THE META-HELP Browser: Towards a Practical Interface on-line Documentation (1)

Andrew Law

1986

Cognitive Studies Research Papers

Serial No. CSRP. 063

The University of Sussex
Cognitive Rtudies Programme,
School of Social Sciences,
Falmer, Brighton, BN1 9QN

The META-HELP Browser : Towards a Practical Interface to on-line Documentation [1]

Andrew Law

Cognitive Studies Programme
University of Sussex
School of Social Sciences
Arts E
Brighton BN1 9QN

1. Introduction

This paper considers the problems and possibilities of providing an interface to on-line help facilities in a large AI software development system. Such systems and the interface developed to access on-line help have much in common with other large information sources such as public view-data systems (e.g. PRESTEL) or on-line library catalogues.

Some of the properties of an "ideal" interface and information source on which it would operate are described. This description is based on an analysis of the nature of large software development systems and users' needs. The construction of such an interface at present is considered impractical for two reasons: firstly, it requires techniques and knowledge beyond the state of the art and secondly, the documentation of many existing environments is incompatible with such an interface — the aim of the paper is to outline ways in which existing systems can evolve towards the provision of more adequate help. However, the analysis highlights users' needs and the problems of providing an ideal interface, and specifies some of the properties required of less ambitious interfaces.

Menu interfaces are simple to implement and are believed to have several advantages over other interfaces (e.g., command based interfaces). However, several hitherto unrecognised problems are involved in implementing menu interfaces in large, complex information sources. These involve: first, problems of navigation and speed of travel through the menu network and second, their lack responsiveness to users* immediate tasks and general levels of experience.

Having discussed the problems of both types of interface, a third form, the META-HELP browser, is described. This is an unintelligent, menu based browser. It is considered to have all of the advantages of other menu based interfaces but overcomes some of their problems. It is seen as a step towards full specification of the properties of an ideal interface.

Overview of **Paper**

Section 2 provides a brief description of AI work where the need for documentation and interfaces to the documentation are identified. In Section 3 there is a brief description of the the sorts of tasks with which an ideal interface and information source would need to cope. The interface envisaged is an intelligent knowledge based system whose construction is presently beyond the state of the art of Cognitive Science. Having considered some problems with the development of intelligent interfaces, and the poor state of existing on-line documentation in some help systems, Section 4 considers the possibilities and problems of providing menu

interfaces to on-line help. There is a selective review of research on menu interfaces and a discussion of its implications for the design and implementation of menu interfaces in this domain. It highlights fundamental problems which have not hitherto been considered. In Section 5 there is a discussion of the development of the META-HELP browser interface. This is an implementable, unintelligent menu based browser for on-line documentation. This interface is considered to embody the advantages of menu interfaces as well as overcoming some of their problems.

2. AI Environments : Why on-line documentation and why the need for an interface?

In order to understand the problems involved in producing an interface to on-line information it is first necessary to briefly describe the nature of AI work and AI environments.

In AI it is common practice to use the *process* of exploratory programming and debugging to solve a problem. As Khabaza (1985a) points out, programming to further understanding, rather than to produce a useable, efficient product, is in sharp contrast to conventional programming, where clear design specifications are usually required before programming begins.

This rather unusual style of programming demands a special form of environmental support and documentation. There is usually a profusion of facilities available in the system and (as shown below) each of these has many properties about which the user may need to know. There is often simply too much information to be remembered by the user, therefore it needs to be stored in a manual or on-line. The user's primary demand is fast and easy selection of the different forms of information about these facilities.

Many of these environments have been developed by industrial or university research departments (e.g. LOOPS, ART, KEE, POPLOG, etc.). Novel computational facilities, perhaps not describable in traditional computer science terminology, are frequently added to the system, necessitating extensible on-line documentation sources.

POPLOG, a multi-language AI software development system, contains over 1500 on-line documentation files, which vary widely in levels of detail but which are organised into four general groups:

HELP — explanation of facilities to the average user.

REF — more detailed account in the style of a technical reference manual.

DOC — long overviews of general facilities such as the editor, primarily intended for being printed out and used as a manual.

TEACH — tutorial introductions to a wide range of subjects from fundamental facilities through to tutorials on AI concepts.

See Appendix A for an example of a short HELP file.

The user can access information by a call specifying the group name followed by the topic name e.g. "help ved" or "ref numbers[11]. However, the user must know the name of the file she requires, or search through large indexes, guessing the relevance of a file to her problem from its title.

Existing Problems

Through our informal observations it is clear that many users prefer to find a human expert than to search for information themselves, even though the documentation may contain the information they require. The difficulty of accessing information via current command driven interfaces may be related to Fisher, Lemke and Scwab's (1985) report that 40% of the functionality of some complex systems is unrealised and/or unused.

Khabaza (1985a) has illustrated a variety of reasons why users may be unwilling to use these command based interfaces.

(a) When using command based interfaces the user must know the name of the facility she is looking for. However, there is often no naming convention in the description of facilities provided in AI environments - so the user often cannot guess the name of the facility she wants.

(b) The documentation is often "uneven" that is, some areas are well documented, others are undocumented. Because of this unevenness, the user cannot know whether some facility will exist, or whether it will be documented.

(c) Even if the user knows that the facility exists and is documented, she may not know how it will be classified, and therefore where to look for it.

(d) Indices of help files are often incomplete. This often misleads users into thinking that existing facilities are not present. It is very difficult to update the help file system consistently because when some information is added, no procedure exists for determining which help files and indexes should be updated to reflect the change. This also means that some help files may not only be incomplete, but also incorrect because they refer to a previous state of affairs.

(e) Related to (a) and (d), the user may not know the name of the desired facility. This can lead to a great deal of fruitless searching through indexes.

3. The Structure of the 'Ideal' Interface

This section discusses the problems involved in the design of an "ideal" interface and information source by considering the needs of users. The phrase "information source" (as opposed to on-line documentation) is used here as the interface may need to draw upon other forms of more abstract machine readable representations than simply textual. The most optimistic and ambitious view of the interface presented here is one which could replace the human expert in situations where a user needs (though may not ask for) some form of help with her activities. In this case the interface may be expected not only to find files but also to generate new text (or diagrams or speech etc.).

