to many systems eg. KMS. It is obvious then Guide can be regarded as a user interface style dependent creator tool. It enforces a particular interface style, like many other systems eg. Chisl and KMS.

5. Summary

Another system which belongs to the family of systems that may be regarded as user interface management systems has been studied and investigated. This analysis has shown that creating menu systems using Guide is possible but not to the extent of supporting the full range of conceptual operations that the user requires for a

given range of tasks. This is mainly due to the lack of functionality of the design tool and inappropriate exploitation of the various concepts embedded within the provided design environment. It may also result from the fact that creating menu-based user interfaces is not what Guide was intended for. In spite of this, some interesting design issues with their respective consequences have been raised. Some are common to many design tools and some others are purely typical to Guide such as: no explicit navigation commands, support for navigation aids aimed at overcoming the drawbacks of hierarchically organised structures, dynamic prototyping, and equal opportunities to designers as well as end users. Finally, I believe that more power and control over the Guide design environment is the key to a better achievement of its stated intentions.

3.3. KMS

1. Description of the KMS system

There is no unique way of categorising KMS, since it combines features from many types of software such as word processors, database systems, document management and information management systems. It can be described as:

- a spatial database system for managing (representing, accessing and using) all kind of knowledge which might be called: free-format

information (information which does not fit predefined patterns),

- a computer-based document storage and training systems,
- an electronic communication system via messages and discussion frames. In other words, it can be described as distributed hypertext system for managing knowledge in organisations.

KMS is claimed by its suppliers to be a general purpose human computer interface system. It is based on the Zog approach to human-computer interaction developed at Carnegie Mellon University and used on the aircraft carrier USS CARL VINSON [Robertson et al., 81]. It uses primarily on the concept of menu selection, with a large database of menus and rapid response to selections. This makes the KMS Interface a particular style or type of interface. But, when considering the retrieval and the structure sides of the interface, KMS is best described as an information retrieval system.

2. Concepts and Principles

In this section, the major components which define or characterise the particularity of the KMS system and all the systems similar to KMS (based on common principles) are identified.

2.1. The Database

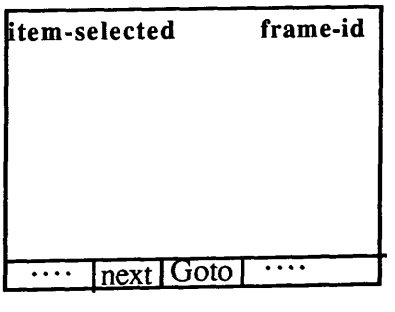
On the storage and the knowledge management sides, KMS is mostly characterised by its database whose design is based on some uncommon notions (different from the traditional ones). The following are worth mentioning:

Large size : The KMS database may be large enough in order to accommodate many thousands of frames without affecting the responsiveness of the system.

Shared by multiple users : The KMS database accommodates simultaneous use by many different users so that it can provide a simple but rich means of communication among the users.

Menus : A KMS database consists of a set of menus, whereas in a more conventional database, this can a set of records. In KMS terminology, a menu is called a frame. A frame is displayed in a KMS window which can have only two sizes: half or the whole screen.It has :

- a unique name displayed in the upper right corner,
- a set of options,
- a menu of global commands at the bottom.



A frame format

It contains *objects* which are of three types:

- *items* : text items or points,
- *connected objects* : items that are connected by lines and may be

simple or complex,

- *sets* : items and connected objects that are enclosed in a rectangle.

Each frame belongs to a *frameset* (set of frames). All the frames in a frameset share a name prefix, that is the frames have the names as *frameset-name i*, where "i" is the creation order of the frame.

Generality of representation : The KMS database is designed to handle all kind of knowledge. It integrates text, graphics and images in frames which are WYSIWYG screen-sized chunks of information. So, frames can be created , edited, modified and saved. There is no separate editor. In effect, KMS is good at handling free-format information.

Network / Tree structures : A KMS database can have a network structure in which data items can be linked to others data items in the database. Links are the interconnections between frames that are the essence of KMS.

Any item can be linked to another frame. This operation involves changing the item's link property. The links between frames are very important because they allow :

- frames to be arranged into hierarchies or network structures.
- creation of cross-references between related frames.

Frames can be linked together to form a Hypertext-like database (Chapter 1). Links can also have attached procedures to be executed when selected.

2.2. User Interaction

This section outlines some important concepts which govern the KMS User Interface and the User Interaction. These are:

Menu selection : Almost all interaction with the KMS user interface is done by making selections from the currently displayed menus.

Except when using the editor and answering for system prompt.

Fast response and Browsing : Upon an item selection, a new menu (frame) appears instantly (about 1s on average). Rapid navigating makes it easy to browse through large portions of the database and quickly move around within a smaller groups of menus. There are three ways for navigating a KMS database:

•Clicking on an item that's linked to a frame using the left button of the mouse which is labelled *Goto* whenever the cursor moves close enough the item.

•Going back, to a frame displayed earlier by clicking the left button of the mouse which is labelled *Back* when the cursor is in empty space.

•Clicking on one of the navigation command items at the bottom of the frame using any button of the mouse. Some of these are: *Goto* , *Next*, *Previous* .

Direct manipulation : The KMS system uses the direct manipulation approach to handle most editing operations which are performed directly on objects using the mouse buttons together with WYSIWYG features.

2.3. Functional extension

KMS provides some mechanisms for extending the system to allow new functions to be added. This is governed by the following principles :

Mapping data structures : The data structure of a new application should be mapped into frame formats and interconnection structures within the database.

Embedded programs : Programs that are needed to implement new functions are written in a special way that allows them to be embedded within the system, so that they can be used without

having to leave the system. These programs can be invoked via active menu selections (items with associated actions).

Environment frames : These are special frames from which the programs are invoked and controlled.

3. Experience in using the KMS system

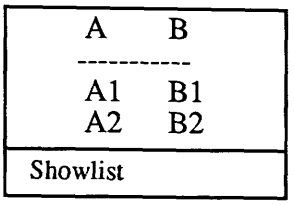
Since, KMS supports only a single selection mechanism, it is apparently clear that the multiple attribute selection property of the Dining Out In Carlton example would not be achievable. Moreover, the achievement of some of the important navigation concepts (chapter 2) would be very difficult because of the unavailability of the required underlying language constructs. However, the concept of rapid response to item selection as well as large frame display in KMS might be helpful and appeared to provide a reasonable solution to achieve the stated goals. Therefore, the strategies devised to exploit the concepts for a design and implementation of a user interface to *the Dining Out In Carlton* example as well as the application of the key design issues described in chapter One are discussed next.

3.1. Design and implementation

Instead of considering the full complexity of the Dining Out In Carlton example, a simpler exercise with the same multi-attribute selection property is discussed. Let's consider two attributes namely *A* and *B* and their respective values are *A1* , *A2* , *B1* and *B2* .

But, before going through this exercise in detail, rementioning some of the important steps of a similar exercise in Chisl at this point will serve as a reminder.

In effect, this is what the main menu would look like if implemented in Chisl:



- A selection of a combination of (A,B) in any order, or one of (A,B), or none of (A,B) would lead to the display of the corresponding menu (level 2).
- The user's choice is taken into account if and only if he issues the *showlist* option (the user is responsible for his choice).
- The displayed menu has a limited number of options where only one selection is made.
- Every time an item is selected, it is highlighted as feedback.
- It is possible to have more than one combination of (A,B) that do not have corresponding menus, for which a warning message is displayed.
- The structure of the example is hierarchically organised.

For this particular example, n=2 (number of attributes).

The maximum number of frames that can be generated is then: 1+ 4 + n1 + n2 + n3 + n4

This means, from the main menu (level 1), four other frames are possible (level 2), "*ni* " is the number of frames (number of options) generated from frame "i" at level 2. These frames represent the level 3 of the hierarchy.

In general, the maximum number is :

$$1 + \prod_{j=1}^{n} |A_j| + \sum_{i=1}^{m} |F_i|$$

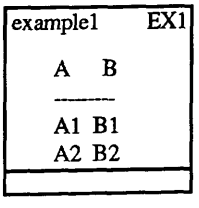
where : $|A_j|$ represents the number of values of attribute A_j
$|F_i|$ represents the number of options in frame F_i (level 2)
$m = \prod |A_j|, j=1,n$

Actually, thinking about implementing the example using KMS suggested two possible approaches which are discussed separately.

The first approach

This approach discusses attempts to follow the same methodology used in the Chisl implementation. An important feature of the Chisl application is the availability of the main menu at all times. Following this approach, the first thing to do is to add to the Home frame (as described in section 3.2) a new item called *Example1*, which will be an index entry to the example's database. The very first link to the database leads to the creation of a new frameset (if desired) which will have a unique name. Alternatively, the item may link to a new frame within an existing frameset. Assume a new frameset is being created with the name of *EX*. So far, a new KMS database is being created and accessed whenever the item *Example1* is selected. As stated from the previous sections, every time a frame is created within the frameset *EX* , that frame is identified by its unique name in the upper right corner *EXi* where "i" is the order in which the frame is created. For example *EX1* , *EX2* and so on.

So, the main menu (first frame) within the frameset *EX* is *EX1* . *EX1* may look like :



It is also stated that only one item is selected at a time and only one frame is displayed at a time .

The selection of one item at a time implies that at least 3 selections (2 for selecting the 2 attributes and 1 for selecting one option which is about to have detailed information) are needed to meet the retrieval task goal.

