The Application of Natural Language Pragmatics in Human-Computer Interaction

Charles Elliot
B.A.hons., M.Phil., M.Sc.

A thesis submitted in partial fulfilment of the requirements of
Sheffield Hallam University
for the degree of
Doctor of Philosophy

February 1993

Sheffield Hallam University in collaboration with British Telecom

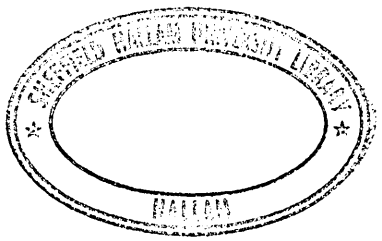


Table of Contents

Acknowledgements

I would like to thank my supervisors, Bob Steele and Rick Osborn, for their guidance and support throughout the completion of this work. I owe a debt of gratitude to my colleagues in the research office for their warmth and humour and I would also like to thank Nick Harper and Caroline Friend for their comments on earlier drafts of this text, Mel McClellan, Duncan Kime, Jan and Steve Peel, Tony Frascina, Sue York and others who took part in the evaluation exercises. Warmest thanks must be reserved for Niki, my wife, who has had to endure me while this thesis was completed.

Abstract

The general aim of the work reported in this thesis is to investigate the viability of applying theories and principles from the field of natural language pragmatics to that of human-computer interaction. In pursuing this aim, the research falls broadly into three phases.

The first of these is the exploitation and adaptation of the Gricean Cooperative Principle, its maxims and inferential rules to situations of computer use which do not employ natural language as the medium of communication. The purpose of this endeavour is to provide a novel and revealing analysis of non natural language interaction and to establish principles for dialogue design, the application of which enhance the quality of communication between system and user in such situations.

The second phase concerns the application of the adapted Gricean principles to the design of a dialogue management system, intended to address some of the problems which other research has revealed users to experience in using the standard UNIX® shell interface. This second phase resulted in the production of the QDOS system, which is both a simulation of part of the UNIX® file system and an implementation of the proposed dialogue management system.

This software acts as the vehicle for all subsequent evaluative exercises constituting the third phase. This takes the form of an evaluation of the QDOS system and its theoretical underpinning, based on a two-condition experiment and a protocol analysis, involving a number of experimental subjects.

This research provides an original application of the Gricean Cooperative Principle in human-computer interaction and a theoretical and practical demonstration of the validity of this endeavour. It also adduces an analysis of the UNIX® interface and its vagaries in terms of a principled and consistent set of criteria as well as identifying a significant class of dialogue breakdown, the circumstances and incidence of which cut across issues of interface style.

1. Introduction

1.0 Aims and Motivations

The general aim of the work described in this thesis is to investigate the relationship between natural language communication and that occurring in human-computer interaction (HCI). In particular, an attempt is made to demonstrate that certain aspects of natural language communication, namely the pragmatic factors in interpretation evident in descriptions of natural language communication, are applicable to the HCI context and that a comparative study provides insights into the processes involved in HCI, leading to conclusions of significance to issues relating to the usability of computer systems.

The aims of the project described here have been motivated by three basic factors. The first of these is the conviction that the study of natural language, especially natural language pragmatics, can provide valuable insights into some of the issues involved in the study of human-computer interaction. Although there has been a significant amount of research on natural language and its relation to human-computer interaction prior to that reported here, much of this work has concentrated either on lexical and syntactic factors or on the problems inherent in computerised natural language processing. One of the aims here has been to conduct some investigation into the application of pragmatic aspects of natural language theory to non-natural language forms of human-computer interaction.

A second motivational factor stems from behavioural studies of

computer usage which indicate similarities between certain situations which occur in human-computer interaction and human-human interaction, especially with regard to miscues, misconceptions and errors. It is also suggested that these cases, where the human-computer dialogue breaks down, provide a particularly interesting view of the communication process taking place between user and machine for the HCI practitioner, in that they often expose to scrutiny aspects of the process which would go unnoticed during 'normal', unbroken spells of dialogue. It is not safe to infer from an unbroken dialogue that all is well. Users may be being forced to compensate and compromise in all sorts of ways by an interface design which they find unnatural, awkward or inflexible. It is not that investigations of 'normal' usage are not designed to reveal such interface characteristics but it is suggested that such methods provide a less direct form of access to them.

The third motivational factor is admittedly a rather negative one but is just as important. In recent years, it has become increasingly common to see computer manufacturers and software companies converting their machines and applications to what is variously called the 'graphical', 'iconic', 'direct manipulation' or 'WIMP' user interface. Whether these terms denote the same thing is a non-trivial question and will not be addressed here but what is usually meant is an interface like that found on the Apple Macintosh computer. While the popularity of such a move is to some extent justified, there are a number of accompanying assumptions which are not at all tenable.

The worst of these, while admittedly anecdotal, is discernible, at least in the lay population and is that most of the main issues and problems in interface design have been solved or are solvable by the adoption of the graphical user interface (GUI). I know that I am far from alone in having struggled with a Macintosh Plus for months before many aspects of its usage revealed themselves.

Two slightly more understandable and related assumptions are that the GUI renders all human-computer interaction into a form similar to that of

the use of a tool, in the same way that someone might use an electric drill, and that (perhaps as a consequence) interaction with a computer is not in any way a communicative process like talking to another person or that all similarities between natural language use and computer use are absent in the GUI. Beside the fact that existing GUIs actually embody a number of non-graphical means of interaction to which the non-linguistic or 'tool' metaphor is inappropriate, computers differ from other devices in being non-dedicated or multi-purpose and are used now in a multitude of tasks, each with its own particular demands and requirements. It is unlikely, to say the least, that a single interface 'metaphor', such as that of the 'tool', is capable of providing suitable interaction facilities for all eventualities. Thus, one of the aims of this research has been to suggest how communicational issues concerning human-computer interaction cut across issues of interface style.

1.1 Methodology

The basic methodology adopted in the research has been to try to analyse some form of communicative breakdown between computer and user by applying the theoretical constructs of natural language pragmatics. The resulting analysis then forms the basis for recommendations or requirements for certain facets of interface design. Both the design and the analysis are subjected to evaluation; the design by experimental methods and the analysis by observational means.

The kind of communicative breakdown studied is that arising from user misconceptions concerning the context of interaction, in terms of system characteristics, caused by the inability of the system to signal those characteristics adequately to the user in any consistent or comprehensive way. We have termed these breakdowns 'context errors' and they are analysed using concepts and categories from linguistic theories of presupposition, speech acts and conversational implicature. The analysis leads to the design of an 'intelligent' interface component, a prototype, test bed implementation which is used in the evaluative exercises.

We hope to demonstrate that, in terms of pragmatics, there are many similarities (as well as some differences) between human-human and human-computer communication and that the exploitation of these similarities can significantly influence decisions concerning dialogue design in the human-computer interface. In particular, it will be shown that principles assumed to be guiding cooperative communication in contexts of natural language use may be adapted and applied in order to render the computer's participatory role in the human-computer dialogue similarly cooperative.

1.2 Structure

The thesis is divided into eight chapters, each dealing with a different aspect of the research. The literature survey is integrated into each chapter and its particular topic, rather than occupying its own section. Following this introductory chapter, Chapter 2 looks at natural language communication, its relation to human-computer interaction and some of the attempts that have been made to mould human-computer communication along the lines of that which occurs between humans through natural language. Some objections to this endeavour are considered and conclusions are drawn as to the circumstances and conditions which govern the applicability of the natural language paradigm to human-computer interaction. The concept of natural language pragmatics is introduced, in particular, the theory of conversational implicature as constituting the underlying principles of communicative cooperation and the application of these principles to human-computer communication is considered.

Chapter 3 is concerned with error situations in computer usage or what has been referred to above as situations of 'communication breakdown'. As was indicated earlier, it is suggested that an understanding of what is happening in such situations offers valuable insights into the general processes of human-computer communication. To this end, classifications of error are examined and a levelled linguistic classification is proposed. As part of this classification, the notion of context error is introduced and

analysed using the theoretical constructs discussed in Chapter 2. Conclusions are advanced for the implications that this analysis has for dialogue design.

Chapter 4 identifies the requirement of user modelling as a consequence of the need for the interface component to behave intelligently and cooperatively. Classifications of user models are compared and *desiderata* are presented for the user modelling enterprise. The notion of the user's Conceptual model is discussed and an informal approach to modelling user misconception concerning the device, based on a 'possible worlds' model of belief systems, is presented. The theoretical question of generating possible world models of user misconception is addressed.

Chapter 5 discusses issues of user support and the various methods employed to aid users in learning and utilising computer applications are considered. The importance of user support, as a factor of usability, is stressed and the relative effectiveness of various approaches to user support is estimated. The requirement for intelligence in the interface is identified and some examples of intelligent interface components are reviewed. The design of an intelligent interface component, based on the analysis of the previous two chapters is discussed.

The implementation of a test bed, prototype interface, as part of a simulation of the UNIX® file system is described in Chapter 6. This simulation is used for evaluation purposes in the project. The processing which the user support component undertakes on user errors is explained, as are any deviations from the design. Some explanation of the actual code found in the appendices is included to enable the reader to refer easily to this if required.

Chapter 7 deals with interface evaluation techniques and their relative merits. The importance of evaluation is identified and three basic types of evaluation are discussed. The evaluative methods and exercises conducted within the project are described and results presented. Conclusions and observations on these results serve as an appraisal of the success of the approach adopted and its application to the aspect of interaction studied.

Chapter 8 attempts to draw together all of the strands of the other chapters and to evaluate their strengths and weaknesses as they affect the initial aims of the research.

2. The Natural Language Paradigm

This chapter looks at natural language communication, its relation to human-computer interaction and some of the attempts that have been made to make human-computer communication like that which occurs between humans through natural language. Some objections to this endeavour are considered and conclusions are drawn as to the circumstances and conditions which govern the applicability of the natural language paradigm to human-computer interaction. The concept of natural language pragmatics is introduced, in particular, the theory of Conversational Implicature as constituting the underlying principles of communicative cooperation and the application of these principles to human-computer communication is considered.

2.0 Introduction

A programming language is a kind of human-computer interface, in that it is a medium via which programmers can 'communicate' their intentions to the computer, in the same sense that the controls in a car are the media which allow the driver to get the car to behave in the desired way. In an interactive programming environment, the use of a programming language will rely upon the existence of other programs (a compiler, for example) just as the users of a word processor achieve their goals via the operating procedures of the particular package. Both the programmer and the user are getting the computer to perform some task for them via a communicative medium. Programming is merely one kind of

human-computer interaction (HCI).

In the early days of computing, the medium of communication was restricted to numerically coded instructions. The difficulty of using this language and its error-prone nature led to the development of assembler mnemonics, these having the advantage of greater semantic significance for the human programmer. Assembly language is still in use, but has largely given way to high level languages whose formalisms, on the whole, are intended to make their functions more obvious and more powerful.

It is not difficult to detect in these changes the way in which the communicative media have become increasingly like natural language (NL). The use of control structures such as DO UNTIL, WHILE DO, BEGIN END, LET and so on are testimony to a tendency which is, perhaps, currently, taken farthest in the Hypertalk© programming language (Weiskamp & Shammus, 1988). However, high level languages are small, highly formalised and very restrictive when compared with natural languages and the question arises of whether this tendency could and should be taken to its logical conclusion by using natural language as a medium for HCI, both for programming and in more generalised contexts of computer use. The apparent attractiveness of this proposal lies in the fact that it is a medium of which the great majority of people already have an excellent grasp, whereas most other forms of interaction have to be painstakingly learned. Thus, should this goal be realised, anyone with an average grasp of a natural language could program or use a computer without training.

In reality, however, the prospect of making NL the primary medium for interacting with computers is clouded by innumerable problems and questions. In this chapter the relationship between NL and HCI is discussed and the various aspects of NL which relate to HCI and the extent to which they are appropriate to it are considered.

2.1 Against Natural Language Interaction

The adoption of NL as a paradigm for human-computer interaction has been opposed on various grounds. Hill (1983) argues against the adoption of NL as a programming language, mostly on the basis of its inherent ambiguity and imprecision and it must be admitted that little justification can be found for the introduction of such features into what is intended to be a precise activity. NL is also rather verbose compared to most programming languages and advantages of familiarity also have to be weighed against the loss of conciseness.

Green (1986) argues that interaction models using NL-like grammars are inferior to event driven models because they cannot be made to represent multi-threaded dialogues. Models of HCI need to be able to handle non-linear interactions because of the existence of multi-tasking systems which allow users to switch between tasks at any point, for example, by activating a window associated with a different process. A similar point is made by Buxton (1987) who states that the development of novel interaction media is stifled by the adherence to characterisations of HCI based on NL, rather than on task domains.