3.1. The Knowledge Base

The nature of some aspects of the ideal interface are discussed in terms of what sorts of knowledge might be needed and how it should be organised if it were to be realised as an intelligent knowledge based system.

This knowledge was informally "elicited" by several methods. First, introspection; the contributors to this analysis are a mixed group of experts, novices, tutors and students, and are all major contributors to the existing documentation [2]. Second, as part of an Alvey project, Aaron Sloman has begun an analysis of some of the kinds of knowledge required by a sophisticated programmer. This analysis has helped us identify some of the types of help that are required in the various stages of the software design process. A third source was provided by analysing the

results of a local query answering service. This system allowed users to present their queries to the system, these were converted to mail messages, logged and answered several days later. However, it is recognised that this is a biased source of data as only a limited set of users used this service (those that could operate it and had problems that could wait a few days to be answered). Finally, the process of building and critiquing the META-HELP browser highlighted different aspects of knowledge involved (see Section 5).

The knowledge involved is discussed under two broad headings:

— *Domain specific* (what an expert should know about the system)

— *Extra-domain* (what a helpful expert guide should also know)

3.1.1. Domain Specific Knowledge

This concerns knowledge strictly related to the target domain, i.e. what on-line information sources should contain

Range and Nature of Problem Domains

Because of the nature of AI work, the information source has to cover a wide range of topics. For example, there needs to be information covering issues that concern the editor/s, library packages or the possible interactions with the operating system. Some AI environments contain multiple languages (for instance, in POPLOG there are three languages; POP-11 (see Barrett, Ramsay & Sloman, 1985), Common Lisp and PROLOG, with the possibility of another object oriented language in later developments). Each of these languages need to be individually documented as well as the possible interactions between them and information common to them all.

The "grammar" or syntax of each of the languages defines a potentially infinite set of possible procedures which might or might not be in the library. Therefore it is not possible to have a finite, pre-determined collection of documentation concerning what information might be requested. The interface should be able to generate text from some general representations. It may also need to recognise that, although what the user requests does not exist, there is a facility which might be transformed, or used as a building block. A human expert can often suggest things the user should look at to see if any of them would help. However, this often requires considerable creative insight on the part of both parties, insight which, as yet, we cannot simulate.

Within each of the domains mentioned above there are variety of problems that could arise. These relate to various stages in the software design process: In the design and implementation stage the user may need to know *what* a procedure does or *how* it does it. For example, early in the design process a user may wish to know what facility could perform some function (e.g., she understands what sort of thing she needs but does not know the name of it in this system), at a later stage she may have identified the required procedure but may not fully understand its precise operations or she may know the name and precise function of some procedure but she does not know how it is formally specified (e.g., what is the order of the arguments?). In other stages of the design process a user may have queries concerning program debugging or program efficiency. It is possible that each of these forms of queries will require different forms of representation of the information in question.

So, there is a wide domain of knowledge that needs to be present in the information source. The knowledge also needs to be structured in a rich enough

manner so as to be used by the interface, and recognised by the user, as a basis of an answer for **a** variety of different types of problems arising at different stages of the design process.

Types of Questions

Within each domain and type of problem the questions can be of several general types e.g.;

General : *'Why doesn't my program work', 'What's a good way of solving this type of problem'*

Specific : *'Is there a facility that does X\ 'Where can I find out about Y'*

Also, a user wanting information about a particular sort of procedure may present the question in terms of a high level formal specification of the desired behaviour, an implementation-dependent specification, or an actual example of the behaviour, e.g.;

*'Is there **a** procedure which when given an association table and a key will return the associated value?'*

*'Is there **a** procedure which when given a list and an item will return the element of the list immediately after X?'*

*'Is there **a** procedure which when given AGE and the list*

[NAME TOM SEX MALE AGE 33 JOB BRICKLAYER SPOUSE MARY]

will produce the result 33?'

There are severe difficulties in representing information so that the interface can recognise the relevance of a single facility to each of the questions (in this case association tables). Information could be organised and presented according to the constructs and facilities provided (the *system's* viewpoint), or according to the possible applications to which they might be put (the *user's* viewpoint). Often the latter would be more helpful for the user, but anticipating the set of all possible requests in these terms would be horrendously difficult, if not impossible, considering the infinite generative power of the languages involved.

Forms of Presentation

Both the questions and answers could be given in various different (though not necessarily mutually exclusive) forms e.g.:

— pointer to file/s
— graphical
— textual
— a program, program fragment or program schema

Given that different presentations may be appropriate under different circumstances, two additional major problems arise: First, identifying when a particular form of presentation is appropriate. This decision process is likely to be guided by extra-domain, pedagogical knowledge. The second problem concerns the

difficulty of translating between different forms of presentations (either from the questions form of presentation to the information representation, or from the information representation into the answers form of presentation). Unique problems arise in this domain even with natural language processing. For instance, there are many problems in coping with multiple syntactic ambiguities and with enriching the grammar to allow sentences in which program code is intermixed with English (e.g., see Allport, 1984).

3.L2, Extra-Domain Knowledge (what it is to be a ^helpful' expert)

From the discussion above, it is clear that a richly structured knowledge base concerning information about the software facilities will not be enough to provide a helpful interface. Other sources of knowledge will be required, some of these are discussed below.

Negotiation About the Nature of the Problem

From our informal observations of novices and experts it seems unlikely that the form of interaction between the interface and the user can, or should, be a uniform unidirectional process, e.g.:

user's question — >
interface's understanding — >
interface's answer — >
user's understanding

In many cases of human-human interaction it is clear that users do not initially ask the right question. In some cases users may not be clear about what the problem is (e.g., a debugging problem), they may know what the problem is but they may ask for the wrong sort of help, or they may mis-identify the problem and hence ask the wrong question, for example:

7 *have a problem but dorCt know what it is, or what question to ask**

7 *have a problem, I know what it is but I*m not dear about how to describe it*[1]

7 *have problem X and need answer concerning Y'*

(when in fact they have problem Z and need answer concerning P, or they have problem X and need answer concerning P)

To solve such problems, as well as having detailed domain knowledge, it is also necessary for the expert and user to negotiate about what each of them thinks is the problem and why. That is, the expert and user need to co-operate in the diagnosis of the 'real' problem and the reason for it arising. In the case of the third example the expert not only needs to negotiate what answer is required, he needs to be able to identify that negotiation is required in the first place. This period of negotiation is crucial in the following processes that human experts engage in:

(1) Identifying the problem (helping the user to find the problem and the question).