The single selection of either *A1* or *A2* or *B1* or *B2* means that 4 different frames, each of which is linked to one one of the 4 items

above have to be created. The first two levels of the structure are as follows:

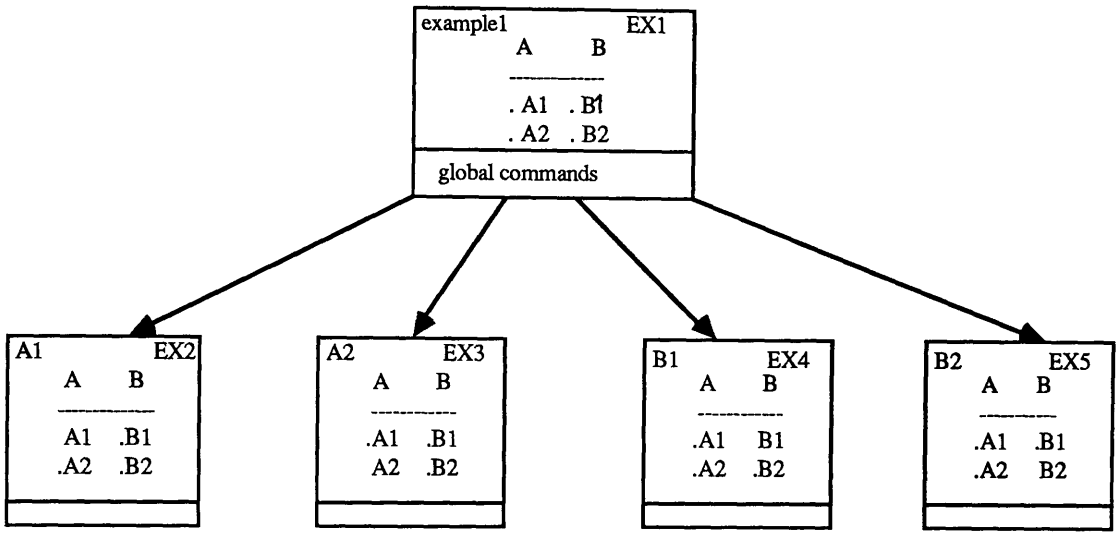


figure 3.1

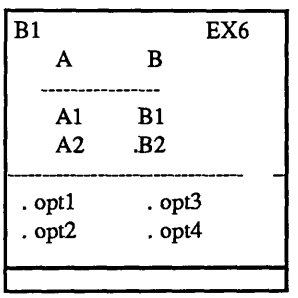
The dots (.) mean that the items have frames linked to them.

Figure 3.1. shows that one of the attributes values has already been selected. The selection of each of the four attribute values would lead to the display of a frame making available the other attributes of the main menu selectable. At this level, a second attribute has to be selected, this achieving the required 2 attribute selection before meeting the retrieval task goal.

At the second level, two different frames have to be created from each frame . This means for example, from the frame *EX2* , two frames linked respectively to *B1* and *B2* have to be created and identified by *EX6* and *EX7*. So selecting *A1* at *EX1* would leads to the display of *EX2* and selecting *B1* or *B2* would leads to the display *EX6* or *EX7* performing 2 attribute selection in consequence.

At frame *EX2* for example selecting *A2* would mean cancelling the previous item (*A1*) and this would lead to the frame *EX3* , thus, the link to the frame *EX3* from *A2* at EX2 has to be created or added. This shows that there is no way of cancelling a selection after it has been done before seeing the frame which is

linking to .i.e. the selective retreat (chapter 2) facility is not supported in KMS. This is because, the selection is directly taken into account and the display is immediately performed. This does not allow much time for decision making. So, the next level (third level) in the frameset (database) structure consists of 8 frames which are respectively (EX6, EX7), (EX8, EX9), (EX10, EX11), (EX12, EX13). At this level, let's consider only one frame for discussion eg. *EX6*. *EX6* would look like:



The contents of the frames of level 3 are different from those of the frames of upper levels, because at this level a limited number of options (requiring detailed information) is also added. This third level is very much like the level 2 in the Chisl implementation. Different numbers of options are available within each frame. So, all the frames linked to those options have to be created. These frames will represent the level 4 of the structure and which also represent the target frames.

As in level 2 (figure 3.1), selecting *B2* at *EX6* would mean cancelling *B1* , then the link to the frame *EX7* has to be created. Note that *A1* and *A2* are not selectable, but remain visible only for keeping the main menu visible at any time. Using this approach, the number of frames composing the first three levels is : 1 + 4 + 8 = 13 frames. The number of frames in the last level (level 4) depends on the number of options at level 3.

In this example we have 2 attributes (n=2), so the number of levels generated is 4.

In general, for n attributes, the number of levels which will be generated is n+2.

At level 2 we have $P = \sum_{i=1,n} |A_i|$ frames, where $|A_i|$ is the number of values of attribute A_i

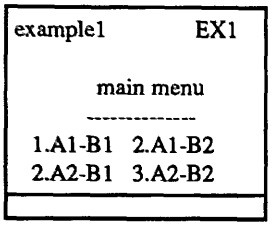
So, if P increases then the structure get broader, and if n increases then the structure gets deeper. This means, at least (n + 1) decision levels are required before retrieving the target. This approach has exploited the rapid response to item selection to simulate the multi-attribute selection property of the example and it is shown that this may lead to a huge and complex structure. From the frame builder's (system designer) point of view this situation may become frustrating, irritating and time consuming. The structure generated in this approach is a network structure.

What has been discussed so far is the way the frameset (database) structure is generated and what's the impact of the idea of attributes on the database structure which might be very huge and complex. Therefore, another attempt to reduce the complexity and the size of the structure is carried out and which is discussed in the second approach.

Second approach

In this approach, the structure of the database is reduced in complexity and size. This is due to the decision making process offered by the frame builder, which affects the way the main menu is presented. In effect, instead of having a decision point as a single attribute value, a decision point in this approach is taken to be a combination of different values of the attributes eg. (*A1*, *B1*), (*A1*, *B2*) and so on. This approach is another way of simulating the multiple attribute selection property. Thereafter, the main menu

may look like:



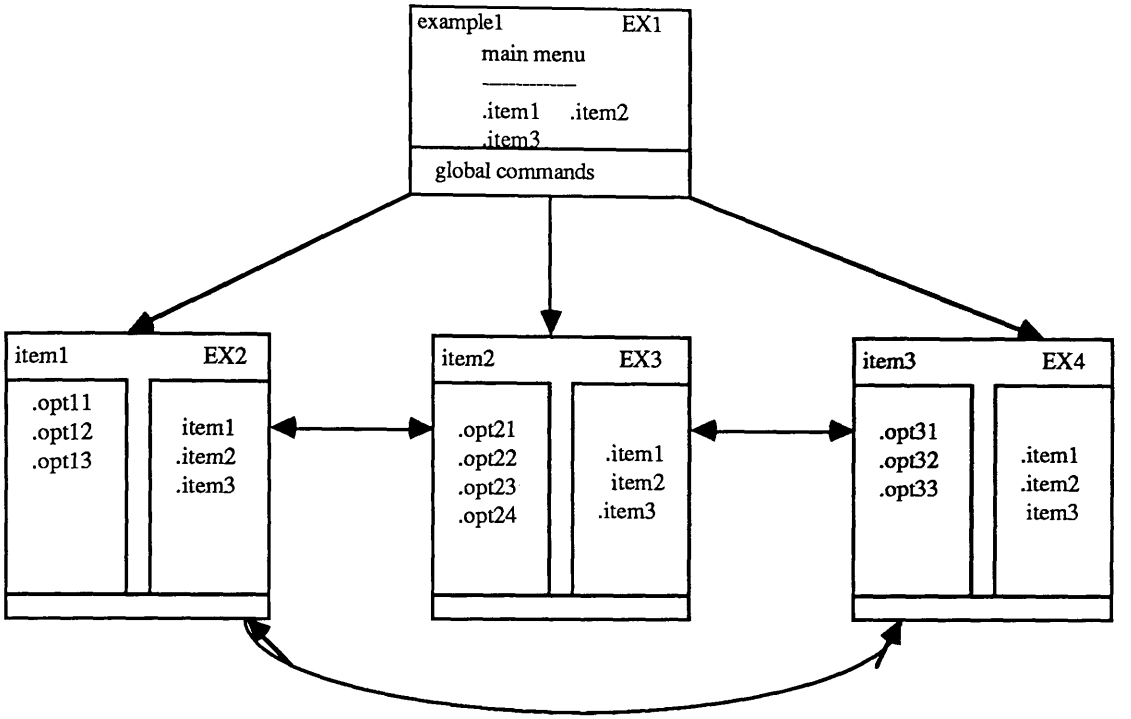
The limitation of one selection (one attribute combination) at a time implies that 4 frames have to be created (as in the first approach). But this number can be further reduced if only the items which are really needed can have their corresponding frames created. This means for example if (*A2-B2*) does not lead anywhere or is not a decision point then this item should be removed from the main frame. This will generate only 3 frames instead of 4. This removal is also motivated by the fact that the combinatorial method might cause the cluttering of the screen. However, an item can be added when needed.

Assume only 3 items (combinations) are available at this stage. Thus only 3 frames have to be created from *EX1* .

In this approach, two design alternatives emerged and considered

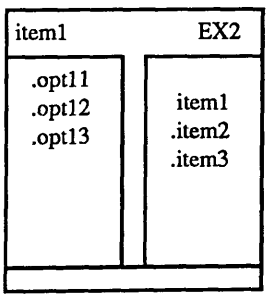
•*Alternative#1*

If the number of items in the main menu is very small and if it is possible to fit them altogether with the frame options within the display of this frame then the implementation of the parameter node concept is possible. This is illustrated below:



At level 2, the frames of the next (third) level or those corresponding to the options available have to be created and the cross-reference links for the main menu items have to be added as well.

Let's consider *EX2* for explanation.



Item2 and item3 are linked respectively to *EX3* and *EX4* . Selecting item2 at *EX2* would mean cancelling the previously selected combination (item1) and therefore changing the node or the path in the tree structure, in this case jumping to the frame *EX3* . This alternative illustrates the concept of parameter node in the sense that selecting another combination at any frame of level 2, would lead to the display of a frame which would have been displayed when selecting the same combination but at the first level (main

menu), this is to say, no explicit backtracking is necessary. Note that creating frames causes the increase of the depth of the structure, and creating links means creating cross-references to the adjacent frames.

•*Alternative#2*

When the number of items in the main menu is large, the fitting of this menu within any frame becomes inappropriate and inadequate. This means a purely hierarchical structure is created by allowing a single selection, and a different display for each frame. Therefore, the navigation or movement through the hierarchical structure is purely based on the navigation techniques or commands available in KMS.

The point discussed so far relates to one of the important design issues, a menu based system designer has to consider. This issue is obviously the user interface structure. Two approaches are given highlighting or illustrating this point and different structures are constructed in this experiment. For this particular example, no one seems to be better or more appropriate than the other since each of them has its advantages and disadvantages. Therefore, the choice of the structure depends on the scope of its application.

Beside the importance of the user interface structure, there are also many other important design issues to consider, especially the one related to the presentation layer of the user interface. KMS provides valuable techniques and facilities that can be used by the menu system designer to improve the presentation layer of the interface. These include the highlighting techniques and the direct manipulation features of the KMS system.