Other critics attack from a more philosophical point of view. Flores and Winograd (1986) pitch their critique at the prospects for natural language understanding and claim that the basic aim is misguided, in that human communicative abilities are unique to the species and that machines cannot, in principle, be imbued with such powers. Their arguments are based on a rejection of what they call the *rationalist tradition* which, they insist, treats language as a formal system; a view which they eschew in favour of an ethnomethodologically-based alternative which sees language as a product of the essentially social nature of humankind and our position in the evolutionary context.

As far as Hill's remarks go, it should be pointed out that programming is merely one form of HCI. The presence of ambiguity and imprecision could possibly be a boon in the context of an expert system by allowing users' input to be interpreted in ways that are not perhaps intended or

foreseen. Also, the remarks only pertain to full NL processing, while it will be argued below that this all or nothing approach is unnecessary in the broader HCI context (Harris, 1983: p. 73). Besides, as Harris points out, in many HCI contexts, the semantic domain is restricted enough to rule out ambiguity.

The point made by Green is aimed at the representation of HCI using formal grammars and not specifically at NL interaction but the point can be taken to apply to any grammatically based NL interface. It should not be thought of as a general criticism, however. NL does not consist entirely (if at all!) of phrase structure rules and if one actually looks at NL usage, copious examples of interleaving dialogues and parallelism can be found. Non-segmental features such as intonation and stress can also be seen as parallel transmission channels and it has also been argued that phonemic assimilation is evidence for the non-seriality of NL (Edmondson, 1989).

Although there is much to dispute in Flores and Winograd's book (for example, the claim that literal meaning is not primary prompts the question of how language is acquired if words have no systematic effect over a range of contexts), it is not the purpose here to argue over the merits of the aim to have computers understand natural language. The technical problems involved in such an endeavour are daunting enough in themselves (Grace, 1987). For the purposes of this discussion, it will merely be reiterated that their point of view should not act as a deterrent to those who study NL phenomena in the pursuit of more usable computer systems, since it is not necessary to strive for systems which exhibit full NL understanding in order to exploit such sources of insight and understanding which may avail themselves. However, if this study is to prove fruitful, it is necessary to look more closely at task domains in HCI and the way they influence the user's representations, as well as the structure and content of the dialogue.

2.2 Natural Language Applications

To start, some important distinctions must be made between the various types of application to which NL has been or might be put. (We will ignore the question of speech recognition and synthesis, since the success or failure of those ventures affects only the convenience with which NL input and output could be achieved. While this may influence decisions concerning the suitability of NL as a HCI medium, it is not at the heart of the issue being discussed here).

Firstly, there are those systems which are designed to 'understand' NL by mapping NL input onto some real world model. Winograd's SHRDLU (Winograd, 1972) embodies such an attempt. This kind of application is intended to mimic the human ability to provide a semantic interpretation to NL utterances or inscriptions. This is undoubtedly the most ambitious kind of application and such success as has been enjoyed relies to a large degree on tight restrictions on allowable syntactic and semantic domain. One ambition level down from this is the NL database query system, where the mapping occurs between NL and some formal database query language such as SQL (which itself embodies a certain amount of NL mimicry). LUNAR (Woods, 1972) is an example of this approach. Natural language is also implicated in the use of grammars and equivalent formalisms to model interactive processes in task analysis, in that these are intended to describe a user's knowledge in terms of an *interaction language* (Green *et al,* 1988). Such models trade on the notion of a user's putative linguistic competence (Chomsky, 1965) and when applied to dialogue design, can be seen as imposing an NL like structure on interactive processes. At the simplest level, use may be made of NL lexicon and syntax to provide a degree of mapping between command function and NL semantics. Thus, the influence of NL on HCI may be viewed as operating on a continuum from the use of simple and fixed constructions and small domains towards full NL functionality (Harris, 1983).

An alternative view is that NL may be pertinent to particular computer applications. Computers are used in a wide variety of ways and it is equally

important to consider types of usage if the relevance of NL to HCI is to be correctly assessed. The relative success of NL database query systems owes much to the inherent similarities between such interactions and a particular and restricted kind of everyday human communication (i.e. question and answer), while the use of a computer aided design (CAD) package, for example, bears little resemblance, at least on the surface, to anything remotely like human discourse. The futility of designing an NL interface to CAD software is left to the reader's imagination. The point being made here resonates with Kammersgaard's distinction between the 'dialogue partner' and 'tool' perspectives of HCI (Kammersgaard, 1990). It is unrealistic to expect human-computer interaction to be best facilitated in all cases by the detailed mimicking of human-human communication.

The issue is also taken up by Buxton (1987), who stresses that forms of interaction other than NL are just as 'natural' and that a true 'natural language' system is only achieved when the language employed is suited to the task. However, decisions concerning the most suitable language for a particular task domain must be made largely on an empirical rather than *a priori* basis. It may be possible to rely on some metaphor taken from an existing manual system but such metaphors are not always available and in any case, this approach may be limiting, in that the potential benefits of task redesign in the computer context are lost. If Buxton's remarks are intended to encourage research into alternative interaction 'languages', then they are well taken. However, his sword is double edged, in that we must take his exhortations to include the investigation of all language types, including NL, and their relationship to task domains.

The point is, of course, not restricted to NL interfaces. The facilities that the interface provides for the user to manipulate system functionality are not perfected by merely adopting a particular style of interaction. The object manipulation metaphor or direct manipulation style does not facilitate all round perfect human-machine communication. For example, in the Macintosh© interface, to copy a file from one disk to another requires the dragging of the file icon onto the disk icon but to print a file

requires the use of a menu and dialogue boxes. This ambiguity is not accounted for by the failure of the designers to fully implement the object metaphor but is rather the failure of the metaphor to cover the full range of communicational phenomena inherent in kinds of interaction people have with computers and the tasks that are thereby undertaken (Sanford & Roach, 1988; p. 568).

The argument may be further elaborated by pointing out that it is not necessary to treat NL as an indivisible whole. Certain facets may be widely applicable in the context of HCI or applicable to certain task domains. The point again is that the application of NL to HCI is not necessarily an all or nothing proposition. The rationale behind NL interfaces is the familiarity of the interaction medium for the user. The issue of whether familiarity of particular NL characteristics can carry over to HCI or particular task domains within it effectively is one which should be addressed separately. As has been indicated above, advances have already been made in system usability merely by the exploitation of simple lexical and syntactic verisimilitude. Simply replacing obscure command language constructions such as R/TOOTH/,/TRUTH/ with the English equivalent: REPLACE 'TOOTH' with 'TRUTH', can improve users' performance with a word processor (Ledgard *et al,* 1980), although it must be said that this effect is not as straightforward as it might seem (Landauer, 1990; p. 145).

In summary then, the application of NL to HCI is seen by some as a means of rendering the latter easier by making it more like human communication. The extent to which the two are related can be viewed as operating on a continuum from the simple use of lexical and syntactic mappings from system functionality to NL semantics to full natural language understanding. However, the nature of the task domain and the consequent effects on the user's approach and representations, as well as dialogue structure and content, must be taken into account. In addition, the NL medium should not be seen as indivisible and NL features must be scrutinised individually for their appropriateness to the general HCI context and to particular task domains or features.

2.3 Natural Language Pragmatics

It is not at all easy to give a precise and comprehensive definition of the term 'pragmatics' (Levinson, 1983) and no attempt will be made to provide one here. The term is used to refer to a number of phenomena, associated with natural language and communication in general, which share certain features but which also possess contrasting ones. It is the attempt to capture both the similarities and discrepancies under a single definition and the fact that such definitions tend to stray across the boundaries of other disciplines which gives rise to this difficulty. For example, deixis is a phenomenon which has certainly been studied under the rubric of pragmatics but is also central to truth-conditional semantics in that sentences containing deictic expressions cannot be assigned truth-conditions without knowledge of the referents involved in a particular context of utterance. 'I am the first man in the world to run a sub four minute mile' can only be truly uttered by one person and it is necessary to know who uttered this sentence to decide its truth value. Thus the distinction between semantics and pragmatics can be blurred.

Also, certain kinds of pragmatic phenomena seem to be attached to particular lexical items. The conveying of contrast between the two conjuncts in the sentence 'I'm going out but I won't be long' is not captured in a truth-conditional account of meaning and so is ostensibly within the realm of pragmatics. However, pragmatic phenomena are by no means restricted to the effects of particular words and linguistic and non-linguistic context can determine the significance of a sentence by more than the reference of indexical expressions. Consider the following exchange:

A: 'Would you like a drink?'

B: 'Is the Pope Catholic?'

The propositions embodied in the two sentences are completely unrelated and yet A can reasonably be expected to infer a positive answer from B, the reasoning being something like 'the proposition expressed in your question is as likely to be true as this question - "Is the Pope

Catholic?'". It is A's 'world knowledge' and belief that B is not being deliberately obtuse which B is relying for A to receive B's intended meaning.

While there is a strong sense that the kind of phenomena under discussion at least have a common relation to the notion of the interpretative process which operates on the raw material of semantic content, there seems to be no unifying principle which enables them to be treated homogeneously and perhaps, attempts to do so stifle rather than facilitate progress. For our purposes it is only necessary to note the kind of interpretations which take place and make use of some of the explanations which have been put forward for them, without having to justify their pragmatic credentials (whatever this turns out to mean) or arguing which theory best explains particular phenomena. For this reason, no attempt is made here to explain or justify the various theories and their place in pragmatics in general. For the most part, the theory of Conversational Implicature (Grice, 1975) will be alluded to in explanation of such pragmatic phenomena as reveal themselves, while it is acknowledged that the theories of speech acts, presupposition or discourse analysis may have as much to say about them.

Notwithstanding the disputes concerning the appropriateness of NL models of HCI, one point about which there seems to be little disagreement is that, given the dialogue partner perspective, the computer side of the communication ought to exhibit a degree of co-operation comparable to that which could be expected from a human being (Pinsky, 1983). This kind of co-operation is nicely captured in Grice's cooperative maxims (*ibid*) which, because they are based on an idea of rational and maximally efficient co-operative communication, are purported to apply to communication *per se* and not just to NL. If the intention then is to render the computer side of the human-computer dialogue cooperative in this way, then it seems worth investigating the extent to which Gricean, or for that matter any, principles of cooperative communication apply in this context.

2.4 The Cooperative Principle

The Cooperative Principle is intended to embody rational considerations as guidelines for the effective and efficient use of language in conversation to further cooperative ends (Levinson 1983: p.101). It is expressed as follows:

The cooperative principle:

> *Make your contribution such as is required, at the stage at which it occurs, by the accepted purpose or direction of the talk exchange in which you are engaged.*

The generalised cooperative principle is elaborated by reference to a set of maxims which a cooperative dialogue partner is expected and assumed to follow or exploit.

These are:

The maxim of Quality:

> *Try to make your contribution one that is true, do not say what you believe to be false and do not say that for which you lack evidence.*

The maxim of Quantity:

> *Make your contribution as (but not more) informative as is required for the current purposes of the exchange.*

The maxim of relevance:

> *Make your contribution relevant.*

The maxim of Manner:

> *Avoid obscurity and ambiguity, be brief and orderly.*

The point of these guidelines is not that people follow them to the letter but that they form a background of assumptions to which most kinds of ordinary conversation are orientated. When contributions do not literally conform to their specification, it is assumed that the maxims are being observed at a deeper level. There are really two aspects to the operation of these principles. On the one hand, they are used as background assumptions in the interpretation of another's speech acts. Thus, when we are told for

example that someone has six children or received a fine in court, we naturally infer an implicit 'only' and would consider such information misleading if we were to find out that the said person had in fact ten children or had also been imprisoned for eighteen months, although what we were actually told was true. On the other hand, the principles also guide us in what to say. They are guidelines for both encoding and decoding extra-literal content.

The idea also has some possible design implications in that it might be possible to adapt conversational maxims to non-NL HCI contexts for the purpose of producing guidelines for the dialogue. In conversation, people are naturally disposed, both in speaking and in listening, to use conversational maxims such as be brief, relevant, orderly and so on. If people were also inclined to expect the same 'rules' to apply in the HCI context, then an interface which paid no heed to them would be inherently less usable than one which was in-tune with them. In reference to human-computer dialogues, conversational maxims probably constitute the most pertinent and accurate interpretation available of that much abused phrase 'user-friendly'. It should not be thought that this train of thought is in any sense original. Human-like cooperativeness is an oft expressed *desideratum* in HCI. It is not that the goal has not been stated often enough but that there seems to have been precious little concrete progress in achieving it.

Quite understandably, these ideas have been investigated in the context of NL interfaces. For example, Allen (1983) uses utterance or utterance fragments to infer users' travel plans. Users' plans will inevitably involve such contextual features as times of train departures, platform numbers and so on. If there is a contextual obstacle to a plan, a feature which might block its execution such as a train cancellation or a change of platform, then this fact is supplied to the user.