(2) Clarifying the problem originally identified by the user, explaining it and helping the user to generate a more appropriate question.

(3) Identifying the real problem and appropriate question, understanding how the user made the mistake of identifying the wrong problem and hence the wrong question, and explaining this to her.

It is unlikely that information will be required only when the user asks for it. A really intelligent interface would need to be able to "look over the user's shoulder". Part of the knowledge involved in this process would be the system's sensitivity to the context in which a request is made. For example, a human adviser might mention the efficiency issues only after noticing that the user is writing code which could produce a lot of garbage collections. More generally, like any intelligent interface, it would need to be able to build a "deep" user model — a task currently beyond the state of the art.

Sensitivity to the User's Needs

The various ways the questions illustrated above can be asked should reflect aspects of the way it will be answered. For example, we should not provide an answer paralleling the level of abstraction of the first question illustrated above, to a user asking the last question. Therefore, if the interface is to be able to cope with a variety of users with differing levels of expertise, it is likely that some tutorial or explanatory facilities will need to be provided — as well as some decision procedures for recognising when they need to be utilised.

The work of Kidd (1985) on the analysis of "what users ask" and its relation to the design of more appropriate expert system architectures indicates that the design of expert systems that respond sensitively to users is a non-trivial problem.

3.2. Problems with Incomplete Implementations

The are several problems involved in producing interfaces which are semi-implementations of this ideal interface. If they have limited power they may be inconvenient to use — though of course it is possible that they would be used more effectively than present interfaces. Users may find that simplistic models of themselves are restricting, (say models based on broad classes of types of user, or simplistic rules such as "if user has seen document Y, X times, then they understand the concepts introduced in document Y and should not be referred to it again"). This may be especially annoying to users if the system is insensitive to their individual rate of development or progress. These restrictions may be greater than those provided by "inflexible", unintelligent interfaces.

However, these systems may not simply be inconvenient to use, such implementations may be harmful, particularly if they are the only means or the best means of accessing information, or where they have become the means upon which users come to rely. For example, a system which has no information about the context in which a user is working may be "destructively" helpful. A question concerning how to reverse the elements of a list may be inappropriate if the procedure which produced the list in the first place could have have been written to construct the list in the reverse order. If we implement a system which does not take into consideration the context of the questions it is asked, we must be certain that the question is clearly understood by the user and that the user has identified the appropriate question. These contraints will not always be fulfilled.

If we rely on such a system as the only means of help, (i.e. only when the user asks a question is more information given), then a user may "make do", and not consider developing her understanding any further. Hence an intelligent help guide which just answered question presented to it could lead to the development of

bad programming styles, limited conceptions or even misconceptions about the system. This is not to suggest that unintelligent interfaces would not produce these problems, rather that we should not assume that the more intelligent an interface is the more flexible and useful it is.

3.3. Prognosis

It is clear that the state of the art in Cognitive Science does not provide us with the knowledge or techniques to produce the ideal interface. However, there is research that is relevant to the solution of some of the problems involved in developing such an interface. This research includes the investigation of expert system architectures that will support multiple representations and inference techniques. The work on blackboard architectures is clearly relevant here. The research into expert system architectures that support mixed-initiative interactions will also contribute to providing systems that can support more complex forms of user-system interaction. It is possible that we will have to develop "deep" representations of the domain in question and the user. Therefore the research into deep modelling techniques will also be relevant. However, it appears that the major problem will not be in the development of adequate techniques and formalisms, but in the identification and organisation of the knowledge involved. The HCI research on user's understanding of programming languages will be relevant to some aspects of this problem.

It is also clear that the on-line documentation of many exisiting programming systems would be incompatible with the use of such an interface. The documentation is often not complete or richly enough structured.

However, it is also necessary to acknowledge that there is a demand for more adequate, implementable interfaces to existing on-line documentation. In the next section there is a discussion of the use of menu based interfaces to on-line documentation.

4. Menus — The Simple Answer?

This section presents a selective review of some research on menu interfaces and discusses its implications for design and implementation. While it is shown that menus have many advantages over command based interfaces, this section also highlights some fundamental problems with the use of menu interfaces which have not hitherto been considered.

4.1. Menus and their Advantages

Menus have been put to two main uses in the interface:

— "View Data" systems (e.g. PRESTEL)
— Menu driven command language interfaces
(e.g. ZOG, see Robertson et al (1981)

Both forms of interface offer the user a limited set of options. In the first case the user chooses between different domains or items of information she wishes to look at. In the second case the user selects a command she wishes to invoke (the distinction is not that clear cut; selecting some piece of information indirectly involves issuing a command to invoke that information).

Some of the advantages of menus are (Norman, 1983) that they are fairly transparent in their use and require little, if any, training to introduce the user to their basic operations.

In presenting a list of appropriate options they capitalize on the advantage of recognition as opposed to command languages which rely heavily on recall, which can be a problem even for experts in large complex systems. Well structured menus should also encourage the development of a spatial map or model.

The user learns where to expect certain types of information; eg. references to text files giving examples of a particular domain may always be found at the end of the menu concerned, or introductory references at the beginning. If there are no references in these places then the user can conclude that further searching is pointless. This enables the user to know what is not available as well as what is and begins to overcome the problem of users not being able to know what is and what is not documented.

In hierarchical menus the user can begin her search with broad category descriptions leading to menus with more precise definitions. This means that she can begin to search for information about some domain without knowing how it is named within the system, provided that the broad category names are well chosen for the task. This begins to overcome some of the problems associated with users not knowing the names of functions because of the lack of naming conventions.