The concepts of frame and rapid response to item selections can be exploited if help facilities are needed to be included within

the *Dining Out In Carlton* example. However, giving instructions and providing help facilities for the example designed can have its impact on the overall structure. In effect, on-line instructions and help facilities can be provided within different frames. There could be an instruction or a help frame for each menu item. This means that a whole help structure must be created and can be huge and complex itself. Meanwhile, some other design features are purely under control and restriction of KMS . For example , the display rate and response time are important features of KMS that can not be handled by the menu system designer. Moreover, handling error messages, and allowing typeahead and short cuts schemes (chapter 1) cannot be achieved within KMS.

3.2. Using the example

This part of discussion will focus mostly on the way the user interacts with the KMS environment within which the previous example is implemented and how information is retrieved.

3.2.1.Starting KMS

In the KMS version 4D available on the Sun-3 workstation, users must enter KMS directly from the basic Unix shell. KMS can not be run from within the Sun View environment.

To start KMS: The user types the word *kms* <CR>. After a few seconds :

- The screen is divided into three windows, one small across the top for messages from KMS, and two large windows. In each of the large window a KMS frame is displayed.
- The home frame is in the left window, it is the base of operations in KMS. It serves as a top-level index to the user's area of the KMS database. This frame is automatically displayed whenever

KMS is entered. The first time the user runs KMS, a home frame is created for him/her. The frame will be called *user-login-name1* . But in (section 3.1.), the item index *example1* is created from the frame builder's home frame. This frame is very much like the Home Card in HyperCard (next target system)

- On the right window is one of the KMS information frames, which indexes some interesting features which can be used later by the user (on-line tutorial).

Throughout all this discussion, I have considered the frame builder (menu system designer) to be different from the system user (end user). Their home frames are different. But let's assume that the item index *example1* is added to the user's home frame. This means that the user can access the example's database directly from his/her home frame. However, this is not the only way for accessing the database, going directly either to the frame builder's home frame or the example's main frame (*EX1*) if their names are known to the user is also possible. Assuming the example's database is accessed, from the user's home frame by clicking on the item reading example1 to display the main frame *EX1* from where the information seeking process begins.

3.2.2. Browsing and retrieving

There are two different ways for accessing and retrieving information within a KMS database.

(i) item selection

The information seeking process can start from the main frame (EX1) by selecting appropriate menu items or browsing through the information space by selecting the navigation commands till the target information is found.

(ii) string searching

The example's database may also be searched for a specific string. This can be used via the search facility available within KMS, and if the string is found a frame containing all the occurrences of the string is created. These occurrences serve as links to the frames containing the strings.

4. Discussion

This section focuses mostly on what might be called the limitations or deficiencies of the KMS Interface and their impacts on both the application designer and the end user. Finally, some possible improvements based on recent research findings are discussed.

The major points considered are as follows:

- Frame and Commands concepts
- Selection mechanism concept
- System structure and navigation concepts
- Interface modification and interface level
- Error messages, error prevention and error recovery

Frame and Commands concepts

While interacting with the KMS environment, I found that the commands and the KMS concept of frame easy to use. Whereas, differentiating or distinguishing between frames was quite difficult except by the frame names and the contents of the frames.

This fact has also been reported in Mantei's work on disorientation problem in the Zog system [Mantei, 82]. I believe, however, this is due in part to the similarities of the frames (standardised frames) i.e. same formats, same commands, same location on one hand. On the other hand, to the very rapid display of the frames.

Yet, another major component which is worth considering within the KMS concept of frame is the use or presence of a standard set of commands at the bottom of the screen (frame) and the commands associated with the mouse cursor (labels). I found that the availability of the same set of commands at the bottom of the frame confusing, misleading and error-prone especially in the very first time (beginning users). This is due because, some commands are made available in inappropriate context such as: *save* and *rest* (restore), where there is no change made to the current frame, *undelete* , where is nothing to undelete, *home* , where you are already in the home frame and finally, *previous* , *next* where there is neither next nor previous frame to go to.

So, in order to prevent the user from any confusion and allow the dialogue to be more appropriate and more efficient, I believe, either removing these commands and make them visible only when needed and appropriate or make them unselectable (mouse not sensitive to these commands) could greatly enhance the user interaction with the system. The idea is well supported by Lieberman since it is used in his EZwin kit which is used to implementing a wide variety of interfaces [Lieberman, 85]. He also stated in his paper that using the mouse to select a command or displayed object in situations where it is inappropriate is a common source of error in menu systems, thus he suggested a dynamic control of mouse sensitivity or command visibility in order to prevent erroneous selections and which KMS does not handle very efficiently. Another drawback of the mouse sensitivity in the KMS interface is the negative effect of the immediate selection response, providing no cancelling or undoing the action taken.

When in empty space, the cursor is associated with three commands which are *back* , *line* and *rect* (rectangle). Unlike the

goto and *create* commands there is no implicit cancel to these commands. However, cancelling them is possible whenever another command is pressed at the same time which causes the system to ignore the function of the buttons pressed. This is not apparent at all to the user (I discovered it myself accidentally).

Another inconsistency concerning the "dialogue manager" is that when the cursor moves close to the frame name (upper right corner), the cursor is associated with four commands which are *goto* , *move* , *delete* and *copy*. The inconsistency concerns the first three commands, in effect when clicking on goto, the command is highlighted but nothing happen, when clicking on move, the command is also highlighted and a warning message which says the frame name can't be moved is displayed. Finally, when the delete command is selected, a prompt waiting for a yes/no to delete the contents of the frame, even the frame is empty (the contents has already been deleted or just created) is displayed. Once again, these commands should be removed or be context sensitive as it is stated before. In addition to all this, there is no way neither for aborting a command nor undoing the effect of some unwanted commands.

Selection mechanism concept

It is stated in the previous sections that single menu selection, and the display of one frame at a time represent the central aspect of the user interaction with the KMS interface.

From my own experience with the KMS interface, hence gaining more familiarity with it, I found the Interface rather restrictive and limited concerning the user's activities. This particular style is forced upon the application designer. In effect, the specific application carried out has used only a single menu selection scheme. Adopting this selection style together with the mouse as a pointing

device, KMS does not allow either the application designer nor the end user to use none of the *type-ahead* or *short-cuts* schemes.

However, KMS uses direct access and rapid response as its strategy. This implies that the frame should be known. Moreover, rapid response can have its negative effect on novice users who have not enough time to build a cognitive representation of KMS frameset. Mantei [82] reported that there were more complaints of users becoming lost at 9600 baud than at 1200 baud. As a consequence of all this and especially after carrying out the exercise, I believe that a multiple menu selection mechanism is more appropriate for the tasks that require several menu selections and these menu selections should be made bistable (chapter 1). Moreover, these multiple menu selections should taken into account only upon the user's confirmation.

Part of this idea is supported and evaluated in Dunsmore's study and reported by Shneiderman [86] where most of the subjects have preferred the *highlight-return* form to the *item-return* and *immediate response* forms. With this form, the errors made were very fewer but slightly slower than the immediate response form (the form KMS adopted).

Finally, to my knowledge, apart from Brown's work on controlling the complexity of menu networks, little work has been done on systems which permit multiple selections from the same menu, which could be in my opinion of a great importance for the design of user interfaces. In his paper Brown [82] presented some basic but very important structures that arise in most menu systems. These are inspired by top-down structured programming techniques, and include *1 OF N* , modelled by the case structure. The idea of a multiple selection scheme is also supported as he extended the 1 OF N structure to the *M OF N* structure. This structure is less

commonly used but is very useful. It allows a user to pick any number of entries (including Zero) from a list in any order. This is very important in application with no obvious, natural order for presenting things. In such cases, each user needs the freedom to make decisions in the order that seems appropriate at the time, given the user's specific knowledge, background and orientation with respect to the problem at hand.

System structure and Navigation concepts

The navigation concept plays a big role within the KMS environment. It represents the way of moving around different locations within the environment. This movement is made very fast and quick enough that the links provided by the KMS interface act like "magic buttons" [Conklin, 87]. Moreover, this feature makes KMS behave as a hypertext system (Chapter 1).

So, link following makes the navigation easy, straightforward and surprise free if the location within the menu network and how to get to specific places are both known. However, this is not the case all the time, i.e. the answer of where am I and how to get to X is not always obvious and sometimes can be very difficult to be aware of that frustration and desistment are the most common consequences for such situation which is known as the " disorientation problem" [Mantei, 82, Conklin, 87]. From this point, it is obvious that the navigation concept is directly linked to the system structure which is being navigated.

My own experience with the navigation commands within KMS showed however that some of them are still lacking of consistency in such a way that a novice user can be easily misled in his/her exploration of the system structure. I am referring particularly to the *next* and *previous* commands which have been

mentioned earlier, but this time the inconsistency concerns the way these commands behave or guide the user in his/her decision making process. The simultaneous use of the *next* , *previous* and even *back* commands when exploring a system structure which is not necessarily equivalent to the structure of the information being presented may very well result in a search task failure because of the unfruitful paths taken. This can disappear gradually when the user becomes more familiar with the behaviour of these commands. Moreover, a successful backing up to a recognised or a visited frame may help the restart of the task. Another user difficulty when navigating through a large structure is the difficulty in maintaining an overall understanding of the semantic organisation. This is due mainly to the way the structure is being viewed, where only one frame is viewed at a time. This is very like much seeing the world through a cardboard tube [Shneiderman, 86]. This forces the user to rely entirely on his memory for efficient exploration of the information space. Thus another problem in the KMS interface is encountered. It is known as a memory overload problem and which affects especially novice users. Two major problems related to the KMS interface have been identified and discussed in this section: disorientation and memory overload problems. It is obvious that the second one can cause the first one. The disorientation problem was the subject of Mantei's thesis where she concluded that the major cause for user disorientation was due to the interface structure.

Interface modification and interface level

Some specific points concerning the user interaction have been identified throughout the previous sections. It is known now that KMS provided a menu-based interface where most of the user interaction is via menu selection which is particularly suitable to

novice users. This means that the users are not in full control on the system nor can the application designer offer them such a possibility apart from invoking some agents (programs) and the freedom of choice of the menu items or commands. In addition to menu selection (ignoring the editor interaction for the moment) a sort of a conversation window only for system prompt and user response is also provided.

Thereafter, I believe that in order to generate a more appropriate dialogue and enter in a more effective interaction with the user interface, an alternate or a mixture forms of dialogue is required and suggested rather than base the user interface on one particular format. I am particularly suggesting that the alternating with a command-driven interface is essential not only on the KMS environment level but extending it to the operating system level (Shell level). This would allow the user to control and initiate an interactive dialogue instead of a menu item or answering to a system prompt.