Kaplan's (1983) NL database query system is essentially intended to prevent the user from being 'stonewalled'. A user might formulate a number of queries on some particular topic, such as the number of students

passing a certain course in a certain year, the number with a grade higher than 50 and the number of referrals. If it happened that the course had not been run in that year, an uncooperative system might just give the answer '0'. But by looking at the implications (they might actually be classified as presuppositions, implicatures or perhaps felicity conditions) of the users' queries the behaviour of the system can be made more consistent with what might be expected from a human interlocutor.

In both of these examples, the general idea is that users' NL input is interpreted as implying certain contextual features which are compared against the world as seen by the system. Any discrepancies must be negotiated for or with the user. In the case of NL interfaces, the procedure for interpretation can exactly follow the linguistic paradigm.

The procedure or 'rules' for deriving such inferences or Gricean conversational implicatures runs as follows:

1) S has said that P.
2) I assume that S is cooperating.
3) For 1 and 2 to be true S must think that Q.
4) S knows that 3 is mutual knowledge.
5) S has not indicated that Q is false.
6) Therefore S intends me to think that Q.

How well does this process fit in non-NL interaction? The real problem is with 4. For 4 to be true S must know 3. For S to know 3 S must know the meaning of P (in order to determine Q) but as a possibly novice user, S may not know the meaning of P. The user may not be competent in the medium of the application and this makes the implicature possibly unsound. Thus any non-NL system utilising such co-operative strategies must allow for this kind of dialogue failure.

2.5 Exploitation

One of the most interesting facets of Gricean implicature, as far as NL is concerned, is the way that the maxims may be flouted or exploited for communicative purposes. For example, irony or sarcasm can be achieved merely by saying something that one obviously believes to be false or something with such a consequence: if a teacher says 'good afternoon Smith' to a pupil arriving late for a morning class, the implication would not be that the teacher thinks it is past noon. While one of our aims may be that of making computers behave less literally, (Jerrams-Smith, 1985: Houghton, 1984: Lewis & Norman, 1987) interpreting sarcasm seems a little superfluous to a computer system except perhaps by way of research into the mechanisms of pragmatic inference.

2.6 Conventional Implicature

Grice states that certain implicatures stem from particular lexical items and expressions and these are termed conventional implicatures. Such items as *but, even* and *yet* have been suggested as giving rise to conventional implicature in that they possess a component over and above their truth-conditional contribution to a sentence. (Reichman, 1985) makes use of such structural elements in the analysis of discourse structure). For example, *but* has the truth-conditional meaning of conjunction but also signals contrast. These items are often used to convey an attitude on the part of the speaker towards the information being conveyed.

Although, as we have seen, lexical items are often borrowed from NL for use in a command language, such conventional implicatures which might be attached to them in their NL context would appear to have little relevance in non-NL HCI. It is not normally expected that users will see any point in conveying to the system the fact that they feel that the current command is, for example, slightly unusual or that it contrasts with the previous command in some way.

2.7 Mood Restrictions

Another way in which NL dialogues can differ from others is that the former may involve different declarative, interrogative and imperative sentence types. These are referred to under the grammatical classification of *mood* and are closely connected to the 'speech acts' (Austin, 1962; Searle, 1969) of assertion, question and request but are not tied strictly to these. For example, the question 'do you know the time?' is not adequately dealt with as a speech act by responding to its interrogative form. Therefore, the answer 'yes' would be considered uncooperative unless asked with the obvious intention of ascertaining whether someone who needed specifically to know the time knew it. Similarly, 'I wonder if you might tell me the time' has the grammatical form of a declarative but also instantiates a request in speech act terms. Indirect speech acts are often involved in 'politeness conventions' in English.

In non-NL interaction, such variations are not applicable. As the name suggests, command languages are based on the view of the dialogue as a series of commands by the user, perhaps followed by a report of success or error by the system. Whether a user's intentions are signalled in one mood or another is of no account to the system and so have no functional role in the dialogue. Neither does there seem to be any cause to imbue direct manipulation or WIMP style interaction with this kind of subtlety of expression. Thus non-NL interaction should not require the kind of reasoning processes needed by NL interfaces which attempt to interpret indirect speech acts such as these in user input (but see Ehrich *et. al.*, 1986).

2.8 Applying the Maxims to Non-NL HCI

If I tell you that my brother has four 'O' levels and you happen to know that he has six, what do you make of this situation? Well, in the first place you might suspect that I am deliberately trying to mislead you or you may think that I do not know the facts. There are, however, other alternatives which are rather less obvious and which we do not normally pursue. If you

know me to be a competent native speaker of English, you will not tend to suspect that I do not understand the meaning of the words I have used but if I were not, then that would be conceivable. A much more radical scepticism would involve the possibility that I might be following an alternative cooperative convention or conversational maxim.

Let it not be thought that my utterance embodies an untruth. Having six 'O' levels entails having four. The reason that my utterance is misleading is that there is an expectation that I will follow a certain cooperative convention, the Quantity maxim in Grice's terms, which instructs me to say the 'strongest' thing I know to be true. This expectation is so deeply ingrained that, given the situation described, it is the last thing you would think of questioning, if you ever did at all. That is why it can seem like a matter of logical implication when it is not, which can be shown by the following:

(1) My brother has four 'O' levels, if not more.
(2) *My brother has ten 'O' levels, if not eight.

If my utterance meant or logically entailed that my brother has ONLY four 'O' levels then (1) would be as anomalous as (2).

Suppose that you knew :-
a) that I knew the facts of the matter.
b) that I knew that you knew the facts of the matter.

It is highly likely that you would still assume that I was cooperating but you would be forced to make inferences about what I intended to convey in order to maintain this assumption. The relevant maxim would ensure that, *ceteris paribus*, I would say 'six' and not 'four', allowing you, lacking knowledge to the contrary, to infer an implicit 'ONLY'. However, it could be that I am following an alternative convention which, in effect, enables the inference of an implicit 'AT LEAST'. This convention would work just

as well if universally adopted but would be less useful since the information conveyed would be less precise than in the case of the actual convention. The point of all this is to indicate that there are in fact a number of ways of reasoning about the intended meaning of an utterance if we are prepared to be a little perverse. Non-NL HCI may have its own pragmatics. When the literal content of an utterance is anomalous in some way, there are various ways of interpreting the situation:

i) Exploitation.
ii) Misuse of terms.
iii) Alternative convention.
iv) Factual Error.
v) Mendacity.
vi) Various combinations of i - v.

We will discount v since it is not relevant to our concerns. In human communication between native speakers it would be unusual to infer iii, ii or vi so that leaves i and iv.

If we apply this kind of reasoning to HCI, things are rather different. Firstly, we can assume that the prospective system knows all the relevant 'facts', whereas you may not have known about my brother's 'O' levels. When interpreting users, although we will not be interested in v and i, we cannot assume that ii and iii and possibly vi will not occur. The 'language' of the interface may not be familiar to the user.

However, we still have a means of reasoning about the interaction. Any utterance with imperative force will carry the generalised implicature that the utterer believes the propositional content to be false (or have the felicity condition or presuppose that it is false), that the utterer has the authority to issue the request and that the 'requestee' is in a position to be able to accede to the request.

Hence the request to close the window indicates *inter alia* that the requester believes the window to be open and that the requestee can close

it. For any particular request there may also be particularised implicatures arising from the nature of the requested activity. For example, In order to be able to buy a paper for someone the newsagent must be open and a suitable means of payment must be available. Particularised implicatures are those which are context specific and require world knowledge or knowledge of some domain in order to achieve a successful communicative transaction (Allen, 1983).

Interaction with computer applications involves the same kinds of restrictions and contextual factors in the successful management of human-computer dialogue. In the domain of a computer application, there may be many contextual factors determining the appropriateness of any particular instance of a user command or action. These restrictions may or may not be known to users, depending on their level of competence in that domain (and possibly other factors), just as one may not realise that the shops are closed when requesting someone else to buy a newspaper.

A requestee in possession of the facts would obviously apprise the requester of them rather than try to buy a newspaper (unless there happened to other means available). In this kind of case it would usually be adequate to merely give the information that the newsagent's shop was closed but there are circumstances, perhaps in the case of a foreign visitor, where it might be necessary to indicate the link between the buying of newspapers and newsagents' opening hours. When the requestee has reason to believe that the requester lacks domain knowledge, a cooperative response may involve the provision of any part of the domain information relevant to the anomalous request.

This is essentially the context in which much HCI takes place and it is with the aim of rendering the computer side of the human-computer dialogue more cooperative that these remarks are made. This approach also has relevance to other forms of interaction. For example, if we take this term to characterise responses which fail to exhibit the level of cooperativeness which could reasonably be expected from any well-disposed and informed person, one can see in many WIMP

applications a version of what Kaplan (*ibid*) calls 'stonewalling'. The practice of dimming menu items when the context renders their selection inappropriate prevents users from making errors but is effectively a 'no comment' strategy. The user is prevented from gaining any other information about that option except that it is not available. The precise details of how these ideas relate to concrete systems and live interaction are the subject of subsequent chapters but the general thrust of the argument may be evinced.

People who take all that is said to them in a literal fashion are judged to be pedantic or even obstructive and uncooperative. They also miss out on a large proportion of the expressive power of language. The literal content of an utterance is often only a cipher to the communicative action and the hearer may have to perform a significant amount of reasoning to appreciate the speakers intended message. Sarcasm is often wasted on children because they are allowed to be conversationally undisciplined with regard to conversational maxims and therefore do not always recognise their exploitation. Computer systems designed to date perform the role of pedant extremely well! It is no wonder that they are often difficult and frustrating to learn and use.

Interacting with computers is a relatively new activity, with its own particular characteristics. New methods of interaction, in the form of novel input and output devices and techniques, as well as general developments in computer science, constantly extend the scope and possibilities of the relationship between humans and machines. But humans are communicators *par excellence* and have evolved and developed rich and complex communicative functions which are so ingrained in our behaviour that, from day to day, we are scarcely aware of them.

Whatever the computer of the future looks like, we should at least make sure that its communicative behaviour is in tune with any fundamental aspects of our own. It is suggested here that the kind of reasoning supporting pragmatic inferences involved in human communication is one of these fundamantal components which should be addressed.

3. User Errors

This chapter is concerned with error situations in computer usage. It is suggested that an understanding of what is happening in such situations offers valuable insights into the general processes of human-computer communication. To this end, classifications of error are examined and a levelled linguistic classification is proposed. As part of this classification, the notion of context error is introduced and analysed using the theoretical constructs discussed in Chapter 2. Conclusions are advanced for the implications that this analysis has for dialogue design.

3.0 Introduction

The consideration of user error is an inevitable part of the process of designing computer systems, either from the point of view of attempting to minimise error commission or that of handling error when it occurs. To this end it is necessary to gain a good understanding of errors that can and do happen, together with their causes. Part of this understanding should be the realisation that errors are a function of the interaction between a user and a system and not the sole responsibility of the user (Booth, 1990b). Computer systems exist in order to serve the purposes of their users and not *vice versa*. The term 'user error' is not well chosen in this respect as Lewis and Norman (1987) point out.

Another common and largely tacit assumption about user error is that it is entirely negative and without utility. While nobody likes to make mistakes, this is not to say that there is nothing to be gained from doing so.

One would rather get things right and have done with it than have to deal with some anomalous situation but if, in so doing, a gross misconception is corrected then possibly the error was fortuitous and could conceivably prevent the commission of further and more serious errors. Making mistakes can be extremely effective in didactic terms and has its own rewards, as we hope to make clear.

3.1 The System Perspective

Any situation can be described in a number of ways. The book is on the table, the table is under the book, the book is not in the library and so on *ad nauseam*. The description of a situation as an error is a particular view of it and is only valid in respect to certain other assumptions. Errors can be viewed both from a system perspective and a user perspective and it is important that this is acknowledged, since both the user's and the system's contribution to the anomalous situation are relevant to its characterisation. Where the results of a user action are not as expected, this is an error as far as the user is concerned but not so from a system point of view. User misinterpretations of error or system/user model mismatches are often the result of conflict between the user and the system perspective of a problematic situation. For example, the 'file not found' error message common in many operating systems can result from various kinds of error viewed from the user perspective; typographical, spelling, synonym, pluralisation, wrong directory, wrong pathname etc. An important distinction within the user perspective is that between errors of *competence* and those of *performance* (Chomsky, 1965). Performance errors result from the incorrect implementation of a correct rule, whereas competence errors result from correct performance of an incorrect rule. As may be the case in any mismatch between system and user perspective on an error situation, although the error may be the same in system terms, the appropriate response obviously differs.

It is notoriously the system designer's failure to see things from the user perspective and the understandable inability of the user to see things from

the system perspective that leads to interfaces which perform poorly (Lewis & Norman, 1987). However, even with the best intentions towards user-centered design, design considerations will never be just a matter of what is best for the user. Other pressures, of an economic or technical nature for example, dictate system design features. What is important is that where these pressures lead to a conflict of basic perspectives between user and system, mechanisms for bringing the two perspectives into line are incorporated into the interface.