Therefore from a technical point of view menus seem to offer a feasible, solution to the problem of providing an interface to a complex domain — they are relatively easy to implement. They also seem to have many advantages over command based systems, and overcome many of the problems facing the designer of an interface to on-line documentation in existing software development systems. The next section examines some of the research on menus for implications for menu design and in order to clarify the question of their utility.

4.2. Research on Menu Interfaces

Much of the empirical research on menus has produced results which are difficult to assess in terms of utility when considering design and implementation issues. Where there are design implications, they have addressed "low-level[11] factors (size of menu frame, effects of different select character types, number of items in a menu frame, etc.). More importantly, the problems of applying menus in specific situations (in particular in large complex view data systems with many different users searching for information for different purposes) have not been fully considered. It is suggested that these offer serious problems for traditional menu interfaces.

Although menus appear to have useful qualities there seem to have been very few empirical comparisons of menus with other interfaces. An exception to this is a the study by Whiteside et. al. (1985). They compared seven different interface designs (representing command, menu and iconic styles). The menu system in this study produced the lowest performance and subjective reaction scores for all users. That is, compared to the other interface styles, menus were liked the least and produced the worst performance. However, the authors strongest conclusion was that interface style was not the important factor in interface design (e.g the difference in performance is not a question of the disadvantages of menu interfaces compared to command systems per se), instead they suggest that 'the care with which an interface is crafted is more important than the style of interface chosen* (p. 190). However, they provide no indication as to what "good crafting** is.

Rather than compare menu interfaces to other forms of interfaces, it is possible instead to identify what are likely to be some problems with menus *per se.* A major problem with menus is that, even in moderate sized systems, the trees to be navigated can be very large. This can lead users to forgetting where they are,

getting lost, make mistaken choices etc. Several researchers have attempted to define features which aid performance in large trees.

Gray (1986) assessed whether menu titles had any significant effect on performance. He concluded that in small systems they had negligible effect, but with larger (deeper) systems, while there was no increase in speed of performance, there was an increased accuracy of performance. However, the reasons for this increased accuracy are not reported and it is not possible to assess them from the data provided.

Perlman (1985) assessed whether select key types (the items which have to be typed in or otherwise selected to invoke the command or other items) had any effects on performance. He found that select key/item compatibility enhanced time performance, and that single letter select keys matched with the first letter of the item to be chosen were the optimal type. He also found that sorted menu items (e.g. alphabetical and numerical) decreased performance time, suggesting that users may use simple searching strategies. However, in the design of many menu systems such conclusions will not be that helpful. In some systems there would be many items beginning with the same characters meaning that the intitial character could not be used as a select character. In other systems it is often not the case that there can be a simple, predictable, one word description of the item to be selected. As mentioned, in AI software development systems there are few clear naming conventions. This means that users cannot predict the function of a command from its name. Hence a user cannot predict an item's initial letter, and so they cannot predict where it is likely to be located in the menu frame. However, this does not mean menus should not be used. Users may find it much easier to be presented with a list from which to choose since an option's neighbours help clarify its meaning.

These results, while potentially conclusive for some systems, can only inform designers about low-level details of implementation. Other work has more directly assessed the problem of the complexity of menu networks. In particular, it has concentrated on what is known as the depth/breadth trade-off.

Miller (1981) compared the effects of four different menu structures on performance;

(a). 1 level 64 choices at this level
(b). 2 levels 8 choices at each level
(c). 3 levels 4 choices at each level
(d). 6 levels 2 choices at each level

Miller found that configuration (b) produced the fastest performance and the fewest errors. He concluded that the number of levels in a hierarchical system should be minimised, but not at the expense of display crowding (as with (a).). However, Snowberry, Parkinson, and Sisson (1983) repeated the experiment and showed that the first configuration could produce the fastest performance and the least errors if the items in the one level menu were categorised. This is related to the point made above, that the meaning of an option can be disambiguated by other options available. If the range of options is very large then the process of disambiguation is intractable. However, if the options are well organised, then each option can be disambiguated by its local neighbours. Some problems of categorisation have already been discussed in relation to Perlman's work, where it was suggested that it is often not always possible to provide a simple alphabetical or numerical sorting of the items in a menu frame. However, there are other problems concerning

what design criteria should be used for the categorisation and description of menu items. These are discussed in Section 4.3.

4.3. Some Problems With the Research

This section outlines additional sorts of problems that arise out of the use of such systems, and questions whether they can be dealt by using traditional menu interfaces.

Large Domains

Most of the studies mentioned above have involved "toy" menu systems with at the most 64 items. Many large software development systems would require menu interfaces (whether command or view-data) much larger than this. As mentioned POPLOG has over 1500 cross referenced documentation files. If, for example, menus had eight references in each frame then this produces approximately 200 nodes. Although there is some evidence from Tullis, (1985) that the "breadth is better than depth" slogan holds for large systems too it is not clear that the problems which arise can be ameliorated simply by juggling with the structural parameters of the menu system.

Complex Structures

All the studies mentioned above have dealt with simple hierarchical, tree structured menu systems. In many cases the structure cannot be so simple. For instance many of the POPLOG files are heavily cross referenced, producing a tangled network rather than a hierarchy. This is partly because the same information can be seen as having different hierarchical structures imposed upon it, depending on the needs of the user. Each file may refer to documentation about some function that can be used in a variety of ways. The menu structure will need to reflect this rich structuring. It is not enough to provide only one route to the item, but several — i.e. a heterarchy is needed rather than one hierarchy. It is also likely that the network will be "multi-dimensional"; not only will users need to access the same information via different routes, but they may need different types of descriptions of the same function. For example, a user may want to have a description of the purpose of some function, she may know the name and function of some facility but only need its syntactic specification, or the user may be a novice and would require more of an introduction to the function than an would an expert. Several related problems arise from the characteristics of large complex information sources.