In effect, while practising with the KMS environment which can not be run from within the Sun View environment, I had the impression as though my activities with the computer are limited and also obstructed or prevented from another environment (the Unix environment). Then exiting the KMS environment is necessary before shifting to the other environment.

Concerning the interaction style, the KMS interface can be considered as a one fixed and shared level user interface. It does not allow neither the novice users nor the experts ones to accommodate this level (changing the interaction style) at their will.

However, it does provide a valuable feature through its editor interaction though there is no separate editor. This feature represents the possibility of the tailoring or modifiability of a given

menu network. This means that the user may become an application designer or enter the designer mode. This modifiability is only supported at the frame level. This facility enables the user to represent his own understanding and referred way of dealing with the material of the net [Robertson, McCracken & Newell, 81]. But the ability to modify some structures may have some negative effects such as forgetting the changes made [Mantei, 82], causing the explosion of the overall network, whereas the freedom in linking may complicate some search or learning tasks [Shneiderman, 88]. Therefore, the application designer is provided with a frame protection facility which can avoid the problems above.

Error messages, error prevention and error recovery

The last point to discuss in this section is the one concerning the error handling within the KMS environment. Since, KMS is supposed to be everything to the user, where he/she encouraged to experiment and to explore the environment more freely. Thus, errors may be made at any time as a natural result of attempting to do a task [Lewis & Norman, 85].

Most of the errors which can be made when interacting with the KMS environment are principally due to the inconsistency of some of the commands discussed earlier. But the errors are minor because of the simplicity and the interaction style used. There are few situations where errors can be made. Principally, during an editing session, most of the errors made are minor and easy to recover from.

In effect, the *Restore* command is used to undo all the typing previously done, and the *Undelete* command is used only to undelete at most the last 32 deleted items, but before saving any changes made explicitly or displaying another frame, otherwise they

are inappropriate. These two commands can be considered then as error prevention or error recovery facilities. Another error-prone situation could arise whenever the creation of a new frameset with a non valid name is attempted. Therefore, KMS just ignore the action taken, displaying a warning message saying that the name was invalid. This also can be regarded as an error prevention scheme.

Finally, concerning the messages displayed or prompted to the user, most of them are explicit and understandable.
All this is seen mostly more beneficial and helpful from the user's point of view. But, KMS does not provide the application designer with any simple and possible facilities to handle the error cases himself, apart from may be a special language in the frames themselves are written, but this is not even recommendable at all.

5. Summary

I have discussed the most relevant points of one particular style of human-computer interface and outlined some important characteristics of this particularity and its impact on the design of user interfaces in general. I have mostly focussed on the user interface issues and identified some important problems and deficiencies in such interfaces. Therefore, I believe that reconsidering some design issues within this type of interfaces is undoubtedly necessary in order to improve the user interface both at the human and system sides. Although this, KMS has a great success over the years it took to be developed. The reason may be attributed to the simplicity of the interaction style and frame concept. Moreover, its success may also be attributed to the concept of hypertext systems which is taken very seriously in the recent years. In fact, KMS is considered to be a particular hypertext system: structured browsing system [Conklin, 87] even if it was not the type of system intended in

its early stages of development.

I have also stated that getting lost or disoriented in a menu network was a fact in KMS. This can be attributed principally to the misinterpretation of the user interface structure.

Different structures have been constructed for the same task because different ways of presenting the information are needed. The differences in the information presentation is motivated by the way or strategy for the simulation of the multi-attribute selection property. This leads me to formulate the following idea: Providing a better selection mechanism than the one used in KMS (single menu selection only) may lead to a better presentation of information, therefore to a better perception of the user interface structure which will surely improve or decrease the disorientation problem. I am particularly suggesting that a multiple menu selection mechanism may be used for this end. Moreover, improving the navigational techniques used can also have a great impact on the problem: providing or giving a global view of a menu network is greatly recommended and helpful in such systems.

It is understandable that the KMS system is intended to be used by novice and expert users, providing a single interface mechanism which is sufficient to support most computer functions needed by the user.

3.4. HyperCard

1. Description of HyperCard

When it comes to the amazing number of things that can be done with HyperCard, it is very difficult to describe it accurately. However, this can be considered as a personal toolkit that gives users the opportunity to use, customise and create new information using text, graphics, video, music, voice and animation. In addition, it offers an easy-to-use English-based scripting language called HyperTalk that allows users to write their own programs. Goodman [87] describes it as a multi-faceted authoring system in the sense that it allows the creation of proper applications and running others' applications. Unlike database managers, which store information into a predefined pattern or format, HyperCard permits browsing through information, cross-referencing and establishing new relationships between pieces of information. Bill Atkinson, the author of HyperCard, has described it as a "software erector set" that allows non-programmers to easily construct sophisticated interfaces [Conklin, 87]. Finally, HyperCard can be considered as a UIMS that can be classified among those which share a similar way of specifying the interface, but differs in the way that the underlying concepts of this class of UIMS are handled or supported. These differences are discussed next in terms of the concepts and entities which give HyperCard its originality.

2. Concepts and Entities

This section gives an overview of the concepts and basics which govern the HyperCard philosophy and also outlines some of the important underlying features. Typically, this section is

concerned with the way of creating, representing and accessing information within HyperCard.

2.1. Objects

Like many other new UIMS, HyperCard uses the concept of objects through which all the user interaction is performed and within which information is stored. HyperCard provides five different objects which are:

•Stack : This is the simple idea HyperCard is based on. A stack is a named collection of related cards. This can be seen as a disk file that serves as a HyperCard application.

•Card: This represents the on-line screen metaphor of any HyperCard information base or in other words, HyperCard's basic unit of information. A card may contain buttons, fields and MacPaint-like graphics combined in any way. In hypertext terminology, a card may represent a node (Chapter 1) within a HyperCard information space.

•Background : This is very similar to a card in the sense that buttons, fields and pictures may be contained within a background as well. A card has only one background, but a number of cards can share the same background.

•Buttons : They are the primary action parts of a HyperCard stack. They may point to a specific card or perform a complex task. These may be considered as links in hypertext systems technology (Chapter 1). There are two different kinds of buttons :

- background buttons, Which appear on every card associated with a given background.
- card buttons, which appear only on the card where they have been created.

•Fields : These are the place or recipients where only text is entered and stored. Like the buttons, fields are also of two types: background fields and card fields. A card can have several fields which can overlap one other to any depth.

Each of the five objects mentioned above has its own properties which allow the object to be handled as a separate and different entity. These properties may include: the object's name, object's number, object's id, object's style, object's script and link.

2.2. User interaction

Interaction with HyperCard (objects) depends on the user level. This means that different levels of use are provided in order to control the use of the objects. HyperCard offers five different levels of access which are discussed next according to level order.

- *Browsing* : This read only level enables users only to roam around the information space. At this level, only a few functions are allowed such as opening, copying and printing a stack..
- *Typing* : Beside all the functions allowed in the previous level (browsing) more new abilities are added at this level such as adding new cards and changing text within existing fields.
- *Painting* : Beside the functions allowed in the above levels, painting functions are added and background as well.
- *Authoring* : The ability to deal with the remaining objects (buttons and fields) is provided. All the five objects mentioned earlier as well as their underlying properties apart from the script one (see next) are made available to any non-programmer designer to become a stack author.
- *Scripting* : It is the highest level, thus providing more power to the user interaction over HyperCard objects. In effect, the script property that every object is associated with is exploited, that is a

script is attached to the object. A script may contain one or more handlers, where a handler is a set of instructions or commands that HyperCard executes in response to an action or upon the selection of that object. The language used for this purpose is called HyperTalk which is an object-oriented programming language like and English-based. It is obvious then that the higher the user level is the more power HyperCard provides since each level incorporates everything from previous levels as well as more added abilities.

3. Experience of using HyperCard

3.1. Design and implementation

As stated before, there exist five different levels of use of HyperCard. However, when it comes to consider HyperCard's environment, these levels may be collapsed into two major ones known as designer and user levels. HyperCard differs from other systems previously discussed in that it considers two types of designers: non-programmers and programmers. This section outlines these two types of designers but more focus is made on the second type which is related to the scripting level.

3.1.1. Non-programmer designer

All HyperCard objects are available and accessible at the authoring level. Therefore, people without any programming background may become a stack author. However, any user interface designed at this level is rather restricted and limited in that most of the interface components come in a predefined form. In effect, a stack may be created with very little effort, either by copying and pasting elements from other existing stacks, customising (changing a card's look), or by creating new objects as well as deleting existing

objects. As a matter of fact, an attempt to implement the *Dining Out In Carlton* example is made at the authoring level. At this level, HyperCard behaves very much like KMS. Therefore, the strategy to be used to implement the the example could be very similar to the one carried out in KMS. The multi-attribute selection property would be simulated in the same manner as it has been done with KMS. However, the card's size in HyperCard could be an obstacle for illustrating the parameter node as it has been done in the second approach with KMS. Therefore, the scripting level is provided to achieve or meet just that.

3.1.2. Programmer designer

It is at this level that the design of a more suitable and appropriate user interface to the *Dining Out In Carlton* example can be undertaken because the ability to attach a script to object is added and therefore more control over the user interface is assured.

In this exercise, two different approaches are considered in the implementation of the example. Each approach is illustrated and explained in a different designed stack. The stacks to be designed in each approach operate at three levels, and the cards composing the stacks must be created such that the hierarchical structure of the example is reflected, because HyperCard does not provide any underlying mechanism for such structures.

The first approach

In this approach, the stack consists of three cards (a card per level). The aim of this approach is to use the scripting power of HyperCard to handle the dialogue part of the example as well as the display of the information.

At level 1: The first level of this stack consists of displaying and presenting the different attributes in the main menu represented by the first card of the stack. The attributes are represented as buttons. A script consisting of one handler is attached to each button of this card and is executed upon the selection of that button. A script is attached to this card. Among the instructions of the button's script is a call to the card's script which is executed in turn. The card's script consists mainly of testing whether a chosen combination of attributes does exist in the database so that a second level information is displayed, and if not a message is displayed in the message box saying so. Some menu commands each performing a specific action such as quitting HyperCard or going to the Home Card are also created and represented as buttons, (figure 4.1).