3.2 Error Classification

The purpose of classifying phenomena is to improve the quality of information about it. So long as the classification relates to fundamental underlying similarities and causes, it aids in the formulation of responses to the phenomena in question; rationalising approaches towards it as opposed to dealing with each event as it arises in an *ad hoc* manner (Booth, 1990b).

There are various ways in which errors can be classified. Some of these focus on the cause of the error in terms of its cognitive provenance. One such method starts by distinguishing between errors of intention and errors of performance. Norman (1983) takes this line, calling the former *mistakes* and the latter *slips*. Slips are held to be errors in carrying out a valid 'intention', while mistakes are errors in the 'intention'. Norman's distinction between *mistakes* and *slips* is an important one, in that the system responses appropriate for one are inappropriate for the other. By definition a slip does not require any correction in the user's model, whereas a mistake does. Thus it is important to be able to distinguish between them so that appropriate system responses are generated. This distinction corresponds to that between *competence* and *performance* alluded to above.

It is not clear from Norman's definitions that a maintainable distinction is made. Norman's classification can lead to taxonomic problems in that he goes on to say that there can be slips in forming 'intentions'. This presumably means that a) the result of the slip is a mistake in the form of

an invalid 'intention' and b) there must have been an 'intention' to form the invalid 'intention', which was carried out incorrectly owing to the slip. We apparently have an infinite regress of intentions here. Norman also classifies mode error as a slip. This entails that the 'intention' involved is valid, which further entails that the 'intention' is logically independent of the system state so that, in principle, the user may form a valid intention using a completely erroneous conceptual model of the system.

The problem lies in the concept of 'intention'. Statements of intention are inherently ambiguous. The claim that Oedipus wanted to marry his mother is true in one (transparent or *de re*) sense and false in another (opaque or *de dicto*) sense (Quine, 1960). While typing errors can easily be classed as slips, if someone misnames a file in a delete command (for example, typing 'rm address' for 'rm addresses') it is not clear that a genuine distinction can really be maintained between the idea that this is a case of a valid intention to remove a file, combined with a slip in specifying the filename, and the idea that it is a case of a belief that the file was called something other than it was without appeal to some as yet unexplained notion of intention as the 'mental pointing finger'? All in all, while this classification enables Norman to draw up some useful design recommendations, it does not seem to constitute the basis for a coherent, consistent and comprehensive taxonomy of error (not that this is Norman's claim).

Another classification scheme which focuses on the cognitive provenance of error is Booth's (1990a) Evaluative Classification of Mismatch (ECM). Here the discussion is not in terms of error but rather 'dialogue failure' signalled by user reports of misunderstandings, requests for help, illegal commands not arising from slips and uneconomic command sequences. The categories of mismatch are then derived from the relation of either an object or an operation to a symbol, a concept and possibly a context, producing eight different categories. This scheme appears to offer a comprehensive conceptual classification of model mismatch between system and user and to provide a terminological

framework for discussion of between user and system communication problems. However, it suffers the problem that some dialogue failures can be classified in more than one category.

		Object	Operation
	Concept	Object/Concept Mismatch	Operation/Concept Mismatch
	Symbol	Object/Symbol Mismatch	Operation/Symbol Mismatch
Context	Concept	Object/Concept/ Context Mismatch	Operation/Concept/ Context Mismatch
Context	Symbol	Object/Concept/ Context Mismatch	Operation/Symbol/ Context Mismatch

fig. 3.1 Booth's Classes of Mismatch following dialogue failure.

Booth (1990c) has also looked at the possibility of using the GOMS and TAG interaction grammars for error classification but has found these to suffer the same problem of multiple classification, as well as a tendency to allow decomposition past the point of usefulness for this purpose. It is worth noting that the kind of situation which is addressed here seems to allow almost any problem to count as dialogue failure. For example, the fact that a naive user did not understand the relation between a Macintosh disk icon and its related window is given as an example of dialogue failure, while it seems clear that there is no simple relation between these two 'objects'. Their relationship is functionally complex, involving many dependencies. What this situation describes, if it describes anything at all, is not a misconception regarding one of these functions but a lacuna, a lack of any useful mental model of any kind of that part of the system. Granted, the system has 'failed' to make the user understand this relationship via its iconic 'language' but isn't this expecting rather too much? After all, what are we comparing its performance with? What would a human instructor have to do in order to communicate these ideas to a naive user? It thus seems a trifle optimistic to expect such situations to fall into a simple

classification.

It is also worth noting that the other two dialogue failures which are cited are both amenable to the linguistic classification scheme detailed below. The Macintosh is prone to request excessive numbers of disk swapping operations in order to find data or applications distributed over more than one disk. To the uninitiated, this can be interpreted as an error situation. Under the classification scheme advanced in this thesis, this would be categorised as a pragmatic error on the part of the system, detailed as a failure to comply with conversational maxims concerning brevity or conciseness and one answer to this might be to ensure compliance, perhaps by having the system supply an explanation for the apparently obtuse dialogue moves.

The other dialogue failure can be classified as semantic under this scheme, in that the two partners assign different meanings to an action sequence.The "eject" menu selection ejects the floppy disk from the drive, while leaving its icon and any associated windows dimmed on the monitor. A user's expectation that these items will be removed from the display by this command will therefore be disappointed. Of course, problem classification does not determine or even necessarily suggest problem solution, as in this case. The so-called 'desk top' metaphor is ambivalent here, in that there are two totally different methods for ejecting discs; one graphical and one command orientated. There is no obvious reason for attaching the icon and window removal functions to either. In fact, it would be more in keeping with the metaphor and more consistent with file deletion if dragging the disc to the wastebasket erased the disc.

Carrying out some task using a computer broadly consists of identifying goals and objectives, determining sets of actions which will achieve them and performing those actions in the required order. This gradual decomposition of a super-ordinate goal into sub goals and ultimately keystrokes as part of an overall plan of achievement can be seen as a descent through various levels of activity. Moran's (1981) Command Language Grammar (CLG) characterises the interaction in this levelled

way in order to highlight mismatches between the *conceptual* and *communicational* components of a system. This framework has also been put to the task of error analysis (Davis, 1983a).

A task is composed of a number of sub-goals, each of which must be successfully completed to attain the overall goal. This is represented in CLG by the *conceptual* component, containing the *task* and *semantic* levels. These specify the tasks the user wishes to carry out (*task*) and the conceptual commands and entities available on the computer (*semantic*). In each level is a summary of the *procedures* available and the *methods* by which these *procedures* can be sequenced to accomplish a particular goal. The *communication* component specifies the available commands and the syntax rules (*syntax*) and the actual interaction itself (*interaction*), complete with prompts, user key presses and computer responses as the interaction proceeds.

Errors can be classified in terms of this characterisation of the interaction as conceptual, procedural, methodological, semantic, syntactic or lexical according to the level at which they are deemed to have their cause. This classification has the advantage that it addresses both the user and the system perspective of error. As it applies to command language interfaces, it also presents us with some familiar and well understood linguistic categories. However, being very similar to GOMS, it is likely that it suffers the same problems of excessive decomposition and ambiguity.

3.3 The Linguistic Classification of Error

As the motivation for the present research comes from studies of error in command line interfaces (Bradford *et. al.*, 1990; Hanson *et. al.*, 1984) and the approach undertaken here follows a natural language paradigm of communication, the classification scheme proposed is similarly linguistically orientated. The kind of error under study consists in command specifications which are intrinsically unexceptional but which are used in system contexts where they might not have been but actually are

inappropriate. That is to say that while it is possible for the system to manifest an appropriate context for the command, the prevailing context is otherwise.

There are several points arising from this which suggest the adopted classification scheme. Firstly, these errors are defined in relation to a single command and therefore do not occur at the task or plan level. This means that procedural and methodological levels are irrelevant to their characterisation. However, in order to distinguish them from other command level errors, lexical, syntax and semantic types are defined. Secondly, the nature of these errors closely relates to the pragmatic aspects of meaning in natural language. A usable definition is that a phenomenon of linguistic meaning is semantic if it attaches to the sentence and pragmatic if it attaches to the utterance of the sentence. In fact, as we have already noted, in natural language some allegedly pragmatic phenomena do attach to the form of words used (e.g. the verb **to fail** is seen to pragmatically imply the notion of an attempt) but it can be argued that these are semantic phenomena and, in any case, the definition seems workable in the more restricted computer command language environment.

There are other reasons why our classification is appropriate to the current research. As Thimbleby (1990, p. 50) points out, much of the classification is inherent in programming languages and particularly compilers, as well as in linguistics. The distinctions also correspond to distinctions of skill, rule and knowledge in the user and are best appreciated in terms of how they relate to errors. Lexical errors can be detected immediately, syntax errors need only to be reported by the system, while semantic errors have unpredictable results. To this we should add pragmatic errors, which also constitute a significant class requiring its own particular treatment. As the system instantiates the communicational rules and protocols of the command language and the situations we are interested in are those wherein the system signals that something is amiss, rather than cases of unexpected results of actions, linguistic error classifications represent a system perspective. By looking at the relationship

between system and user perspectives, the two can be brought more into line with each other so that the system can respond more appropriately.

3.4 Error Classification Scheme Categories

Lexical Error

Lexical error is essentially the use of a term which is not in the relevant lexicon. However, just as in natural languages, a command language system's vocabulary typically consists of closed and open sets. A closed set, members of which might be, among other things, the prepositions of the language, is (relatively) fixed for a language but the open set can shrink and grow as items are added or lost. Lexical items can also be seen as belonging to various grammatical categories such as verbs, nouns and so on. These distinctions are built into the system, as can be seen from the responses to lexical errors in closed and open set terms. An unrecognised command name (in UNIX® e.g. **ly myfile**) results typically in a 'command unknown' type message, whereas an unrecognised filename typically results in a 'file not found' type message. The system has an implicit representation of grammatical categories; it 'knows' (or its behaviour is consistent with the following of a rule that) file names are referring terms and belong to the open set. This is also evident from the actions that the system performs on receipt of a valid command. The upshot is that only command terms which are unmatched in the closed set are classed as lexical errors. Command terms in the syntactic position of those from the open set of referring terms but which are unmatched in the lexicon are designated as lacking referents and therefore fall into the class of pragmatic error.

Syntactic Error

Syntax errors are errors in the order or format of command terms. Missing, transposed or extraneous terms and separators fall into this category. As with lexical error, the command format or syntax is fixed (if macros are used, the commands they replace must be entered correctly and

the macro commands themselves are subject to their own syntax rules) and so there is no real problem in identifying them from the system perspective. Again, there is no one to one correspondence between user and system perspective with regard to these errors.

Semantic Error

The semantics of a command relate to its literal or logical sense or meaning. Semantic errors will therefore consist in the use of illogical (from the system perspective) combinations of command terms. The application of system operations or values to system objects to which they do not apply will be cases of semantic error, for example, attempting to list (ls) a file, make a file the current working directory or copy a file onto itself. These are equivalent to semantically anomalous sentences in natural language such as 'colourless green ideas sleep furiously' or 'the ball kicked the dog'. It is unlikely that this kind of error would arise through incorrect performance, since it would be quite unusual for the fairly random results of typing error to produce a lexically and syntactically correct command. Thus, these errors will almost always stem from a user misconception. However, that misconception may concern the logical or semantic constraints of the system and its language or the current arrangement of objects within it, in which case the error is contextual from the user perspective. For example, a misconception concerning the current working directory might lead a user into trying to list a directory 'X' with the command 'ls X'. The presence of a file called 'X' in the current working directory will entail the interpretation of the command as an illegal attempt to apply the *ls* command to a file, even though a directory X exists elsewhere in the system.

Pragmatic Error

From the system perspective, pragmatic or *context* errors are situations wherein a lexically, syntactically and semantically correct command fails because the system is not in an appropriate state. This should not be

confused with *mode* error, where an ambiguous action or command produces unwanted or unexpected results. Editors often have two modes: one for performing editing tasks, such as deleting a character, and one for performing actions such as saving the file. A typical mode error is issuing a command, e.g. **s** for 'save' while in edit mode, thus inadvertently inserting an unwanted 's' character into the file. Mode error has no system perspective, not being detected as an error at all.

The linguistic equivalent of a context error is a well-formed, semantically unexceptional sentence uttered in an inappropriate context. For example, it would be pragmatically anomalous to say to someone 'would you please open the window?' when the only window is already open or if that person happened to be rendered incapable of complying by disablement. From the user perspective, context errors can result from slips (e.g. the mistyping of a filename will be seen by the system as a reference failure) or mismatches between the system and the user's conceptual model. In the studies of UNIX® use cited above, context errors accounted for a large proportion of the errors made by UNIX® users; this is one of the reasons why it is being studied in this project.