Navigation

One major problem involves supporting the user in navigating complex networks. Users may need navigational information, firstly because of size; it will be easier to get lost in larger systems. The second reason is that in such large systems, with a diversity of users, it may not be simply that menus offer the advantage of recognition over recall, but also that users may be entering into entirely novel domains. They may want to browse through the information, exploring from a node to look for the relevant information and conveniently returning to that node if not satisfied. However, unlike the ideal interface, with menu based interfaces it would often be necessary for users to examine information relating to their specific interests before they found exactly what they wanted — thus it could (inadvertently) act as a pedagogical device. They may want a general picture of the sorts of domains covered beneath the current level (and in networks, above and parallel to it). A user may decide that she has made a wrong choice and wants to quickly review the other choices at that point, without necessarily having

to follow the paths previously selected to return to it. Then again, after browsing through alternative options, she may want to return to a point, reached earlier, via the quickest route, or even in one selection step.

Cognitive Mismatch

Many of the studies mentioned above report that when given a goal to find a certain item, users often make errors in selection. As shown above, altering the structural parameters of the menus can have some effect on performance. It has been shown that the amount of information in the menu frame and the number of opportunities to make "slips" has an effect on performance (see Miller, 1981 and Snowberry et. al, 1983). This balance, between amount of information and the opportunities to make mistakes, defines the parameters of the depth/breadth trade-off. Generally this work has stressed the importance for performance of the quantity of the information. Snowberry et al (1983) pointed to the fact that simple quantitative parameters are not the only factors to affect performance and that "categorisation" within menu frames can aid performance. This increase in accuracy of performance is probably caused by the process of mutual disambiguation. While it has been suggested that categorisation improves performance, problems of categorisation, organisation and description have not received much attention.

However, Young and Hull (1982) suggested that a crucial feature of many existing menu systems is "cognitive compatibility" or 'the extent to which the system repects the users prior knowledge, habits and cognitive limitations ... in particular, certain difficulties are to be explained by a "cognitive mismatch" between the designers decisions embodied in the frame and the expectations of the user' (Young and Hull 1982, p. 571-2).

Most of the studies mentioned above involved simple, well defined tasks (e.g. Whiteside et. al., 1985). With such small, simple domains and with a restricted set of clearly defined possible goals, it is unlikely that the subject would not understand the domain. However, in larger, more complex domains, and with users who have many different purposes and levels of expertise (such as PRESTEL or POPLOG on-line documentation), the problems of "cognitive conflict" become increasingly important. Below, through illustrations from the POPLOG environment and by elaborating on Young and Hull's work, two major forms of conflict are discussed; conflicts between *purposes* and conflicts between levels of *experience*.

Users with Different Purposes

A general problem in many large complex menu systems is the difficulty of organising and presenting information (or a type or set of commands) which might be used for many different purposes. Consider some of the possible descriptions of a spade; a spade, a shovel, a digging implement, a chopping implement, a carrying implement with limited capacity, a prop, a wooden handled implement with a sharp, broad, metal edge. Each description is valid (though not limited to a spade) and only some descriptions might be relevant to deciding on its appropriateness for different tasks. That is, it may not be enough to call a spade a spade, it will depend on what task one has in mind and what the user's "background" knowledge is. For instance, the distinction between a spade and a shovel may not be important for some users. However, some may need a tool embodying the unique properties of a shovel (curled edges, large capacity) and, because they have a specific goal and a refined notion of the differences between spades and shovels, they may not have expected to have found its description associated with spade. Alternatively a user may not recognise that there is a difference between a spade and a shovel, or she may not know the name "shovel", though she does know the name "spade" — if the

description of the shovel is indexed under that of the spade she may never find it.

As shown in Section 3, like shovels, software constructs and facilities can be put to many uses — indeed probably more. A major problem is how to organise and present information in a manner so that different users with different tasks will recognise its relevance to their problem. Presentation (the descriptions) and organisation (the overall structure of information) are clearly related in that the form of description (its level of generality etc.) will determine the menu structure or organisation.

As mentioned in Section 3, in the provision of on-line information, a distinction could be made between organising information according to the constructs and facilities provided (system oriented) or according to the possible applications to which they might be put (task oriented). For example a POPLOG user may simply want information in order to find out about editing a file. In the (caricatured) system orientation this might be found under the description *'copying data from the terminal device to the file device'*. In the task orientation this might be found under the description *'editing a file in VED'*. The system vs task distinction should not be confused with structural vs functional descriptions: e.g. *'a wooden handled object with a sharp, broad, metal edge'* vs *'a carrying implement with a limited capacity'*. Some software constructs and facilities can have system descriptions which are either structural and/or functional.

There are problems with both sorts of approaches. Even in the relatively simple case of organising POPLOG on-line documentation (compared, say, to the librarian's problem of wanting to find an organisation for all human knowledge) we are not sure what a system oriented view would be, or even whether there is a single system view. For example, at the level of a particular programming language there is one system view. But from the viewpoint of how that language has been implemented — i.e. what the underlying representation is — there is another view. The latter view may be important for users concerned with efficiency or obscure errors. Also as mentioned, like human knowledge, software constructs are being continuously developed and so occasionally change their relationship to each other — hence we may have to accomodate a system view that changes over time. Another problem with the system view is whether users would be successfully and conveniently able to use such a description of information to know whether it is relevant to their problem (consider the example given above of editing a file).

However, in developing a task-oriented organisation, anticipating the ways a facility may be used is often very difficult, and in some cases a facility will be useable in many ways (infinitely many in the case of programming languages). Clearly then in many cases there would be a problem with the amount of space taken up by the possible set of all task oriented descriptions (assuming we could specify them all). Consequently, this would cause problems for navigation and ease of browsing.