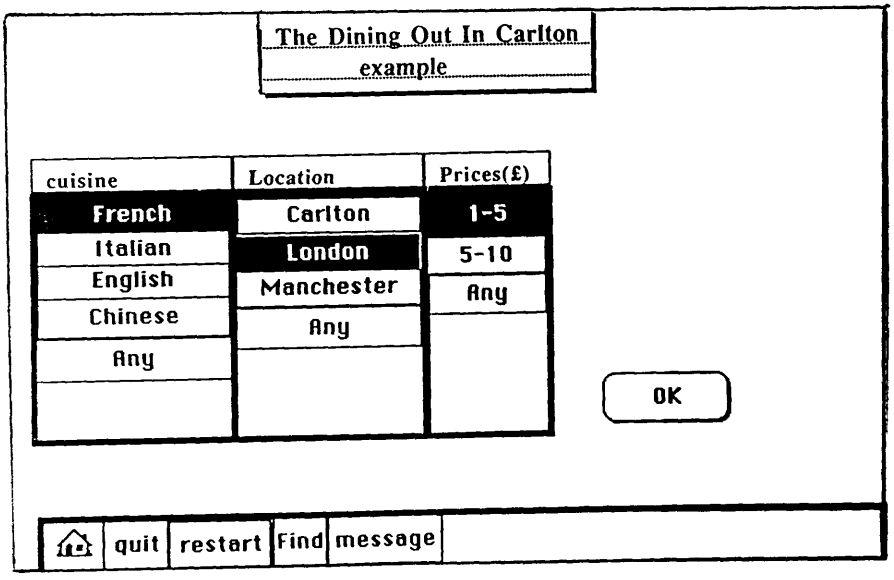


figure 4.1. Display of the attributes

On the storage side of this stack all the necessary information is stored in different fields composing a sort of a database. Unlike other systems, only one card (set of fields) may suffice to hold all this information. Therefore, the first card's script checks this database for every combination selected by the user. The card's script can be thought of as a procedure which takes 3 attributes as parameters and displays the corresponding card containing the available list of restaurants if the parameters are valid. The validity of the parameters is expressed by their existence in the database. And the selection process at the first card (main menu) can be expressed as follows:

```
while (the combination selected is not in the database) do
display message "This combination is not available";
another selection;
end;
```

This cycle is repeated till the combination is found and the next card is displayed. At this level, a menu item can be cancelled by selecting it again, thus the attributes options are made bistable (chapter 2). The three attribute selection is not taken into account till upon the user's confirmation, that is to select the *OK* option. It is clear then, that the multi-attribute selection property is perfectly achieved and illustrated by figure 4.1. The probability of selecting an existing combination is 1/N where N is all the possible combination of three attributes, that is a selection is an element of the Cartesian product of A1 x A2 x A3, where A1, A2 and A3 are the attributes. This method may lead to user frustration and loss in confidence about the potentiality of the retrieval side of the user interface. Therefore a more convenient and consistent interface is required for increasing the speed of the multi-selection process.

At level 2: This one card level is designed such that most of the navigation aids and techniques reported in Chapter 1 are supported. These include selective retreat, parameter node, stability and so on, on one hand, in the other hand, some basic stack navigation commands such as going back, going to the Home card, find and message are also added. This card consists of a set of buttons designed to handle the previous choices (parameters from the main menu) and some fields (figure 4.2). One of the fields is made scrollable in order to handle the list of available restaurants which is ought to be lengthy. Whenever a list of items is displayed, the possibility of cancelling and reselecting one of the parameters previously chosen from the main menu (buttons) is given at this level, so no explicit backing up the hierarchy is needed and an updated list of items is displayed in the same field in consequence (figure 4.3). This illustrates the selective retreat as well as the parameter node concepts.

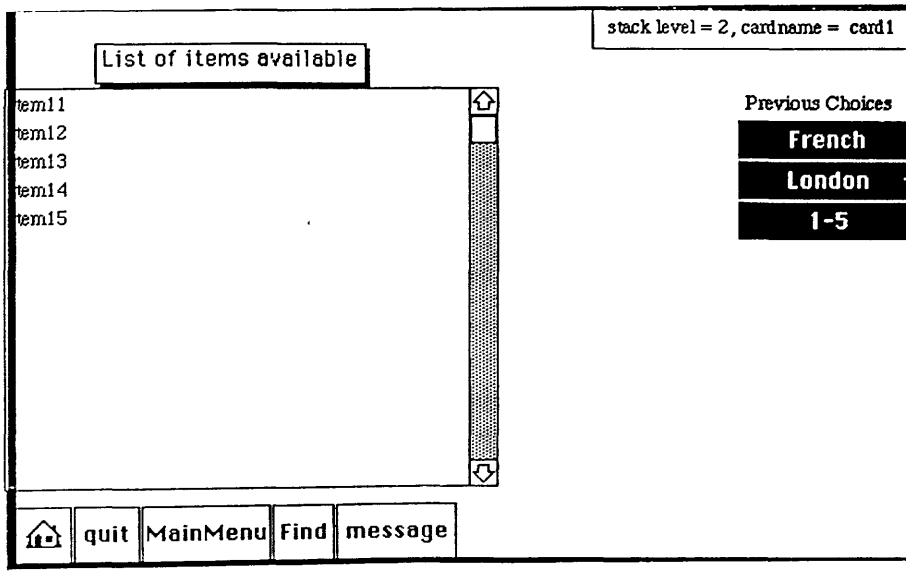


figure 4.2. Display of the available list

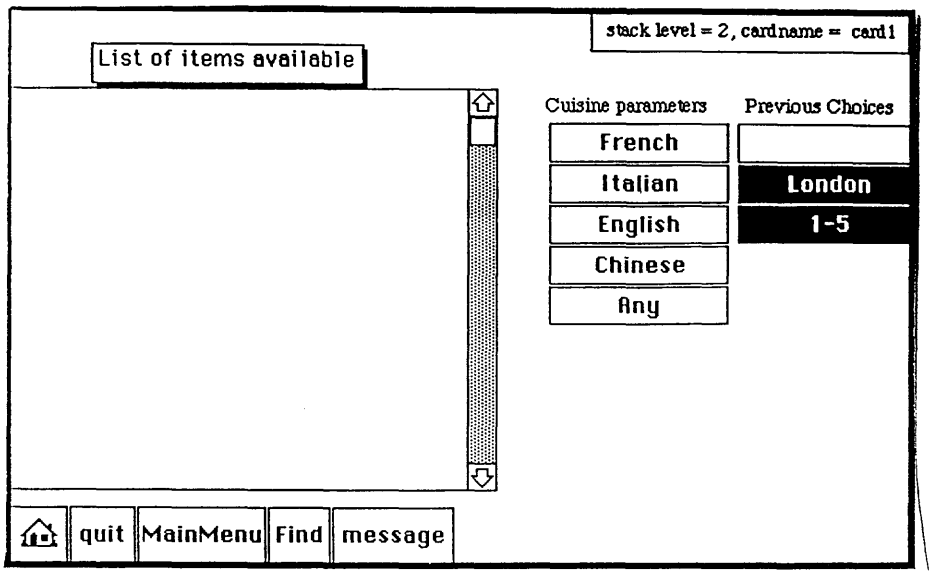


figure 4.3. Display of the cancelled parameters

But, going back to the main menu is dictated whenever cancelling more than one parameter is needed. The problem encountered at this level and which can be considered one of the weakest features of HyperCard is that text (more than one word) within a field cannot be made explicitly selectable. In order to render text within fields selectable, one method is to use transparent buttons overlap the text which is about to select. The drawback of this method is that buttons have a fixed position, therefore making it impossible to cover all the list of items displayed in the scrollable field.

An eventual improvement for this method is to specify the item selection by entering the number of the item via a keyboard making therefore the text selection independent of the fixed position of the buttons. However, to avoid using the keyboard as a means for item selection, another technique is adopted, that is to use the "one word field selection" method supported by HyperCard. Therefore, the handler intercepting this selection must be within the field script.

This method is likely to be more appropriate to the application carried out. In any case, a selection has to be made at this card in order to proceed to the lower level.

.At level 3: This level is one card level as well. The card represents the information page of the item selected at level 2. It consists of previously selected parameters represented as buttons, navigation buttons such as going to the main menu, and a field where the detailed information is displayed (figure 4.4).

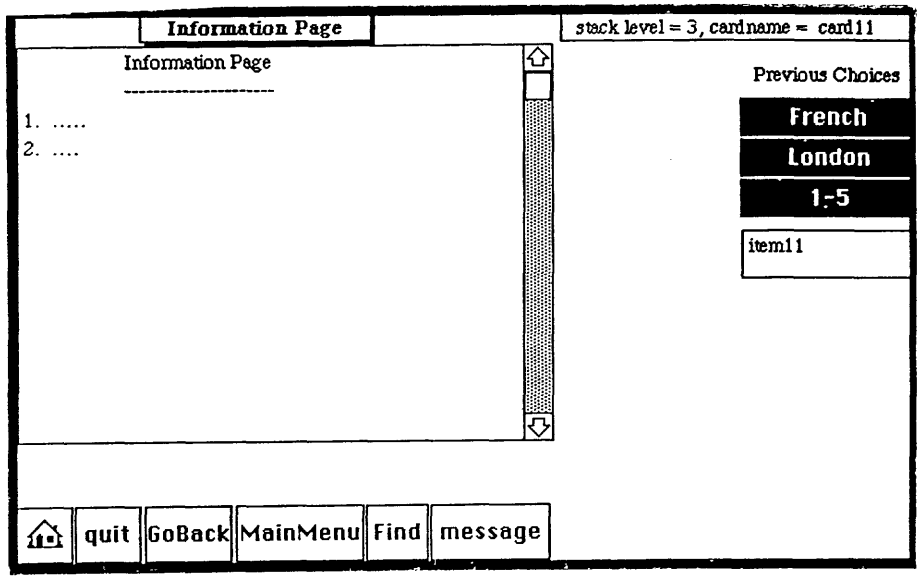


figure 4.4. Display of the target information

The implementation of the example in this approach is similar to the Chisl implementation. But, the instantiation and sideways viewing techniques which were perfectly illustrated in the Guide implementation and more or less in Chisl are not possible using HyperCard. This may be due to the small card's size and the display of one card at a time.

The key issue in this approach is that most of the user dialogue as well as the displayed information are controlled by and within the scripts of the different objects composing the three levels of the stack. The communication between these objects is via message passing (handlers). Form the designer's point of view, the main implication of this approach is that, the programming of the dialogue part of the interface is quite complex because great care and attention must be paid in order to assure a good and surprise free object communication. I believe, this complexity is mainly due first to the creation of a great amount of objects which are uniquely identified and handled. Second , to the lack of efficiency in the way the text field are considered. Hence, the need for only simplifying the programming task of this application emerged and another approach which exploits the concept of background is carried out and illustrated in the design of the second stack.