3.5 Why Users Make Context Errors

Context errors arise from mismatches between prevailing system states and those implied by the erroneous command or action. These mismatches may reflect deficiencies in the user's awareness of the system. That is to say that the user either lacks knowledge about the prevailing system state or about the implied state. The former deficiency can arise from attention or memory lapse, as is usually the case in mode error, or it may arise from a lack of knowledge about the effects or implications of some previous command; if a user does not know that removing execute permission on a UNIX® directory will prevent its listing, the chances are that an unsuccessful attempt to list it will occur at some point. The latter deficiency may also concern knowledge of the preconditions of the erroneous command; a user may misguidedly attempt to list a UNIX® directory

knowing that it is execute protected, since it may not be known that having execute permission is a precondition on the **ls** command.

In order to avoid making context errors, users have to acquire and maintain some kind of conceptual model of the system, aspects of which are constantly changing as a result of the interaction. A conceptual model must not only reflect current system states but also an understanding of how the user's own actions combine with the functionality of the application to produce new states. These two aspects of the conceptual model are here referred to as the *state* and *functional* models reflecting transient and persistent system characteristics (Young, 1991). While experienced users will tend to have built up a better procedural model and their errors will relate mostly to lapses of attention and the novice's model may be inappropriate in both respects, the classification of users into novices and experts is overly simplistic. Competence will vary on a continuum across users and for users across system functionality (Carroll & McKendree, 1987).

3.6 The Pragmatics of Context Error

The relationship between user and computer can be viewed as that of dialogue partners. As the name implies, the command line interface assumes a dialogue form of command-response. In natural language, an utterance with imperative force involves certain general presuppositions, for example, that the speaker has the authority to issue it and that the addressee is able to carry it out. In addition, particular commands have specific presuppositions, for example, that the door indicated is closed at the time of the command to open it or that the addressee has immediate access to some requested object (Levinson, 1983). These presuppositions can be seen as particulars in the equations representing the reasoning about utterances, conducted in the light of general conversational maxims of cooperation.

If any of these presuppositions fail, then the command cannot be carried out as directed and the addressee will typically embark on some kind of

clarification or correction strategy, either entering into a dialogue with the other person or interpreting the request in some way which makes it possible to conform with it. Any clarification dialogue will typically include an explanation of why the command could not be carried out, in terms of the preconditions which failed. This may then lead to a revised command. For example, the addressee might state that the door was already open and the reply might indicate that another door should be opened as well.

On the other hand, the addressee may attempt to rectify matters by acting in order to make the failed presuppositions true. For example, he might fetch a requested object, thus making true the presupposition of immediate availability involved in the command. Or he might interpret the command in an intelligent manner so as to fulfil the identifiable intention behind it. He may open the window in recognition of the intention of the request of improving the ventilation in the room.

It is important to recognise that where requests arise in areas in which little expertise is enjoyed by the one making the request, misconceptions concerning rules and procedures governing that area, as well as misconceptions about states of affairs, will lead to requests embodying unwarranted presuppositions. These general presuppositions attach to the abilities and powers of the dialogue partners rather than to the details of the request itself and concern what is possible in any circumstances. The similarity between this aspect of human-human communication and the phenomenon of context error in human-computer interaction is demonstrable. UNIX® cannot move a non-existent file any more than I can play an eight note chord on a conventional guitar. It is suggested that the kind of dialogue that such situations engender is also similar and that this points to a particular kind of approach to dealing with context error.

3.7 Context Error: Problem Solved?

Context error is a natural consequence of the design of the command line style of dialogue, since, typically, the user may type in any command

and it is only evaluated when the **return** key is pressed. In the WIMPS interaction style, it is possible in many cases to preclude the incidence of context error by 'ghosting out', hiding or removing menu items and icons which have no valid function in the current context or relate to non-existent 'objects'. To some, this may appear to be an adequate solution to the problem. However, this approach may be misguided for a number of reasons.

Firstly, this approach can only be adopted for menus and icons; simply disabling keys amounts to a 'do nothing' approach which suffers the second disadvantage which is that the approach constitutes a bar on exploration; context error may be caused by a lack of knowledge of the preconditions for an action and making the action impossible in certain circumstances does nothing to correct this. In this case, it treats the symptom and not the cause. This argument should not be confused with the positive effects on learning which have been achieved by preventing users from accessing advanced features of an application; the so-called 'training wheels' approach (Catrambone & Carroll, 1987). That approach does not relate to contextual features, in terms of system states, but removes options for the whole of a period of time. The 'ghosting' approach removes options on the basis of current system states during the course of a single interaction. The idea that is being promoted in this thesis is that the user's ability to build and maintain a good mental model of a system is enhanced by the system's *support* of users' exploration of the system in the pursuance of their prevailing goals and plans. This idea challenges the 'ghosting' approach but is neutral with regard to the 'training wheels' approach.

Thirdly, there is evidence to support the view that errors enhance the conceptual model of a system. Frese *et. al.* (1991) found that error-based training produced better non-speed performance and free recall than error-avoidance training. Lastly, learning is an *active* process and is enhanced by interactivity as opposed to factual presentation (Carroll & Mack, 1984: p. 292).

Precluding context error may therefore be, in the long term, counter

productive, since substantial benefits may attach to the possibility of making errors. Not being able to do things wrongly does not imply the ability to do them correctly. A user's conceptual model may be such that he or she would make a context error if it were possible to do so and merely rendering it impossible does nothing to correct that model. These practices could be improved in this respect if, for example, selecting ghosted items provided an explanation of why the option is not operative in that context. (This being functionally equivalent to allowing the error to occur and then providing help). However, this would not be applicable in circumstances, for example, where items are removed from the display, as in the case of deleted files. The graphical user interface philosophy does not allow the speculative action which could give the system information about user requirements or intentions.

These arguments lead us to conclude that context error is not a function of particular interface styles but cuts across these distinctions. It also seems clear that direct feedback mechanisms in the interface are inadequate to enable a user to build and maintain an appropriate conceptual model of the system. After all, there will always be more information concerning system states and functions than it is possible or desirable to present to the user at one time. Even if it were possible to somehow cram everything onto the screen, the user will normally be interested only in the information relevant to his or her current concerns. The problem is in finding a way of determining what is relevant. It seems unlikely that this will turn out to be a static specification or one which can be generated from a static set of criteria. This being the case, what is sought is some interface component whose function it is to determine dynamically the information to which the user should have access.

Error situations constitute an ideal opportunity to provide relevant online help and there are a number of reasons for this. Errors themselves provide the help system with information concerning current user goals and possible deficiencies in the user's model of the system. Information can be presented in terms which relate to the user's current interests, rather

than as abstract descriptions of the system; the user's work can be put at the centre of concerns in the form of the objects and operations involved in the user's current task (Carroll & Mack, 1984: p. 294). Consequently, the information will be more digestible because it concerns issues on which the user is already focussed. Issues affecting the appropriateness of different forms of user support and the requirements that they impose, in relation to the current focus of context error are taken up in the following two chapters.

4. User Models

This chapter identifies the requirement of user modelling as a consequence of the need for the interface component to behave intelligently and cooperatively. Classifications of user models are compared and *desiderata* are presented for the user modelling enterprise. The notion of the user's Conceptual model is discussed and an informal approach to modelling user misconception concerning the device, based on a 'possible worlds' model of belief, is presented. The theoretical question of generating possible world models of user misconception is then addressed.

4.0 Introduction

For interactive systems to be be able to communicate intelligently with users it is necessary for them not only to have knowledge about their own states and functions but also to have access to knowledge about the users themselves. This has been acknowledged in research on adaptive user interfaces and intelligent tutoring systems and has grown in importance with advances in natural language processing (Kass & Finin, 1988).

Kobsa & Wahlster (1986) state that it has become evident that user models are needed not just for natural language interfaces and mixed initiative dialogues but also as a base for intelligent dialogue behaviour in general, in identifying the objects the dialogue partner is talking about, analysing non-literal meaning or indirect speech acts and determining what effects a planned dialogue contribution will have on the dialogue partner.

User modelling is difficult to define. Utilising general principles derived

from cognitive psychology as guidelines for the design of computer systems is a form of user modelling and so is the automatic inference of individual user characteristics in real time interactive contexts. User models can be implicit. System designers who have no explicit knowledge of user modelling will have produced systems which embody various assumptions about the people who will use them and these constitute implicit user models. Where the goal of user modelling is to represent in some way the user's cognitive structures as relating to the system or the task being carried out on it, task analysis can be construed as a form of user modelling. Formalisms like Moran's CLG (1981), GOMS (Card *et al.* 1983), TAKD (Johnson, 1985), and TAG (Payne & Greene, 1986) can all be seen as models of (expert) user knowledge and therefore as user models.

This wide diversity in what can be classified as user modelling has led to a certain amount of confusion in that different authors refer to similar models using different terms and dissimilar models using similar terms (Whitefield 1987, p. 57). Whitefield suggests that models may be classified according to what they are models of and who or what does the modelling. Identifying five possible objects (System, Program, User, Researcher & Designer) and four possible agents (Program, User, Researcher & Designer) provides Whitefield with a classification of twenty possible combinations and so twenty model types as shown in figure 1.

Although the utility of some of these is doubtful (e.g. the program's model of the researcher), this is a useful classification, not least because it allows various component models to be discerned in what are usually presented as unitary schemes. For example, Reisner's formal grammar (Reisner, 1984) combines references to both the program in terms of terminals relating to keystrokes and the user in terms of terminals denoting cognitive actions such as memory recall. Similarly the GOMS model contains a model of the user in terms of Goals and Selection rules and the program in terms of Operators. It is clear that most uses of modelling combine various types of model as defined in this taxonomy.

Whitefield's purpose for the classification is to consider how different types of model might be utilised in the system design process. The kinds of model that will be of most interest here are models of the user which the program may use in the course of the interaction. Even narrowing the scope this far still allows for a wide range of concerns.

Program's Model of System	Program's Model of Program	Program's Model of User	Program's Model of Researcher	Program's Model of Designer
User's Model of System	User's Model of Program	User's Model of User	User's Model of Researcher	User's Model of Designer
Researcher's Model of System	Researcher's Model of Program	Researcher's Model of User	Researcher's Model of Researcher	Researcher's Model of Designer
Designer's Model of System	Designer's Model of Program	Designer's Model of User	Designer's Model of Researcher	Designer's Model of Designer

fig. 4.1 Whitefield's Classification of Models

Kass and Finin (1988) describe a set of dimensions along which user models may be differentiated. For example, a model will be differentiated by the particular aspects of the user about which it contains information. These may be goals and plans (Carberry, 1988; Allen, 1983; Kaplan, 1983; Wilensky, 1983), capabilities (Goldstein, 1979), attitudes (Rich, 1979) or knowledge and beliefs (Konolige, 1981; McCoy, 1988). User models may be generic, that is representing users in general or groups of users, or they may be specific to particular users. They may be static or dynamic, either staying unchanged once constructed or being modified over time. They may be short term, such as might relate to models of user plans, goals and beliefs or long term if related to relatively stable user attributes. They may be used

descriptively as a kind of database of user attributes or prescriptively in the form of a user simulation. The method of acquisition of the model also distinguishes it. Explicit acquisition is where the user provides information directly. If the information is inferred from user behaviour, the method is implicit. Systems may use multiple models. Generic user models in the form of stereotypes may be used as an initial estimate of user attributes, this being modified as and when new information about the user is available. The above collection of distinctions also equates broadly with that given by Murray (1987).

A form of user modelling especially important to intelligent tutoring systems, intelligent database query systems and intelligent help/advisory systems is the modelling of user plans, goals and beliefs. In these circumstances it is very useful for the system to be able to reason about and infer the user's intentions and misconceptions, as may be revealed indirectly from their actions. 'One of the most important components of a user model [in query dialogues] is a representation of the system's beliefs about the underlying task related plan motivating an information seeker's queries.' (Carberry 1988, p. 23). In intelligent tutoring systems and help or advisory systems it is equally important that the program be able to identify misconceptions behind the specific errors that users and students make so that the source of the error may be corrected rather than its mere symptom.

4.1 Realistic Models

Since these models are meant to represent real human cognitive states and processes, it is clear that a certain amount of notice must be taken of what can be gleaned about real user's mental models. Although our access to sources for this are limited, there are a number of general remarks which will serve to give the flavour of the kind of caution that is being advised. Humans are not logical. That is to say that we do not, for example, believe all of the logical conclusions of our beliefs. We may also hold contradictory or inconsistent beliefs. Humans are also fallible. We forget things, fail to understand information presented to us and believe things which are false.