Learnability, Ease of Use and Levels of Experience

Related to the problem of users having different purposes is their level of experience. It may be that in making a interface easy to learn for beginners (such as a menu interface) it is made awkward for experts. It is likely that there will be many different users with a range of experience with the system, and therefore unlikely that any one interface would satisfy all users all the time. Whiteside et. al. (1985) suggested there is no trade off between ease of use and learnability with menus. Interfaces (including menus) that novices found easy to learn were also found to be easy to use by experts and novices. However, the tasks presented to the

users involved "simple operations on text files, such as displaying, merging, and sending to another user" (Whiteside et. al., 1985 p. 186). Menu systems are often designed to shield the user from the intricacies of the intermediate steps of a command (hence making them easier to use and learn). This may also prevent users from understanding and using the intermediate steps that may, in some systems, for some users be important in providing the extra "generative power" those intermediate steps could produce in different combinations. Providing the novice "tourist" of the interface with a phrase book may help her on initial "day trips" into the system but may not be the best way of helping her learn the grammar of the "language" so that she can generate phrases of her own. That is, it may prevent novices from becoming experts. It may also prevent expert users from utilising the commands in a more flexible manner.

Information can be presented to users in a variety of ways and at different levels of complexity. There may be a need to distinguish between information produced specifically to help users learn about some new concepts or facilities and information provided to generally help users use these concepts or facilities (as with the TEACH/HELP distinction in POPLOG). Systems which only provide one level of information may be easily learnt and, for some tasks, found easy to use by all users, but experts may require significantly different types of information than novices and may not find them adequate for all purposes. Adelson, (1984) has found that experts and novices have different program recall abilities; experts tend to be able to recall the general or abstract features of the program while novices tend to remember far more fragmented and concrete features of the program. So, for instance, experts may not find it so difficult (indeed they may prefer) to infer the relevance of systems oriented decriptions to their task, whereas novices may find this process much harder. It seems therefore that the problem of "learnability" vs "ease of use" will be serious for simple menu based systems where the range of complexity of activities is extensive.

5. META-HELP Browsers and Support Mechanisms

This section describes the development of the META-HELP interface to the POPLOG on-line documentation. This is a menu based interface embodying their advantages over command based versions but overcoming some of their problems.

Our aim was to provide a conceptual structure representing the contents of the documentation. The intention was that users should use a menu based version of this in order to identify and invoke the information they required. This was done in the context of the new These give meta-level information about the structure and contents of the help files in order to help users browse "over the top of the documentation" — dropping down into it when they identify the material they require. The META-Help files and the access mechanism are located in the same interface as is used for most other activities, namely the POPLOG editor — VED. This means that users can switch freely between various modes or tasks such as sending mail, reading HELP file, running a program, copying sections of a documentation file into a private file etc.

Each META-HELP (M-H) file has a title, parent/s (a reference to another M-H file) and children (a reference to either other M-H files or documentation files). Each reference has an associated description of what information it is concerned with. This mechanism will support a heterarchical organisation which can be viewed as a collection of interlocking tree-structured hierarchies. However, we are not limited to producing tree structures as each reference can be to any node at any position in the network. References can also be to sections within documentation files. This means that when these references are invoked the user is not forced to work

through large amounts of information irrelevant to their task.

As a menu based interface it has all their advantages, that is, it capitalises on the user's more effective cognitive processes (recognition as opposed to recall). Therefore there is no need for a user to know the name of a file in order to begin a search for it. There is also no need for a user to type the name of a file in order to invoke it. Two simple key sequences are provided — one to place the cursor on the next cross reference in the M-H file and another to invoke the reference under the cursor. The categorisation of files within these M-H files facilitates the process of mutual disambiguation and the simple procedures required to use it mean it is easy to use and easy to learn to use. Finally, it is implementable and compared to other more intelligent systems it is computationally inexpensive. This mechanism will also, in time, be able to be made more intelligent and more responsive to users* needs. Some of the ways it can and has been extended are discussed below. [3]

5.1. Additional Mechanisms

5.I.I. Navigational Information and Speed of Travel

There are at least two (not necessarily mutually exclusive) types of navigational information that users may need. This may require the design of additional mechanisms to be used in conjunction with the menu based system.

Static map : a picture of the system

This might simply be a "map" of the network displayed graphically or in some other form. It would represent all the nodes and their relative position to each other. However, additional information could be provided about the nature of the information; M-H or leaf node (the item which was searched for via the M-Hs), introductory information, tutorials, examples etc.

Dynamic map : the user's path history

This would be a map of the path that the user has taken through the network. This could be provided separately (e.g. an unstructured list of items viewed), or it could be used in relation to the static map of the network. A simple form of this dynamic map could be developed so that if at the beginning of a session a user "jumps" to a certain node in the tree, only the map below this point is invoked (this would only be appropriate in hierarchies, rather than networks). A more elaborate map would not only remind users what node they had visited, but where it was in relation to other nodes, showing them how to get back to that point. This contextual information could be further elaborated to highlight, for instance, what was not selected at each particular node — i.e. what remains to be viewed, as well as what has already been viewed). This would obviously be useful in view-data systems, but also in menu based command interfaces where a set of commands frequently used could be highlighted (through frequency of use) on the map, and hence more easily located and invoked, though there would be no additional constraints on finding new information. A prototype dynamic map has now been built for POPLOG.

There are two main constraints on providing this information. First, there are *technical constraints:* In very large systems it is likely that both forms of maps would be very large and complex. The screen size of terminals and the availability of high resolution graphics would probably be crucial factors. The colours, textures and shapes provided by high resolution graphics would probably be needed to present information about the items in the map, such as the status of the nodes:

whether the are active or inactive (M-Hs or leaf nodes viewed or unviewed), or the contents or category of the nodes: M-H or leaf, tutorial, introduction, examples etc.

Second, there are *user constraints:* Even if the technical constraints were overcome, it is unlikely that all this information should always be on the screen. Much of the information will be redundant at times and may produce unnecessary cost in terms of computational power or screen space. It may also be too much information for the user to attend to at once, so that she may enter into a situation where she needs a "meta" browser in order to sucessfully use the navigational aid. Another user constraint derives not from the user's general attentive abilities but from her range of experience.