The second approach

In this approach, a second design alternative is undertaken at the second and third levels of the hierarchy of the example only whereas the first level remains similar to the one in the first approach. Therefore, only the design of these two levels are discussed.

At level 2: In this case, instead of creating only one card where the user dialogue is controlled by the script of the different objects of this card, a different card is created for every possible existing combination chosen at the first level. Thereafter, a number of different cards sharing the same background constitute this second level of the stack. A card at this level may have same number of objects with same purposes as the card of level 2 in the first approach i.e bistable buttons to handle the previous choices,

navigation buttons and a field containing the list of available items to choose from (figure 4.2). On the storage side of this stack, all the necessary information is made available within the fields of the cards composing the stack, that is each card is filled in with its specific information.

At level 3: For each item selected at level 2 (from a given card) is associated with a card at this level. Therefore, this level consists of a set of cards each of which is referred to as the information page of the item previously selected. These cards are created with the same background which consist of the previously selected choices represented as buttons, some navigation buttons and a field where the whole detailed information is displayed. Therefore, apart from their properties, the only difference between these cards is the field content (figure 4.4).

The implications of this approach are: simpler programming task despite the awkwardly way the selection mechanism is made, and huge number of cards, therefore large stack structure.

Finally, an eventual third approach may envisaged in order to improve the multiple attribute selection mechanism adopted in the above approaches. In this eventual approach, only the possible existing and needed combinations should be made available to the user, increasing the probability of getting to the right information and in less search time. This alternative might be called a context-sensitive selection mechanism, that is whenever, an attribute is selected then highlighted, all the other attributes but those logically linked to the one or ones selected are removed. This approach has not been implemented in this exercise however.

3.1.3. Using the example

Like the designer level discussed earlier where two sublevels of design have been identified, three sublevels may be identified in the user level as well. This may represent the key difference between HyperCard and the systems previously discussed. In effect, the end user is recognised at three different levels of use (browsing, typing and painting). This recognition is expressed by the fact that the stack author may restrict the use of the stack by deciding to which kind of use the stack is intended to. More importantly at this level is the way the stack is accessed to and how the information stored is retrieved or read. Unlike, Guide, KMS and many other interactive systems which allow generally at most two different ways of moving around its underlying information space, HyperCard provides a third valuable way (message box) which can be seen as a natural language interface like. Three ways can be used to conduct a search task within the implemented example.

i. item selection

The information-seeking process begins from the first card (main menu) of the stack where the user is required to make more choices towards meeting his/her search task goal (figure 4.1). However, this form of interaction is more suitable for user who has a well defined and understood retrieval task, for example the user has a specific combination to which he/she requires more details. This method of searching is more likely and preferably to be used with the first implemented approach of the example because all is controlled by the different scripts which display the relevant information, but eventually can be used in the second approach as well.

ii. String searching

The search task may be carried out only for looking for a specific string within the stack information space. In a situation where the end user does not care about none of the attributes (cuisine, location and price), but interested only in a particular dish eg. *fish*. Therefore, the string searching method can be used to search the information space of the stack. Thus, this method can only used in the second implemented approach of the example. By this means, the user can see all the information pages (cards) containing the word *fish* . This can be done by using the standard *find* command in the provided menu, i.e *find "fish"* .

iii. Direct access card

Some basic navigation commands are included in the display of every card. Among the commands is the *message* command which is principally used to issue or send commands to HyperCard via a message box. The user conducts a more or less natural language dialogue with the user interface. This gives a HyperCard user interface its power over those implemented with the systems previously discussed. In effect, the end user can directly go to a known or recognised card if he/she knows that the card contains the relevant information. This command can be issued as follows: *go to card card1* . This method is also not suitable to be used in the first approach.

Finally, concerning the two implementation approaches undertaken is this experience and discussed in the previous section, a relatively small and modest comparative evaluation on time acquisition or mean traversal time is carried out. The two approaches has shown no significant difference in the mean traversal time (from level 1 to level 3), with and without parameter cancelling

at level 2 (19 and 10 seconds in average respectively). However, when cancelling a parameter at level 2 by using the selective retreat, stability as well as parameter node instead of explicitly backing up to the first level (using go back) increases the mean traversal time significantly.

4. Discussion

The major points concerning HyperCard and which seem to have great and direct impact on the user interface design are: Object, Structure/navigation and User level concepts.

Objects concept

Unlike many design environments, HyperCard offers an object-like environment based on the concept of objects. It allows a user interface to be specified and designed differently as it would be in other traditional programming environments. This concept of objects is inspired from the object-oriented programming methodology, therefore inheriting most of its advantages such as reducing the cost of building user interfaces by preventing the designer from all the low level details of the interface and increasing the consistency and power of the interface in consequence.

Any HyperCard application (stack) is simply programmed by creating the objects which represent the interface itself. The interface is either graphically specified using the direct manipulation approach, i.e many objects have associated semantic routines (scripts) can be invoked and used directly by a designer who is not necessarily a programmer, or by using a special-purpose language (HyperTalk). In either cases, the interface objects are handled and dealt separately, therefore debugging, testing and modifying the interface are made simpler and easier. I strongly believe that the use

of objects has great impact on the design decision process of user interfaces and particularly on menu-based ones. Therefore, some design issues have to be taken or considered at some level of the design of the object.

Consistency in layout and design of the stack, as well as consistency in the background are important design issues in HyperCard, that is to choose the appropriate background and not over-design the background because this may confuse and frustrate the user. The fact that the primary object which is likely to be the most understandable and visible is a card makes the display or presentation of the information within a card or a group of cards another issue, that is it should be consistent and efficient. As a matter of fact, a button is an element of a card and also most of the user interaction with the stack is via the buttons, Therefore it is very important to consider carefully the design of such buttons and the design issues at this point may include consistency in the use of the standard HyperCard buttons, feedback, and with the Mac interface.

However, this present experience with HyperCard has shown that some objects are lacking consistency and more functionality which have affected in certain situations the design process of the example undertaken in section 3.1.2. This has affected particularly the dialogue part of the interface. There was a difficulty in choosing a more appropriate and elaborate selection mechanism at the field level where textual links (Chapter 1) are not supported. Moreover, only one single font is allowed within a given field.

Because HyperCard lacks true inheritance, sometimes too much effort is required to represent the dialogue in a suitable form, eg. only one object at a time is selected, moved, deleted, copied or pasted if changing the interface layout is needed.

The other inconsistencies or deficiencies related to the use of objects in HyperCard appear in the following situations: when the name of stack is changed (within a script) all the links made between any card in the named stack and any other card in any stack are broken. Therefore, all these links must be redone after changing the name of the stack. Sometimes, a small change in a part of the dialogue may affect the overall dialogue. Finally and since only one card is displayed at a time and the material in a card is not scrollable this may have two consequences: first as in KMS the disorientation problem may arise and secondly, multiple windowed user interfaces are not supported by HyperCard.

Structure and Navigation concepts

It has been shown from the previous experiences with other systems that there exist a close relationship between the structure (how the information is organised) and the navigation process (how the information is accessed and retrieved). The design issues considered in this direction in the stack design are therefore directly related to nature of the information and its organisation. Information can be stored in a single stack or in a group of connected stacks that are closely related, loosely related or virtually unrelated.

Because HyperCard does not support any particular structure mechanism due to the way the cards are arranged when created, deciding on the best approach becomes a design issue at the structure level. Different informational stack organisations may exist. These include: linear (sequential), hierarchical, non-linear and a combination of these. Each type of these may influence the way the stacks are organised, therefore the way they are navigated. Navigation issues arise at this level and must be considered in order to reflect the underlying structure.

A hierarchical stack organisation implies that the end user has multiple options at many points in the navigation process. As a design issue at this level is then to consider how the hierarchical structure and the navigation aids are reflected. Assigning a different background to each set of related cards (path) in the hierarchy is useful and helpful to narrow the gap between the designer's model and the user's mental model which might exist, and also reducing the risk of getting lost as seen in KMS. The navigation aids needed for this end not only require forward and backward buttons but also links buttons to other points in the stack or in other related stacks. However, great care must taken when using the standard HyperCard navigation buttons and specially when using the Go menu. In effect, some sort of command inconsistency may occur when using for example the *back* , *next* , *previous* , *first* and *last* commands, because these are related to the order in which the cards are created and which do not reflect the underlying structure at all. Therefore, for a more appropriate movement within the hierarchical structure, the stack designer should consider more specific handlers within the scripts of the navigation buttons and hiding the menu bar as well as disabling the use of the power keys from the casual browser.

A good built in feature that might be used to reflect more the organisational structure of the stack is the visual effect feature such as *wipe down* , *up* , *left* or *right* . This also is true when considering the other two stack organisations which require a sophisticated level of linking and planning. So far two very important points considering the design of a stack are identified along with their intrinsic relationships. Yet another point to consider in this direction is the point whether creating separate stacks and linking them at appropriate points in the navigation process or create only one stack but with different background for the different types of

information. This is known as stack vs background principle. The design process at this point considers for example the retrieval speed, scripting effort where a single stack is more suitable to meeting these requirements. Whereas, multiple stacks would be more appropriate if the information could be subdivided towards meeting different types of users with different needs.

Moreover, if the user navigation would be made via the standard HyperCard *find* command, therefore a single stack is dictated because this command does not work across stack boundaries. However, this can solved by writing a more specific handler within the script of the find command.

Finally, HyperCard lacks several features that would qualify it as hypertext system in the full sense such as bidirectional links and graphical browser.

It uses the *recent* command to display only miniatures of at most the last 42 cards visited. This is mainly used as one means of a direct access to a given card, but if this card is recognised. This does not replace the graphical browser facility where the different relations between cards and stacks would be apparent. Moreover, if the cards miniaturised by the *recent* command have resembling looks (eg. same background) the recent facility would be therefore without any need at all.

User Level

The user level concept in HyperCard may be just one solution or one way for achieving the issues related to the end user as a part of the design process. In effect, a stack author may restrict the use of the stack at different levels. At the first two lowest levels (browsing and typing), only the read-write access is given to the user but still restricted from changing the stack structure. At the painting

level, users can still change the appearance of the stack but not its functionality. However, a stack author can still allow a user to change a part of the stack structure if necessary. This may be at the authoring level, where the end user may enter the design mode. Therefore, great control on the user's access level and the identification of the type of the user have to be included in the stack design process.