We also tend to hang on to familiar propositions despite evidence against their veracity, making *ad hoc* interpretations (Mack, Lewis & Carroll, 1983; p. 190; Norman, 1983a) or employing 'magic models' (Thimbleby, 1990; p. 24) in order to rehabilitate our preferred model. Similarly, we tend to generalise from what we already know. We employ inductive reasoning on the basis of a small amount of evidence or reason analogically from one situation to another. The use of mental representations is supplemented by contextual information available at the time of interaction or task performance, suggesting that mental models need not be in any sense complete or comprehensive where (even vital) components may be safely left 'in the world' as a kind of 'extended memory' (Mayes *et al* 1990). Relatedly, representations which are required to last in the form of Long Term Memory (LTM) are stored in semantic form rather than in the detailed sensory forms associated with Short Term memory (STM) (Thimbleby, 1990: p. 34). Recall is harder than recognition. These last two points may also help explain the effect found in Mayes *et al.* mentioned above. People also have emotional responses to situations which interact with and can interfere with their performance of cognitive, intellectual and physical tasks.

Although theoretical and experimental research in the field of cognitive science cannot be said to have provided any real generally applicable and absolute standards addressable to the design of computer systems (Landauer 1990), the point of all of the above is to stress that any model of user knowledge or intention constructed by any agent should not strain the boundaries or fly in the face of the limitations and attributes that empirical studies of user cognition have suggested to be psychologically valid. Thus, a user model which attributes to the user the quality of believing all of the logical consequences of a set of held beliefs or that of never holding contradictory beliefs will probably get it wrong.

4.2 How it Works versus How to do it

One of the distinctions brought to bear on the question of what knowledge users employ in interacting with a computer or other kind of device is that between procedural (how to do it) knowledge and conceptual (how it works) knowledge. The distinction can be likened to the difference between reciting multiplication tables by rote and reciting them by working out each addition along the way. The former consists in the knowledge (or behavioural conditioning) of what comes next in the series, while the latter implies an understanding of the mathematics involved from which the series is deduced. Procedural knowledge may also be characterised as providing a direct mapping between user goals and tasks, on the one hand, and user actions, on the other, whereas conceptual knowledge mediates between them by providing inferential connections. The distinction is important for the purposes of determining the nature of any information the system may be required to present to the user, either in the form of interface controls or responses or training material of whatever kind. The research to date tends to suggest that users make use of both types of model in a variety of situations and that neither kind is of itself superior.

Kieras and Bovair (1990) suggest that a conceptual (how it works) model can be useful to a user if the knowledge about the internal workings of the system or device allows the user to learn to use the system more rapidly and to infer how to operate the device. However, they also point out that such knowledge need not constitute a complete *device model* and that it should relate to specific control actions rather than general principles. We might add that whether or not such knowledge is useful also depends on how closely the user's task is mapped onto the system's objects and functions. For example, the hierarchical file systems of many operating systems are pointless to the user who does not have some kind of conceptual model of it.

Evidence to support the utility of procedural training material is also available (Cantrambone & Carroll, 1987). Mayes *et. al.'s* (*op. cit.*) experiments may be taken to suggest that the user's long term model is more conceptually than procedurally based, since the former is intrinsically

more semantic along with long term memory. It should also be noted that the question of levels of granularity comes into the discussion. A conceptual model of a computer system could conceivably vary in detail from a bit level representation of a file to a more abstract file concept as a set of data, just as a procedural model may involve gross level actions such as saving a file, as well as the key strokes or physical actions required to do this.

4.3 State and Functional Models of a System

The term 'mental model' is used here to denote a user's mental representation of a system in terms of its internal states and processes and is therefore used to denote 'how it works' knowledge. The present purpose in reasoning about users' mental models is the identification of user state misconception. In order to avoid making context errors, users have to acquire and maintain some kind of mental model of the system, aspects of which are constantly changing as a result of the interaction. A mental model must not only reflect current system states but also an understanding of how the user's own actions combine with the functionality of the application to produce new states. These two aspects of the mental model are referred to here as the *state* and *functional* models.

4.4 Modelling Users' Misconceptions

From the system's perspective, a context error is apprehended as a mismatch between the prevailing system state and some state feature(s) implied by the erroneous command. For example, the command 'rm myfile' presupposes *inter alia* that the file "myfile" exists in the current working directory. However, we cannot infer from this that the user's state model has this false proposition as a component. There are other configurations of the user's mental model which could account for the error. The problem of identifying user misconceptions is thus one of identifying one from a number of possible configurations of the user's mental model.

On the face of it, what we have here is a case of diagnosis similar to that of medical diagnosis, as implemented in expert systems such as **Mycin**

(Charniak & McDermott, 1985). Presented with symptoms (an erroneous command), the task is to infer the underlying disease (misconception) by looking at a set of rules instantiating empirically derived, medical knowledge. However, the situation is significantly different in that the relationship between symptoms and causes is much less clear in the present case. Whereas, in medicine, particular, known diseases have associated sets of known symptoms, no such well-established, empirical links are available to connect particular errors with the misconceptions which causes them. Not only is there little empirical knowledge but it is also debatable whether or not a knowledge base of this kind is even feasible for the present knowledge domain. The diseases (misconceptions) may be novel and the symptoms (erroneous commands) only indirectly connected with them. If we do not know what the possible diseases and their symptoms are, then we cannot proceed by matching symptoms presented with those that are attached to known diseases. The process has to be basically a generative one.

It is proposed that the notion of possible worlds can be used to good effect in dealing with the problem of determining user misconception. A necessary truth is a proposition that is true in all possible worlds. If a proposition is possibly true, then it is true in at least one possible world. A necessarily false proposition (being not possibly true) is true in no possible world. Thus we can think of propositions which happen to be (not necessarily) false as being true in some possible worlds. For our purposes, we are only interested in a certain subset of facts in any possible world, these being the ones which concern the computer system in question. All other issues can be gathered under a *ceteris paribus* clause so that we only consider possible worlds which differ from actuality with regard to propositions concerning the states and functions of the system in question. This means that the possible worlds up for consideration are limited to those in which the system has the same design, and therefore the same constraints, on possible state configurations as the the actual system. Admittedly, this is an arbitrary restriction. It is possible that a user might entertain misconceptions which go beyond the 'physical' limits of the system design,

for example, thinking of the directory structure as a network rather than a tree and thus that a directory could be its own ancestor.

However, the line has to be drawn somewhere if the number of possible misconceptions is to be less than infinite and it is unlikely that any automatic (or even human) system would be capable of diagnosing such 'bizarre' misconceptions from a single erroneous command. In any case, users are not passive learners and are quite capable of inferring characteristics which are not directly presented (Carroll & Mack, 1984). The statement of a general rule of the system flouted by the command, as part of a blanket response, plus the failure of the command, should enable the user to identify the area of their misconception at least.

Thus, misconceived system states are limited to those which the actual system design could support but which happen to be counterfactual. For example, the command 'rm myfile' implies a possible world in which *(inter alia)* there is a file called 'myfile' in the current working directory. The system design itself is neutral with regard to the truth or falsity of the proposition that there is such a file (we might label such propositions 'System Contingent'). System functions are fixed and unaffected by user actions and so any misconception concerning system function inevitably will implicate possible worlds in which the system functions in counterfactual ways. This set of possible worlds is therefore more open ended, in that it is not restricted to worlds in which the design of the system is identical to the actual system. There will consequently be a need to somehow restrict the scope of possible worlds considered, otherwise the search space is potentially infinite. In fact, the response adopted here for functional misconceptions is to attempt to identify the system rule which the user ought to believe but which is not believed, rather than to try to generate the actual false rule believed by the user. The user is expected to be able to infer from our statement of the rule that their incorrect rule is wrong, correcting the misconception indirectly.

The problem of trans-world identity does not arise for us here as we are interested in all worlds which might correspond to a user's conceptual

model. We are just as interested in an item in a particular possible world which is identical with some item in the actual world as with one which is a different entity with similar attributes, e.g. the same name.

We can think of these worlds as existing (or subsisting) in a space of logical similarity. If we imagine the actual world at the centre of this space, then those possible worlds most similar to it will be closest to it in logical space and those which differ radically will be somewhat further away. We will refer to this as logical proximity. The amount the actual world would need to be changed to transform it into a particular possible world can be used as a measure of logical proximity.

4.5 The User's Mental Model

The point of these possible worlds is that they can be thought of as a set of hypotheses about the user's model of the world (the system), since they are 'generated' from a user command which is, we assume, intended to work, i.e. a command which the user assumes is consistent with the actual world. The next step is to choose from among a number of candidate possible worlds one which is deemed most likely to correspond to the user's model of the system. Logical proximity can be taken as an initial measure of best fit, since it credits the user with 'distorting' the actual world as little as possible. In other words, it assumes the user behaves rationally.

However, logical proximity only gives us a partial aid to discriminating between possible worlds, since it only addresses one dimension of user psychology; one which we might call 'conservation'. By this is meant the natural tendency of the user's model to 'track' the actual world rather than radically diverge from it, stemming from the user's desire to maintain control of the interaction and to 'know what is going on'. This use of possible worlds has similarities with Nozick's (1981). Other factors, such as failures of short term memory affect the user's ability to do this and these need to be taken into consideration so that a measure of 'psychological proximity' can be determined.

Thus we imagine the set of possible worlds generated by the erroneous

command as subsisting in a psychological space and, again, some will be closer to the actual world than others. Psychological proximity is the epistemic closeness of a possible world to the actual world. Thus we have to seek evidence which is held to support or undermine the user's belief in the various possible worlds compatible with the anomalous command. Of course, there will be a cut off point for proximity beyond which it is considered highly improbable that the possible world concerned could be believed in by any user. This will limit the search space by reducing the number of possible worlds requiring consideration. (An alternative would be to pursue a notion of epistemic possibility in order to limit the search space to those possible worlds which are compatible and consistent with what the user knows. However, this would require the creation and maintenance of an extremely detailed model of user belief, which would be a significant achievement in itself. Therefore this avenue is not pursued here).

We have noted elsewhere the difference between system state and system functionality. The latter relates to the rules of the system and is static in nature, while the former is constantly changing. This means that knowledge of function is more likely to be stable and less prone to short term memory limitations. As is evident in the protocol analysis described in chapter seven, short term memory problems are legion with regard to state information concerning the UNIX® file system. This leads us to two different types of justification for psychological proximity.

The kind of evidence required to justify the user's belief in a possible world where the functional rules of the system are other than the actual rules will relate directly to the user and will generally be disconfirming. If a user has negotiated a particular function (F) of the system several times successfully or has been notified of a certain rule on several occasions, then a possible world with an 'incorrect' rule (F) is less likely to be that believed in by the user. Thus a user model of competence or knowledge can be used as a source of evidence in disconfirming user belief in functionally distorted possible worlds. Note that a system function may embody several rules and

that a single invocation of a function may not involve all of the rules relating to that function. Thus functional knowledge should be represented in terms of these rules and not in terms of whole functions.

As state model mismatches relate more often to short term memory limitations, evidence in support of state-distorted possible worlds is unlikely to be derived from this source. However, past system states can provide such evidence of psychological proximity. What we seek is any justification the user might have for favouring one possible world over another and the fact that a particular possible world was actual at an earlier time may provide such justification. For example, if the change that needs to be made to the actual world in order to transform it into a certain possible world is equivalent to the 'undoing' of some past user action, this may mean that the user has forgotten that past action and believes the system state to be that which would have been the case had that action not been performed. Of course, these circumstances are also compatible with a possible world in which the past action does not have the effect of rendering the later action invalid and that possible world is also a candidate for the user's mental model and misconception. In order to resolve this ambiguity, we require evidence to support one candidate over all others. Failing this, all candidates may assume equal status and so it will be necessary to correct for all of them and allow the user to identify the information appropriate to his or her misconception.

4.6 Generating Possible Worlds

A formal representation of a possible world need be no more than a set of propositions true in that world. As the possible worlds that we are interested in are ones that will be substantially the same as the actual world, we can concentrate on the differences rather than the similarities. Thus a possible world W1 is deemed to be a version of the actual world W0 in which a small number of propositions are altered. How they are altered is determined by the erroneous command, for we are only interested in possible worlds in which that command is consistent or compatible.

To illustrate this let us take a simple example from the UNIX® file system. Suppose there is a root directory (A) which contains two sub-directories (B) and (C) and one file (X). At a certain point, the current working directory is B and a command is issued "rm X". This command is incompatible with the actual world, since there is no file (X) in the current working directory (B). However, there are a number of possible worlds with which this command is compatible, as represented in figure 1. Unless stated otherwise, the world Wn is taken to be identical to the actual world.