In order to limit the information available on the screen and to aid selective attention, certain criteria could be given for the presentation of information. Rather like traditional geographical maps, the presentation of information could be given in different "scales", where "scale" simply constrains the amount of information provided. The "scale" could be adapted to the user's needs. A simple way of providing "scale" related to the user's immediate needs would be that, for recently invoked or frequently used M-Hs or leaves, more information would be provided about them and their relation to surrounding nodes than others. More advanced maps might utilize "fisheye" techniques (e.g., see Furnas, 1986).

The map is not be just navigational tool but also a means of travel within the network. The user can choose between selecting an item from the current M-H (and hence placing it in the dynamic map) or choosing an item from the M-H map and treating this as an item to be selected. This makes travelling around the networks much faster. The user need not return to a point via the intermediate nodes indicated on the map, she need only select the node at which she wishes to arrive.

5.1.2. Keyword Searches Linked with M-H

Because of the number of documentation files, their diverse subject matter and the lack of rigour in their structure, it is unlikely that a keyword mechanism for directly accessing documentation files could be built or would be useful. The number of keywords would be too many for the designers to adequately collate and for the users to remember. However, a more limited subset of keywords could be produced for the M-H system. When a user invokes M-H she may include several keywords with the command. These keywords could then be used by the system to infer which are the most relevant area/s, these could be highlighted on the map. She could then chose to "jump" to one of these nodes directly, once she is at this point she can perform a more refined search herself. This will allow users to travel much faster within the network. It may also help in overcoming some of the problems of users not recognising the relevance of some menu descriptions to their tasks in that much of the work in inferring the relevance of material would be done by the system's comparisons of keywords, leaving the user to make a more refined search at the end only.

In the future a synonym search mechanism could be usefully integrated with information about the user to make the search more responsive to her past experience or immediate goals. For instance in the discussion about the description of spades it was shown that certain "users" (e.g. a "novice" digger) may not realise that there is a distinction between spades and shovels. If "spade" is given as a keyword then the system might return information relevant to both the use of spades and shovels (perhaps including tutorial material on the distinction between them). However, if it is an expert user (in this domain) then only information relevant to spades should be given. Far more intelligent systems might also examine the task

that the user is trying to perform and point out that a shovel would be far more appropriate.

5.13- The M-H Database

Whenever a M-H file is installed a database is updated. This contains a description of the M-H files name, its parent/s and children. Using this database a user can (with a single keystroke) "jump up" from a particular documentation file to the appropriate place in the M-H system (i.e. the place where it is referenced). If there is more than one reference then a menu of options is presented. This allows a user to go from the specific (a known file) to the general (a M-H file giving references to related subjects they may wish to know about).

This database also allows us to keep the documentation more up to date. Whenever documentation files are altered the author can be informed about its reference in the M-H system and question about how the M-H description needs to be changed. With an extended keyword mechanism, when new files are added (perhaps tagged with some keywords) it may be possible to advise the author where in the M-H system the new file should be referenced. This advice would be based on a comparison of the keywords in the existing M-H structure and the keywords in the new file.

5.1.4. Responsiveness to Users' Needs

While menus have many advantages over command based systems a major problem with them (and many command based systems — but not the "ideal interface") is that they are not dynamically responsive to users* individual needs or levels of experience (though neither are most command based systems). However, it is hoped that the M-H system will be extended to be more responsive.

Use of Maps

A simple version of the adaption of M-H structure and content to the user's overall experience would be simply to provide alternative M-H systems; e.g. an "expert's" M-H system and a "novice's" M-H system. However, as mentioned earlier, users may find simple blanket classifications of expertise overly restrictive. As Draper (1985) has shown, such notions of expertise are unlikely to be useful. In an analysis of expertise in UNIX, one of his findings was that those users commonly classed as experts do not have expertise that encompasses all domains. Instead, "experts" tend to be fluent in certain areas, and relatively inexperienced in other domains. However, the structure of the M-H system allows us, in principle, to design a simple system that represented the user's experience in a more flexible manner. A record could be kept of what and how many times a user had visited different areas of the M-H system. This record could then be used to change the structure or the content of the M-H system in accord with their experience. This begins to overcome the problems of "learnability vs ease of use".

More intelligent systems would be able to identify patterns in the paths taken by users (during one session or over a period of sessions) and use this either to not only constrain what is on the screen but also to suggest new routes or items not identified or known by the user. For instance, if a user is searching an area of the M-Hs which has not been viewed before, introductory texts could be included. Alternatively, if she has viewed this area many times it is likely that she will not need introductory texts, and hence they could be excluded — though some indication of the exclusion should probably be given so that the user could invoke them if necessary.

6. Conclusion

Although menu interfaces are now in common use, and there seem to be many advantages in their use, empirical research has not provided am adequate clarification of their benefits and has had little to contribute in terms of design implications. In certain systems many different kinds of problem arise with menu use. These systems are characterised by large amounts and complexity of information, diversity of users and of possible uses of the information. Many of the problems that arise concern navigation through the menu structure and the lack of responsiveness to users' needs.

The META-HELP browser is a menu based interface to on-line documentation in a large software development system. As it is menu based it has many advantages over the existing command based interface. However, the additional support mechanisms overcome some of the disadvantages of menu based systems applied in this domain.

Notes

[1] Much of this paper is the result of the research involved in the Alvey project "Towards an Intelligent Help-File Finder" carried out at the Cognitive Studies Programme at Sussex University. Aaron Sloman, Tom Khabaza, Andrew Law and David Allport all participated in this research. Some of this paper, namely parts of Section 3.1.1. and Section 5. have been taken from Sloman, Allport, Khabaza & Law (1986), Sloman (1985) and Sloman (1986), and some examples have been taken from Khabaza (1985b). However, the mistakes are entirely the author's responsibility.

[2] Many of those involved in the development of POPLOG and in teaching and research in the Cognitive Studies Programme at Sussex contributed to this analysis. This group includes Aaron Sloman, Tom Khabaza, Jon Cunningham, Claire O'Malley, Josie Taylor, Steve Draper and David Allport.

[3] See Appendix B and C for examples of M-H files.

References

Adelson, B., (1984). When novices surpass experts: the difficulty of a task may increase with expertise. *J. Exp. Psychology, Learning, Memory and Cognition, Vol 10, No.5*, 483-495.