Different ways may be used for this end. Either setting the user in a script or scripts, or intercepting and preventing an effort by the user to modify the script. This means, a script can monitor and modify the user's access level accordingly. And finally, by using the HyperCard protect-stack facility which permits either the complete protection of the stack or just private access.

5. Summary

In this experience, I have discussed one of the most recent software systems that can be considered as a major breakthrough in the family of user interface creating tools. Very often, this involve the creation of menu-based user interfaces. Although the full power of HyperCard has not been explored, and despite of the limitations encountered while carrying out the task in section 3.1.2, I believe however from the outcome of the experience, many important issues on Human-Computer Interface design have been raised. Undoubtedly, HyperCard and HyperTalk together provide a powerful programming environment that is rich in functionality.

Chapter 3

Conclusions

1. Summary

One of the most commonly used interaction techniques in Human-Computer Interfaces has been discussed and surveyed in this thesis. The menu selection technique, which continues to flourish because of its simple interaction format and its adaptability to the many diverse applications has contributed significantly to the widespread acceptance of menu-based user interface systems despite their inherent disadvantages and drawbacks (chapter 1). Chapter One has addressed particularly the navigational problems encountered by users of menu selection systems, and identified various navigational aids as well as other important design issues that a menu system designer should take into account toward a design of an effective menu-based system.

It is often argued that the menu selection technique was a cumbersome method of finding one's way around a system, and only novice or casual users may benefit from it. However, despite the inherent disadvantages of menu systems, menus have been shown to offer one solution to the problems encountered with other interfaces such as command-driven and natural language interfaces as reported in chapter 1 . However, their value depends on the degree of cognitive assistance and ease of implementation that they provide. This offers a significant challenge to the menu system designer to ensure that the user's needs and abilities are properly considered.

Four different menu specification systems have been discussed and described in chapter 2 . Each of which has adopted a different approach to implementing menu-based user interfaces. These systems are motivated by the need to make the user interface

cheaper and easier to design and implement. Apart from the Chisl specification system (discussed in chapter 2 section 3.1), the remaining three systems (here HyperCard is considered at the authoring level) use the Direct Graphical Specification (DGS) approach for the design and implementation of menu-based user interfaces. The advantage of this approach is that it allows the menu system designer to place text (Guide, KMS and HyperCard) and light buttons (HyperCard only) on the screen using a mouse and see exactly what the end user will see when the application is run. Currently, Guide supports only a small part of the user interface design task, it cannot be used to help control the display and manipulation of the real application data objects. A drawback common to all the systems used was their inadequacy for implementing and managing user interfaces requiring a multiple selection of items from the same menu which has been shown to pose a major challenge to these conventional menu specification systems.

Strategies to solve or address the multiple selection mechanism problems as well as some the navigational concepts discussed in chapter two have been devised and used within each the four target systems. The use of the Chisl specification system and HyperCard (here it is considered at the scripting level) has highlighted the need for a menu system designer to be a programmer in order to be able to design and prototype a suitable user interface to the *Dining Out In Carlton* example. HyperCard required the use of a special-purpose language (HyperTalk) to handle the stated problems as well as the semantics of the menu application, while the Chisl specification language was required to handle or define the interaction techniques (local and global buttons) as well. In its present form Chisl is therefore not appropriate for user interface designers who are not programmers.

2. Further work

Improving human-computer dialogues has been and still is the most recognised and important objective within the human-computer interaction area. A lot of improvement has been achieved in the recent years, but there's still a lot more to be done. From the menu system designer's point of view, improvements in menu-based user interfaces have been concentrated mainly on the implementation aspects of the menu system, that is most of today's user interface tools (eg. the last three target systems discussed in chapter 2) use the direct manipulation approach and more recently the visual programming methodology to build menu-based systems, thus makes the design and implementation tasks much easier and quicker.

From the user's point of view, however, emphasis has been on improvements in the user interface aspects such as the presentation as well as the structure layers in order to improve the user/menu system interaction. But, menu systems still suffer from two major complaints, namely the difficulty in navigating accurately and efficiently the menu system structure, and the difficulty in accommodating or addressing the user's skill levels. In effect, it has been noticed from experience of using the target systems (chapter 2), that the way information within a menu system is organised and made available, affects the strategies used to access this information. The multiple-attribute selection scheme as well as the underlying information space structure highlighted the need for more techniques which should address the navigation problem, as well as the need for the user's skill level to be included in the user interface in order to improve the user/menu system interaction. Desirable improvements may include, for example, large display surfaces in order to allow a better perception of spatial relationships between

the menus (frames in KMS, views in Guide, DUs in Chisl and cards in HyperCard). This might not only be beneficial from the navigation point of view (since it would provide the user with the ability to both determine the approximate location of the goal and the effort required to reach it), but also from the expert user's point of view as well, because he/she could use this spatial relationship as a means of speeding usage of a menu system. Moreover, the highly repetitive series of mouse movements and button pushes which must be executed in a menu system (KMS and HyperCard at the authoring level) may feel increasingly slow and annoying as the user becomes more skilled. One solution would be to allow sequences to be encapsulated as "macros" invokable by a single action on the keyboard or using the mouse. Techniques of macros consisting of keystrokes already exist and are applicable in some menu systems, as illustrated in the BLT (typeahead) approach, but techniques for recording a series of mouse movements and button pushes need to be developed and used and especially within direct manipulation interfaces.

A particular problem highlighted in this study is the multiple selection problem, which was found to be unachievable unless a special purpose language was provided within the underlying design environment, as was the case with Chisl and HyperCard. Where available, the multi-selection scheme can be used as a decision-making process reducer, therefore reducing the menu structure complexity and enhancing user's performance. Finally, since most the target systems studied in chapter 2 are considered to be hypertext systems, I believe that menu-based user interface systems will benefit from most of the improvements made in hypertext systems technology, since they share many the HCI design issues.

References

AKscyn, R. M.; McCracken, D. & Yoder, E. A.
"KMS: a distributed hypermedia system for managing knowledge in organisations," *Communications of the ACM*, July 88, Vol 31 (7), pp 820-835.

Apperley, M.D. & Field, G.E.
"A comparative evaluation of menu-based interactive human-computer dialogue techniques," *Human-Computer Interaction*, INTERACT'84, B. Shackel, ed. 1984, pp 323-326

Apperley, M.D. & Spence, R.
"Hierarchical dialogue structures in interactive computer systems," *Software-Practice & Experience* , Vol 13, 1983, pp 777-790.

Arthur, J. D.
"A descripitve/prescriptive model for menu-based interacton," *Int. J. Man-Machine Studies*, 1986, Vol 25, pp 19-32.

Arthur, J. D.
"Partioned frame networks for multi-level, menu-based interaction," *IEEE expert*, 1985, pp 34-39.

Bellingsley, P.A.
"Navigation through hierarchical menu strucutres, Does it help to have a map?," *Proceeding of the Human Factors Society*, 26th annual meeting, 1982, pp 103-107

Bennett, J.L.
"Tools for building advanced User interfaces," *IBM Systems Journal*, Vol 25, no 3/4, 1986, pp 354-367.

Brown, J.W.
"Controlling the complexity of Menu Networks," *Communications of the ACM* , Vol 25, no 7, 1982, pp 412-418.

Brown, P.J.
"Interactive documentation," *Software-Practice and Experience*, 1986, Vol 16, 3, pp 291-299.

Brown, P.J.
"Guide User Manual," Computing laboratory, the university Canterbury, July 1987.

Buxton, W., Lamb, M.R., Schuman, D. & Smith, K.C.
"Towards a comprehensive user interface management system," *Computer Graphics* , Vol 17 (3), July 83, pp 35-422

Conklin, J.
"Hypertext: An Introduction and Survey," *Computer* , sept 87, pp 17-41.

CR, Acm press,
Computing reviews, acm press, January 1988, Vol 29, 1.

Goodman, D.
"The two faces of HyperCard," *Macworld*, October 87, pp 123-129.

Gray, J.
"The role of menu titles as navigational aid in hierarchical menus," *SIGCHI* , Januaary 1986, Vol 17 (3), pp 33-40

Guedj, R.A.
"Remarks on some aspects of man-machine interaction," *Methodology of interaction , IFIP* , 1980, Gueddj et al., eds., pp 235-238

Guevara, K. & Newman, W.
User Interface Design. A course developed by Beta Chi Design Ltd.1986, pp 3-17.

Hammond, N. & Barnard, P.
"Dialogue design: Characteristics of user knowledge," F*undamentals of human-computer interaction* , A. Monk (eds) 1984, pp 127-164.

Hepe, D.L., Edmondson, W.H. & Spence, R.
"Helping both the novice and advanced user in menu driven information retrieval systems," *People and Computers : Designing the Interface* , ed. P. Jonhson and S. Cook, Cambridge University Press, 1985.

Hodgson, G.M. & Ruth, S.R.
"The use of menus in the design of on-line systems: A retrospective view," SIGCHI , 1985, Vol 17 (1), pp 16-21

Hopkins, D., Callahan, J. & Weiser, M.
"Pies: Implementation, Evaluation and application of circular menus,"

Jacob, R.J.K.
"Using formal specification in the design of human-computer interfaces," *Communications of the ACM* , 1983, Vol 26 (4) pp 259-264

Karat, J., McDonald, J.E. & Anderson, M.
"A comparison of menu selection techniques: touch panel, mouse and keyboard," *Int. J. Man-Machine Studies* , 1986, Vol 25, pp 73-88.

Koved, & Shneiderman, B.
"Embedded Menus: Selecting items in context," Communications of the ACM, April 86, Vol 29 (4), pp 312-318.

Lewis, C. & Norman, D.A.
Designing for errors. User Centered System Design, D.A. Norman, S.W. Draper (eds) 1985, pp 411-432.

Lieberman, H.
"There's More to Menu Systems than Meets the Screen,"
Computer Graphics , Vol 19, no 3, 1985, pp 181-189.

Mantei, M.
"A study of Disorientation behaviour in Zog," *Ph.D. thesis* 1982, University of Southern California.

Miller, D.P.
"The depth/breadth tradeoff in hierarchical computer menus," *Proceeding of the Human Factors Society* , 25th annual meeting, 1981, pp 293-300.

Norman, K. L.; Weldon, L.J. & Shneiderman, B.
"Cognitive layouts of windows and multiple screens for user interfaces," Int. J. Man-Machine Studies, 1986, Vol 25, pp 229-248.