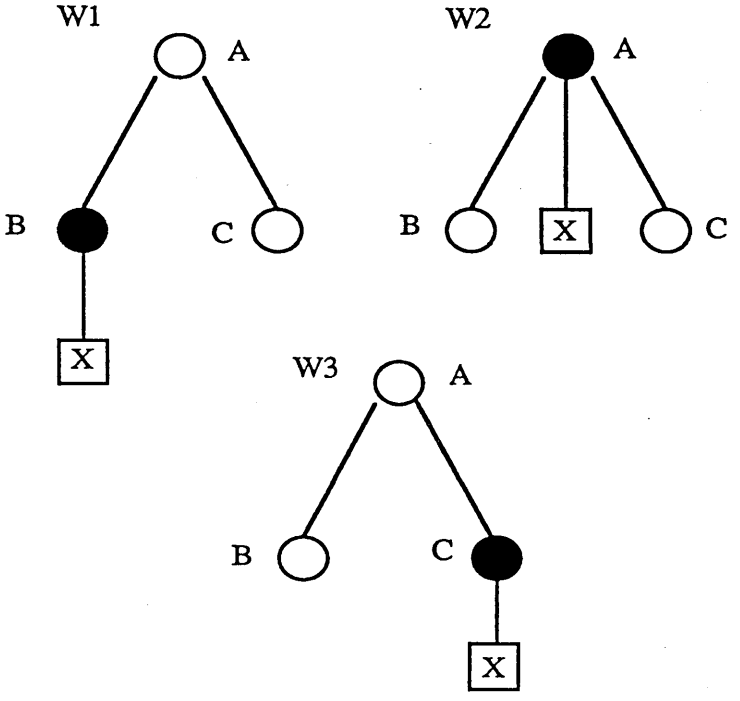


fig. 4.2 Possible Directory Configurations

W1: A file (X) exists in directory (B)

W2: (A) is the current working directory.

W3: A file (X) exists in directory (C) and the current working directory is (C).

W4: The command 'rm' can access sibling files.

W5: ...

etc.

These possible worlds are generated by a combination of the erroneous command, the system state and the rules of the system. Such multiple solutions are a feature of programming languages exhibiting *polymodal* parameters such as Prolog (Thimbleby, 1990 p. 352). It is only a matter of being able to specify commands, system states and system rules in the programming language in order to generate possible world solutions corresponding to W1, W2 etc in the form of Prolog facts and clauses.

Given a range of solutions W1, W2,...Wn, it is then necessary to select the most likely candidate for correspondence with the user's mental model. In logical space, W3 is less close to the actual world than W1 and W2, since it involves two alterations. In psychological terms, we need to look for facts that the user might be justified in taking to support a belief in one world over another. We can characterise this in terms of possible world histories. The system state is a result of its initial state and its history, in terms of the events which have transformed it. Any possible world can be derived by following a possible world history, in the form of some possible transformation(s), as opposed to the actual course of events leading to the current system state.

A particular possible world is psychologically/epistemically close to the actual world if there is a feasible possible world history from some prior state of the system resulting in that possible world. The divergence between actual and possible world histories might be constituted by the presence or absence of some possible transformation. For example, the absence of a change of directory in some possible world history would result in a possible world identical to that corresponding to the prior system state, whereas the presence of some possible transformation in a possible world history would result in a possible world different to that corresponding to both the prior and current (identical) system states.

Of course, possible world histories may also differ from the actual in terms of the number and nature of possible transformations. However, the notion of psychological proximity demands that possible world histories be 'feasible', in that there be reasonable or rational grounds for belief in that

version of history. This would be the case if, for example, there was some past action whose effects were to make fail the one precondition of the erroneous command which rendered it thus. If the undoing of a past command would rehabilitate the erroneous command, this corresponds conceptually to the situation where a user has forgotten performing the associated action (or in some cases miscalculates its effects), which constitutes a feasible possible world history, culminating in a possible world according with the user's state model of the system as one generated by the erroneous command.

The question of introducing additional possible world histories through possible world system rules will not be pursued here. As is indicated above, this option is seen as introducing vast difficulties, because of its open-ended nature, and added complexity in the form of criteria for logical and psychological proximity of possible world rules.

4.7 Conversational Implicature and Possible Worlds

At this point it seems prudent to make explicit the relationship between conversational implicature and the possible worlds model as discussed above, as it may not be completely clear to the reader how these two approaches constitute different views of the same phenomenon. When conversational maxims are *exploited,* as discussed in Chapter two, the assumption is that the cooperative principle is being adhered to on a 'deeper' level than that of literal meaning. In these circumstances, it is necessary for the hearer to reason about the utterance in order to maintain the assumption that it is indeed a cooperative response. For example, if you ask me what the time is and I reply that the street lights have just gone on, You would have to reason that I believe that the street lights come on at a certain time and that I believe that you know (at least roughly) what time that is, in order to maintain the assumption that my response is cooperative. Thus, you have constructed a conceptual model for me or, in other words, a possible world consistent with my utterance's cooperative status. If it happened to be the case that you did not know what time the street lights

come on, you would be able to identify the misconception which led to my consequently, erroneous response.

In the case of the computer user, we must still maintain the assumption that the user's commands are issued in the cooperative spirit in order to make any sense of them. If they make sense given a literal interpretation then the system can obey them in a straightforward manner. And if they do not, then it is still possible to assume that the user is behaving cooperatively but that the behaviour is based on a different world model from that with reference to which the commands are being interpreted. It makes no difference that, in the human to human case, the interpretive clues are *deliberately* encoded into the utterance, in contrast to user errors. In both cases, the interpretational process is based upon the relationship between respective belief systems of the participants in the communicative act. It is just that the computer system is the participant who is always right and that (unlike the native speaker) the computer user may also be unaware of the conventions and maxims associated with the system. Such a user is like the person who, in declaring that his brother has six children, follows the perverse maxim of quantity: "Only say things which you know to be *at least* true". Any native speaker of English apprised of the facts might deduce either that the speaker has his facts wrong (state misconception) or his linguistic conventions wrong (functional misconception). Thus the 'rules' or functional characteristics of a computer system correspond to the conversational maxims of natural language.

The possible worlds model is merely a way of making the relationship between respective belief systems more perspicacious, by allowing us to talk of their logical and psychological proximity. In the next chapter the issue of how such models may be utilised in support of the user's effective interaction with the system is considered.

5. User Support

This chapter concerns issues of user support and the various methods employed to aid users in learning and utilising computer applications. The importance of user support, as a factor of usability, is stressed and the relative effectiveness of various approaches is estimated. The requirement for intelligence in the interface is identified and some examples of intelligent interface components are reviewed. The design of an intelligent interface component, based on the analysis of the previous two chapters is discussed.

5.0 Introduction

It is not easy to give a simple characterisation of what constitutes the user interface of a computer system. From a systems perspective (Kammersgaard, 1990), it can be viewed as those parts of the software and hardware via which the user accesses the functionality of the system and via which the system delivers its responses to the user. However, on closer inspection, it can be perceived that design decisions involved, for example, in how system responses are to be delivered and the assumptions made about users and the environment in which the system will be operated are embodied in the design and contribute to the usability of a system. For in talking of good and bad interfaces, what we are concerned with is whether the users of a system find it easy, effective and efficient to use and moreover, find it *useful* and hence, use it (Booth 1989, p. 112). It is not difficult to see how this issue can be extended to include such matters as

user consultation in the process through which computers are introduced into a working environment by management, training and support services, ergonomic factors such as seating and lighting and a whole host of considerations which initially may seem remote to the system designer.

5.1 The Importance of Usability

Although computer science has advanced enormously in the last thirty years, the general perception of HCI practitioners is that the systems designer's knowledge and understanding of the user has not changed and that it is the communication between system and user which constitutes the biggest obstacle to the efficient functioning of many systems (Booth, 1989). The problem stems from the historical fact that in the early days of computing it was necessary to understand how computers work in order to be able to use them or even to have access to them. Thus the prevailing user group consisted of a clique of 'cognoscenti' and the use of jargon and technical terminology became a norm in the production of systems whose sole users would be members of the clique (Lewis & Norman, 1987). Unfortunately, these habits seemed to prevail, even as the use of computers spread and the backgrounds of users became more diverse, so that people with no knowledge of the internal workings of computers are still too often faced with awkward data entry procedures, arcane, terse or hostile error messages, intolerant error handling and confusing screen layouts. Consequently, the costs in terms of poor performance, decreased job satisfaction and even absenteeism and high employee turnover rates are often a more significant factor than the cost of the systems themselves (Sutcliffe, 1988; Shakel, 1990).

The study of human-computer interaction involves the analysis and organisation of these factors with the goal of providing methods through which designers can deliver more usable computer systems. Having indicated above that the boundaries of the user interface are not clear does not prohibit the making of some divisions and distinctions which render the problems involved in delivering usable systems more manageable - divide

and rule has to be the order of the day in any case, as it is rare for any one person to have control of and responsibility for the enormous range of considerations which are necessary for the stated goal.

5.2 Intrinsic Factors

Although this discussion is not intended to address the issues of interface design *per se,* the main area of concern here being extrinsic aspects of user support, design issues are bound to arise in the consideration of user errors and their causes and remedies. The first line of user-support, after all, is inherent in the way in which the system delivers information regarding task performance in the course of interaction. It is the responsibility of the interface designer to ensure, as much as possible, that the application aids the user in maintaining an awareness of the state of the interaction by feeding back information relating to the system state and changes in it. The way that this is attempted varies between systems but two distinct approaches can be discerned. These are the state oriented approach, exemplified by the Apple Macintosh, and the history oriented approach of the command line interface (Cowan &Wein 1990). Originally based on a hard copy terminal, the command line style of interface, for example the standard UNIX® interface, presents no state feedback. A successful command is followed by a return to the prompt, while the terminal screen or window contains a record of previous commands and system responses to which the user has access for purposes of inspection, copying or re-executing commands. In some cases there is a scroll buffer which may hold the interaction record for a large part of the current session. The display is not updated to reflect subsequent system states.

In this style of interface, the user infers the current system state from the history of commands and responses, together with knowledge of how user actions interact with system functions to produce new states. Although it is possible for the user to gain more information concerning state features, for example, by use of the *pwd* or *ls* commands in UNIX®, these tasks are incidental in that they do not represent primary goals, pursued for

their intrinsic value. They do not affect changes in the system and are usually precursors to actions which do. A more usable interface would obviate the need for as many of these 'incidental' actions as possible. This can be achieved in part by such means as configuring the system so that the prompt displays the current working directory, for example, but such features only deal with a limited range of orientation problems. This style of interface provides very little in the way of support to the user who has an incorrect functional model of the system. States will tend to be wrongly inferred from the history of the interaction if the implications of a command are not known and errors will be compounded as confusion arises over whether the error concerns system states or procedures.

5.3 State Oriented Feedback

In contrast to the command line style, the state oriented interface, often referred to as the direct manipulation or WIMP (Window, Icon, Menu, Pointer) interface, displays a graphical representation of certain aspects of the current state, which is updated as events in the shell are received. The display must contain enough information to enable the user to infer the state information required to predict the outcome of any command. However, it is possible to present only part of the system state and users may execute commands in order to obtain more information. These actions inevitably alter the system state (the *probe* effect). The existence of desk accessories such as 'Find' and 'Locate' also suggest certain shortcomings in the state oriented approach, since they supplement state information provided by the interface.

The state oriented interface is more explicit or revealing in terms of the effects or implications of actions in that they are often graphically represented as they occur. However, these are once-and-for-all, fleeting events and there is no intrinsic potential within the state oriented philosophy for relating current situations to past events as part of any explanatory capabilities, as there is in the history oriented approach.

If it is unreasonable to expect direct feedback to provide all of the

operational knowledge that any user might need in the course of interaction (and the view here is that it is), then the user must be provided with alternative means of accessing the required information and acquiring the relevant skills. There are a variety of ways, with varying levels of success, that this extrinsic kind of user-support can be achieved and we turn now to a review of these.

5.4 Training

It is very reassuring to have expert advice on hand when learning to use an unfamiliar piece of equipment and much frustration and annoyance can be avoided when this facility is available. Good training courses will provide a structured introduction to the system to be learned, allowing new users to progress at an optimum rate and in a relatively smooth manner. They also allow users to combine a variety of learning strategies into the overall learning process. Instruction and documentation is usually heavily supplemented with practical experience in performing realistic tasks and the problems in understanding which are inevitably encountered can be dealt with relatively painlessly by the tutors. The opportunity for discovery learning is often possible within this framework, allowing users to consolidate and extend their knowledge by their own efforts in attempting to predict aspects of the functioning of the new system. This mixed method of learning is something of an ideal in that a number of approaches can address different learning strategies within individuals (Carroll & Mack, 1984). However, training courses are expensive and usually introductory in nature. Although they can provide a useful start for the new user of a system, users tend to forget less well used procedures and functions and courses rarely provide ongoing support for such eventualities. The quality of courses will also obviously vary.

5.5 Documentation

Paper documentation is the most basic form of extrinsic user support and most modern systems and application software comes with at least one,

if not a set of user manuals and perhaps technical system documents for the system administrator/installer. User manuals are now commonly divided into 'getting started', tutorial and reference sections covering the various aspects of system or application functionality and use. The division between tutorial and reference materials reflects the *how to do it / how it works* distinction often found in discussions of user's procedural vs conceptual knowledge (Carroll, Aaronson 1988). It has been found that new users perform better when presented with as small a manual as realistically possible, rather than a lengthy one containing detailed explanations, and that performance in realistic tasks is improved if such so-called minimal manuals encourage the user to infer some of the information about the system instead of explicitly stating it. This phenomenon is probably due to the enhancement of conceptual knowledge that inferential reasoning about the system brings (Black & Carroll, 1987).