Allport, D., (1984). A parser for helpfile English. MSc thesis, University of Sussex, 1984.

Barrett, R, Ramsay, A. & Sloman, A. (1985). *POP-11: A Practical Language For AI.* Ellis Horwood and Wiley: Chichester.

Draper, S. W. (1985). The nature of expertise in UNIX. In B. Shackel, (Ed.) *Human Computer Interaction — Interact '84.* 465-471. Amsterdam: North Holland.

Fisher, G., Lemke, A. & Scwab, T. (1985). Knowledge based help systems. In L. Borman and B. Curtis (Eds.) *Human Factors in Computing Systems: CHI '85*

Conference Proceedings. ACM: New York.

Furnas, G. W. (1986). Generalised fisheye views. In M. Mantei & P. Oberton, (Eds.) *Human Factors in Computing Systems: CHI '86 Conference Proceedings*. ACM: New York.

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

Hardy, S., (1984). A new software environment for list-processing and logic programming. In T. O'Shea and M. Eisenstadt, (Eds.) *Artificial Intelligence: Tools Techniques Applications*. Harper and Row: London.

Khabaza, T., (1985a). Towards an intelligent help file finder. To appear in R. Hawley, (Ed.) *Artificial Intelligence Programming Environments*. Ellis-Horwood: Chichester.

Khabaza, T., (1985b). What is an Intelligent Help-File finder. An unpublished internal paper, produced in the Cognitive Studies Program Sussex University.

Kidd, A.L. (1985). What do users ask? — some thoughts on diagnostic advice. In M. Merry (Ed.) *Expert Systems '85: Proceedings of the Fifth Technical Conference of the British Computer Society Specialist Group on Expert Systems*. C.U.P: Cambridge.

Miller, D.P. (1981). The depth-breadth trade off in hierarchical computer menus. In *Proceedings of the Human Factors Society 25th Annual Meeting* 296-300.

Norman, D.A. (1983). Design principles for human-computer interfaces. In A. Janda (Ed.) *Human Factors in Computing Systems: CHI '83 Conference Proceedings*. ACM: New York.

Perlman, G. (1985). Making the right choices with menus. In B. Shackel (ed.) *Human Computer Interaction — INTERACT '84* Elsevier Science Publishers.

Tullis, T.S. (1985). Designing a menu — based interface to an operating system. In L. Borman and B. Curtis (Eds.) *Human Factors in Computing Systems: CHI '85 Conference Proceedings*. ACM: New York.

Robertson, D. McCraken and Newell, A. (1981). The ZOG approach to man — machine communications. *International Journal of Man-Machine Studies, 14*, 461-488.

Sloman, A. (1985). Case for support: Towards an intelligent help file finder. Unpublished internal discussion paper. Cognitive Studies Program, Sussex University.

Sloman, A. (1986). Towards a model of a good software designer. Unpublished internal discussion paper. Cognitive Studies Program, Sussex University.

Sloman, A., Allport, D., Khabaza, T. & Law, A. (1986) Towards and Intelligent Help File Finder - Interim report to the Alvey Knowledge Based Systems Club.

Snowberry, K. Parkinson, R. & Sisson, N. (1983). Computer display menus. *Ergonomics, 26,* 699-712.

Young, .M. & Hull, A. (1982). Cognitive aspects of the selection of viewdata options by casual users. In *Proceedings of the 6th International Conference o Computer Communication.* 571-576.

Whiteside, J., Jones, S., Levy, P.S. & Wixon, D. (1985). User performance with command, menu and iconic interfaces. *Human Factors in Computing Systems: CHI '85 Conference Proceedings.* 185-191.

Appendix A : An example of a short POPLOG HELP file

```
HELP * FOREACH

            foreach <pattern> do <actions> endforeach;
        foreach <pattern> in <database> do <actions> endforeach;

FOREACH is used for iteration over a database. The
<pattern> is matched (see HELP *MATCHES) against each
element of the database, which defaults to the value of the
variable *DATABASE when the IN clause is omitted. After
each successful match, the <actions> are performed. Usually,
the match will have side-effected the value of some variables.

See also HELP
            * FOR          - Iteration over lists and integers
            * FOREVERY     - like FOREACH, but matches a list of patterns
            * DATABASE     - for the use of the POP-11 DATABASE
            * LOOPS        - for other types of iteration
```

Appendix B : An example of a POPLOG META-HELP file

META-HELP * LIST_INITIAL

0-Parent:	META-HELP * INDEX
1- Introduction to lists	(doesn't exist yet)
2- Overviews of list processing	META-HELP * LIST_OVER
3- List Syntax and creating lists	META-HELP * LIST_SYNTAX
4- Pattern matching with lists	META-HELP * LIST_MATCHING
5- Accessing parts of lists (e.g. getting the head)	META-HELP * LIST_ACCESS
6- Examining the characteristics of lists (e.g., membership, length, etc.)	META-HELP * LIST_EXAMINE
7- Processing the contents of a list (e.g., sorting, deleting elements)	META-HELP * LIST_PROC
8- Iteration over lists (a means of performing some action to or for each element of a list)	META-HELP * LIST_ITERATION
9- Pair mainpulation	META-HELP * LIST_PAIR
A- Input/Output	META-HELP * LIST_IO
B- Dynamic lists	META-HELP * LIST_DYNAMIC
C- Copying lists	META-HELP * LIST_COPY

Appendix C : An example of a POPLOG META-HELP file

```
META-HELP * LIST.JTERATION
0- Parent:                                  META_HELP * LIST_INITIAL

General introduction to iteration
1- Using a "for" loop                       HELP * FOR

Iteration over the elements in a list. The following
facilities can be used to peform some action to or for
each element of a list:

2- Using a procedure for the action         HELP * APPLIST
3- Using a procedure for the action         HELP * MAPLIST
but creating a new list from the
results
4-Iterating over those elements of a        HELP * FOREACH
list which match over a given
pattern
5- Iterating over COMBINATIONS              HELP * FOREVERY
of elements which match a
collection of patterns
```