Norman, D. A.
Cognitive Engineering. User Centered System Design, D. A. Norman, S. W. Draper (eds) 1985,

Norman, D. A.
"Design Principles for Human-Computer Interfaces," *Readings in Human-Computer Interaction, a multidisciplinary approach*, R.M. Baecker, W. Buxton (eds), 1987, pp 492-501

Perlman, G.
"Making the right choices with menus," *Readings in Human-Computer Interaction, a multidisciplinary approach*, R.M. Baecker, W. Buxton (eds), 1987, pp 451-455.

Robertson, G., McCracken, D. & Newell, A.
"The Zog approach to man-machine communication,"*Int. J. Man-Machine Studies* (1981), Vol 14, pp 461-488.

Schultz, E.E. & Currain, P.S.
"Menu structure and ordering of menu selections: independent or interactive effect," *SIGCHI* , 1987, Vol 18 (2), pp 69-71.

Shneiderman Ben.
"Designing the User Interface. Strategies for effective human-computer interaction," Addison, Wesley (eds) 1986, pp 42-52.

Shneiderman, B, & Marchionini, G.
"Finding facts vs Browsing knowledge in hypertext systems," *Computer*, January, 1988, pp 70-80.

Snowberry, K., Parkinson, S.S.R. & Sisson, N.
"Computer display menus," *Ergonomics* , 1983a, Vol 26 (7), pp 699-712.

Snowberry, K., Parkinson, S.S.R. & Sisson, N.
"Design Methodology for menu structures," 1983b, pp 557-561.

Snowberry, K., Parkinson, S. & Sisson, N.
"Effects of help fields on navigating through hierarchical menu structures," Int. J. Man-Machine Studies , 1985, Vol 22, pp 479-491

Simmons, R.F.
"Man-Machine Interfaces: Can They Guess What You Want?" *IEEE expert*, 1986, pp 86-93

Tennant, H.R., Kenneth, M.R. & Thompson, C.W.
"Usable Natural Language Interface through Menu-Based Natural Language Understanding," *Proceeding CHI'83, conference on Human Factors in computing systems*, 1983, pp 154-160.

Wood, C.A., Gray, P.D. & Kilgour, A.C.
"Experience with Chisl: a Configurable Hierarchical Interface Specification Language," *Computer Graphics Forum* 7 (1988), pp 117-127

Appendix A

This Appendix explains the Chisl specification language syntax and shows how dialogues are encoded with this syntax. Then, the preprocessor specifications are discussed.

A.1. The Chisl syntax

A Chisl dialogue consists of a sequence of dialogue units. A dialogue unit consisted of a sequence of options.

An option has a name, a location, a condition, and an action sequence. Moreover, an option can be either local or global (see section1).

The interpretation and execution of the Chisl dialogues are performed by the Chisl system interpreter called : " *Chip* ".

The display generated by " *Chip* " consists of four panels:

- Control panel: is the top panel through which the root dialogue is specified.
- Button panel: is the panel where the local options appear
- Global panel: is the panel where the global options appear
- Text output panel: is the panel where the output text action is displayed.

This is how the elements of an option are specified :

A name of an option is specified by :

B_<*option-name* >, which will have a selectable screen button.

An option is either local or global, so a global option is identified by a "%" as follows: B_<*option-name* >%

The option or button is given a specific location within a panel, specified by the (X, Y) coordinates of the top left corner of the button as follows:

X<a> Y<b> a,b are two integers.

The button can be tested for selection, so it is placed in a selection condition specified by: {B_< *option-name* > }

Finally, the option action sequence is principally a set of pre-defined actions for dialogue specification.

These include:

- General actions

  quit() causes termination of a Chisl dialogue.

  exit! exit from the current DU, returning to the calling DU

- Register manipulation

  assign(<register>,<string>) which assigns the value <string> to <register>.

  reset(<register>) reset the register to the constant UNDEFINED.

  reset_all() reset all the registers.

- Output

  message(<id> , <x>, <y>, <string>) places <string> in message number <id> and displays it at <x>, <y> in the interaction window.

- etc...

Registers can be tested in a combined way using the boolean connectives AND , OR , NOT.

Moreover, the action sequence could include or be a call to a dialogue unit.

This is an example of option:

{B_item1}	X0 Y6	B_item1	D1[];
{B_item2}	X10 Y10	B_item2	assign(reg9,item2);
{B_quit}	X0 Y0	B_quit%	quit();

This means that, if the button whose name is <item1> , displayed at (X0,Y6) in the button panel is selected then the dialogue unit D1 is called or activated. Or, if the button whose name is <item2>, displayed at (X10,Y10) in the button panel is selected then the string <item2> is assigned to the register9. Finally, if the global option whose name is <quit> , displayed at (X0,Y0) in the global panel is selected then the Chisl dialogue is terminated.

A.2. The Chisl preprocessor specifications.

The user defines a sequence of textual files which is then translated into an executable specification (Chisl) where a prototype has been generated from the specification itself.

The Chisl preprocessor is called "*PreChisl* ". It is implemented in C, on a Sun Workstation.

A.2.1. Description of the PreChisl files.

The textual files created are called " *PreChisl files* " whereas the files containing the Chisl specification are called " *Chisl files* "

In the current implementation of the PreChisl preprocessor, there are three types of files.

A.2.1.1. Attributes files.

There is only one file of this type for each dialogue or information system (only three attributes are supported for the time being).

Such a file consists of a sequence of blocks where each block is composed by seven (07) items and defined as follows:

- item1: a string of characters which is used to identify an option and represents a value of the attribute1.

- item2: a string of characters which is used to identify an option and represents a value of the attribute2.
- item3: a string of characters which is used to identify an option and represents a value of the attribute3.
- item4: a string of characters which is the name of a file which should contain all the information needed upon the selection of (item1, item2 and item3). This file contains only ordinary text.
- item5: a string of characters which is the name of a dialogue unit (see section1) into which the contents of the file referred to by `item4` are translated to Chisl specification. So, the file `item4` is called a PreChisl file and the file `item5` is called a Chisl file.
- item6, item7, item8: are all strings of characters which are the names of the files containing an icon image which should be displayed into the graphical window upon the selection of item1, item2 and item3 respectively.

So an Attributes file consists of a sequence of such blocks where each item should be in a separate line of the file (for simplicity). Each block present in the file results in a corresponding frame being displayed on the selection of the first three items(i.e. the values of the attributes).

A.2.1.2. PreChisl DU files type1.

There are as many DU files as there are blocks in the Attributes file. These files are referred to by `item4` in each block. Each file consists of a sequence of three item blocks, where the items are defined as follows:

- item1: a string of characters which represents an option in the page or frame displayed.

- item2: a string of characters which is the name of the file which should contain all the information needed upon the selection of the option `item1` (it is also called the information page).
- item3: a string of characters which is the name of a dialogue unit. This dialogue unit is constructed by translating the file referred to by 'item2' into a Chisl specification

A.2.1.3. PreChisl DU files type2.

There are as many DU files as there are blocks. Each file consists of ordinary text which corresponds to all the detailed information needed about the previous choice.

Note that only one item should be in a separate line and no space between strings or before the first character of the string is allowed because Chisl does not provide otherwise. However, any space required should or could be replaced by the underscore (_) character in order to make the options more readable and clear.

A.2.2. The PreChisl file structure.

As seen from the description of the PreChisl files, a hierarchical structure is being built.

Each PreChisl file corresponds to one dialogue unit file, apart from the Attributes file where a root dialogue unit must be specified. So, the PreChisl file structure is hierarchically organised as the dialogue or the information system which is being built.

Appendix B

This appendix gives an overview of the menu command which compose the command dialogue.

B.1. The Menu Commands

When Guide is run for the first time by issuing the guide command, the following window appears on the screen.

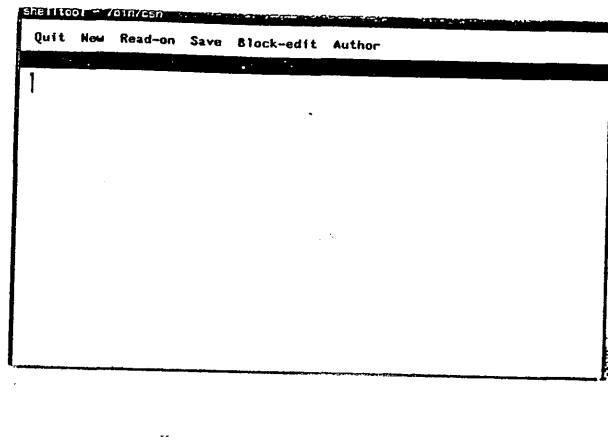


figure 1a.

The main menu consists of the following commands :

Save : save either a text or a source file during a guide session.

Block-edit : used for moving, deleting or copying block of text and/or picture.

Quit : used to end a guide session or if there is more than one view, to delete the last view.

New : used to add new source file(s) to the source. This can be done in three ways:

- completely replacing the original source
- adding a new view
- inserting the material within the existing source.

Read-on : used to advance forward and backward within the

frame-of-view (scrolling)

author : used to enter the author mode

So selecting the author command from the menu of figure 1a, an extra menu commands is added to the main menu as follows:

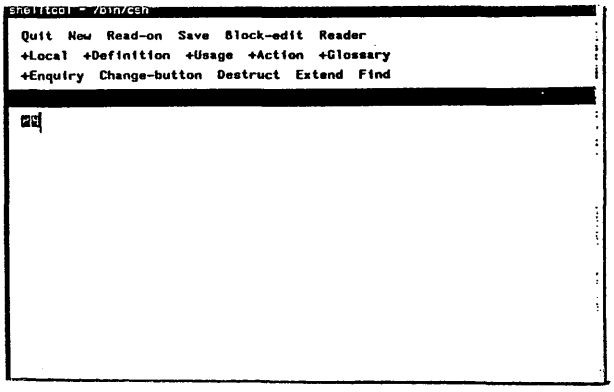


figure 2a.

+Local, *+Definition*, *+Usage* and *+Glossary* : are used to create the different buttons mentioned earlier.

Destruct : is just the opposite of the four commands above(only the structuring is deleted, not the text or picture)

Change-button : used to change the name, type or asking-level of a button.

Find: the command searches for a string of characters defined by the user either within the names of replace-buttons(button search) or within the complete source(complete search).

Action :this command gives the author extra power and flexibility in constructing the replacement of the button. An action button is a Unix shell command.