Although the quality of written manuals has shown some improvement in the last ten years, their effectiveness remains minimal for a number of reasons. Firstly, the use of a manual is often seen by users as a nuisance rather than a facility since it takes them away from their primary goal; the task they wish to perform using the computer system (Carroll, & McKendree, 1987). If they are used at all, it will often be as a last resort, a necessary evil, when all direct attempts to achieve the desired goal have met with failure. Secondly, manuals are static and cannot adapt to the users' level of knowledge or the specific nature of their query; bad habits, inefficient usage and lack of knowledge about facilities are not addressed and users have to recognise the need for help and be able to formulate it in a way that fits the organisation of the manual (Erlandsen & Holm: 1987). Thirdly, whereas the tutorial type of exposition of some user manuals is predicated on the idea of 'active learning', this approach actually does not alter the basically passive role of the user in following a set of instructions. Too often users complete the relevant task without gaining any understanding of what it is they have done. Lastly, user manuals are often misplaced, lost or anonymously borrowed and thus cannot always be relied

upon as a source of support.

5.6 On-line Support

On-line documentation has the advantage that it will not normally be removed from access by the user and can be called upon at any time without the user having to move from the computer. However, it has a number of disadvantages. In common with paper documentation, it is static in nature and cannot adapt to the users state of knowledge and formulation of the problem. The user must be able to translate the problem to suit the structure of the on-line system in order to make best use of it. Also, in some systems it is necessary to scroll through large volumes of text in order to find the desired information.

Where on-line documentation allows the user to specify a topic about which help is sought, the topic typically has to be specified in system terms rather than from the user's perspective of the problem. For example, the UNIX® 'man' command must be supplied with a proper UNIX® command and so the user must know which command relates to the question in hand. So-called 'context sensitive' on-line documentation presents only information pertinent to the current state or mode of the system, thus limiting the volume of material requiring sorting and menu driven on line help systems can provide a more structured access to information but, again, it is difficult, if not impossible, to make these systems adaptive to particular situations (Houghton, 1984).

In a more active vein, some systems provide prompts to the user if the required input is not forthcoming. For example, VAX/VMS will prompt for missing parameters in a command. A more sophisticated kind of prompting is implemented in Interlisp's DWIM (Do What I Mean) where incorrect input is interpreted via a spelling corrector and the intended input presented to the user for confirmation (*op. cit.*). In some cases DWIM will auto correct without consulting the user but this can produce unexpected results (Lewis & Norman, 1987).

5.7 Intelligent Help Systems

The shortcomings of the kinds of user support described above have prompted many attempts to provide users with the kind of help facilities which might be expected from a human expert, without the associated costs. Some of the perceived qualities of such a system are the abilities to answer natural language questions submitted in the user's own terms, diagnose incorrect or inefficient user procedures and extend the user's knowledge of the system by introducing untried facilities.

There is, of course, a great deal of overlap between automated help and coaching or tutorial systems. Both are intended to promote in the user facility over a domain of knowledge or skill and in both, the topic within the knowledge domain addressed may be determined by the user, whether implicitly or explicitly. The main differences are that the knowledge domain of a help system will always be *internal,* in that the information provided concerns the use of some application, as opposed to being *external,* where the information concerns matters unrelated to the application in use. Thus, a user of a word processing package will receive *help* about that package and its use, whereas the *coaching* within an automated tutorial system will concern not the coaching system itself but some other sphere of knowledge or skill. Also, in the help system context, the interaction is always *bona fide,* whereas a tutorial system may take the user through specific *training* exercises and 'simulated' or 'dummy' interactions.

Intelligent user support can also be classified according to a number of other criteria. Systems may be *passive*, requiring the user to recognise the need for help and to request it, or *active*, where the need for help is recognised by the system. They may be *static,* in that the content of the information that they provide does not change as a result of the interaction, or *dynamic,* wherein the help provided is modified as a result of the interaction being monitored. Modifications may address both the content of information being accessed and its presentation. For example, poor user performance might require the help system to access low level knowledge

bases and to present detailed information, whereas a user designated an expert might need only brief messages on advanced topics. A mapping of these help system characteristics to users' information requirements is also possible (Lutze, 1987). In general, the attribution of intelligence is taken as the ability to address wider issues than the literal meaning of a user's command or action and involves the use of knowledge-based techniques (Jones & Virvou, 1991).

The knowledge bases required by an intelligent help system concern the application itself, the user and help strategies for what the the user needs to be told. As well as some representation of the objects and available actions within an application, the application or user knowledge bases may also contain information concerning possible goals and plans, which may encompass a number of commands or actions. These plans may be predetermined or generated in real time by a planning component. The user model will typically model transient and persistent user knowledge, user misconceptions and attributes such as whether the user tends towards serial or holistic learning, for example. The help strategy knowledge base is responsible for providing information which guides the help system's side of the dialogue. The decision as to what to say and when to say it is determined partly as a result of the contents of this structure.

The provision of intelligent help is not only beset with a large number of issues and problems but is further confounded by the fact that very few of these can be tackled in isolation from the others. Most investigations of intelligent help address only a few aspects in the overall scope of the possible number of questions and do so in a restricted interaction context and there is little in the way of a generalised theory of intelligent help or exhaustive empirical data (Carroll & McKendree, 1987). As noted above, it is generally acknowledged that an intelligent help system needs knowledge of the user in the form of a user model but it is far from clear what user characteristics are appropriate or necessary. Would it, for example, be useful to model users' irascibility levels in order to determine the level of interruption by the help system that would be tolerated? What the system

determines as helpful information may be received by the user as an infuriating interruption. A possible corollary is that all messages are ignored which might lead to a catastrophic error. This might be termed the 'crying wolf' problem.

Models of user knowledge need to address the fact that people do not learn merely by adding facts to a factual database. Learning is an active process wherein hypotheses are tested and revisions made on the basis of exploration. Prior experience is also brought to bear on the interpreting of novel situations (Carroll & Mack, 1984). More generally, it may not be possible to determine whether normative or stereotypical user models are adequate, except in relation to a particular help system function. The various issues in question have a great tendency towards interdependence.

There are often interpretational problems in modelling user intentions, plans and goals, owing to non-contiguity, non-linearity and ambiguity in authentic plan execution (Finin, 1983). Users will often change, suspend or abandon plans in mid flow and the relationship between plans and sequences of action is often many to many. Even human observers cannot always correctly interpret user actions, with the enormously greater interpretational facilities they have over any conceivable automatic system. The problem is exacerbated in situations where the task space is large and open-ended.

The *active/passive* distinction is itself perhaps oversimplified. Once help information has been presented there may well be a need to test its effectiveness or to allow the user to clarify or extend certain points. Studies of human help dialogues have found that users often propose answers to their problems by way of a request for help (Aarronson & Carroll, 1986). If these studies are relevant to the HCI context then a mixed initiative dialogue structure is suggested. Similarly, the level of control exerted by the system over the interaction has to be considered. A help system may merely advise or it may block further interaction until the user conforms to some recommendation or possibly auto-correct user errors. Which of these strategies is appropriate and in which situations has not been

determined. The timing of help provision is also a critical factor, given the dynamic nature of human action (*ibid.*).

On the positive side, it is reasonably clear that help systems which respond immediately to user errors are most effective, since the users attention is focussed on the topic concerned in the help message and the correction is closely attached to the decision point (*ibid.*). Users tend to ignore information which does not seem to them relevant to their current concerns. Thus it is also better to present information in terms which relate it to the user's current task or action. Users learn to recognise dead end situations if they have been allowed to encounter them, conforming to the exploratory characterisation of learning. They also appear to learn more effectively by pursuing their own goals rather than a sequence of instructions, also suggesting that help information should address the user's current task (Carroll & Mack, 1984). Of course, this will tend to limit users' mastery of the functionality of a system to that pertaining to their current goals and plans. The aim of expanding user expertise beyond the scope of users' current goals and plans is an important one but one which lies more within the bounds of tutoring than assistance.

5.8 Examples of Intelligent Help Systems

UNIX® Consultant (Wilensky *et. al.*, 1988) is a passive help system which accepts natural language queries from users and generates natural language responses. It is intended to help users learn about UNIX® concepts and functionality via these user-initiated dialogues. Users' plans and goals are identified from the user query and by reference to a static stereotype model of the user as a novice, beginner, intermediate or expert. UNIX® concepts are ranked in terms of relative difficulty and thus each classification is supposed to be familiar with a certain class of concepts. Conflicting evidence of user classification can override inheritance of class concepts. UNIX® Consultant (UC) operates separately from the user's principle interaction and thus cannot relate its responses to specific states or conditions in which users might find themselves. UC cannot therefore take

into consideration any particular contextual feature which may militate against the advice it presents. It can only generalise from the rules governing UNIX® usage. Each query is treated in isolation so that a dialogue consists of a query and a single response, without any reference being possible to a previous dialogue. The main thrust of UC has been to be able to handle user queries in a variety of syntactic forms and sentence types and in interpreting indirect speech acts embodied in such queries as 'do you know...', where there is an implicit request to be given the ellipted information.

AQUA (Quilici *et. al.*, 1988), also based on UNIX®, is intended to detect and correct users' plan-based misconceptions. The idea is not merely get the user out of trouble but to correct the belief which led to the difficulty. AQUA assumes a description of the user's problem in terms of what action was being attempted, what went wrong and any hypotheses the user might have about the error, so AQUA is a passive help system. AQUA utilises knowledge of plans and goals and typical user misconceptions in order to find a user belief about the system which it does not share. It then determines why it does not hold that belief and this reason is used in the explanation to the user. Rules of the system are explicitly represented in terms of what actions achieve, what states enable, what actions cause by way of side effects and what states preclude and so AQUA can address state and function-based user misconceptions. The user modelling component transforms the problem specification into a set of propositions representing what the users relevant beliefs might be. One or more of these is identified as being a belief that AQUA does not hold and the reason for this used in the advice preparation. A focussing mechanism, based on explicit knowledge about plan failure, is used where possible to limit the set of beliefs which AQUA has to consider from its own knowledge base. For example, if the user quotes an error message, then this can be used to index certain of AQUA's beliefs as relevant. If these beliefs do not provide an explanation, the whole knowledge base has to be considered.

Eurohelp (Breuker, 1988) is both an active and passive help system

designed with the aim of providing immediate assistance in a manner appropriate to the type of user, rather than that of performing sophisticated reasoning about the users' conceptual model. To date, the active component has been applied to UNIX® mail and the editor Vi. The performance monitor scrutinises sequences of user actions in order to diagnose incorrect or inefficient plans. User commands are checked for legality and usefulness and, if passed, compared with information from a plan library to determine if they fit into a recognised and optimum plan sequence. Eurohelp utilises an *overlay* user model consisting of a subset of the plan hierarchy which the user is considered to be familiar with. This information can be used to limit the search space of possible plans the user might be pursuing. The goal of the user's plan picks out an optimum plan and this is compared with the user plan to determine whether the user's plan is optimised. An illegal or useless command or one which does not fit into a recognisable plan, or an inefficient plan triggers Eurohelp's advisor which attempts to diagnose the problem as either lexical, syntactic, semantic, mode, catastrophic or planning error and advice is generated accordingly (Erlandsen & Holm, 1987). Semantic error concerns the existence of objects referred to by a command but it is not clear how Eurohelp treats other contextual factors. The emphasis here is on plan-based errors.

A number of projects pursue a plan recognition based format. SINIX (Hecking, 1987) is another UNIX® active help system with similar plan-based facilities to Eurohelp, SUSI (Jerrams-Smith, 1985) bases similar help facilities on a wide range of *typical* UNIX® user errors and so uses no systematic error classification. FITS-2 (Woodroffe, 1988), PRIAM (Davenport & Weir, 1986) both use similar plan recognition techniques in order to identify user plans and goals. REASON (Prager *et. al.*, 1990) is an intelligent assistant which is designed to address a number of different kinds of user error. An inference engine dynamically generates suggestions about what the user might have intended when an error is detected.

Context errors of the kind central to the current discussion are one of

the error types dealt with by REASON. However, REASON has no mechanism for deciding between causes of error or diagnosing user misconceptions. All possible causes are equally valid and information regarding a number of possible meanings might be presented to the user. The view presented here is that this contravenes the communicative principles of conciseness and perspicuity (i.e. the maxims of quantity and manner) and that at least an attempt should be made to give a precise diagnosis. Other user support systems address the issue of context from different perspectives. The use of navigational aids for Hypertext uses features such as context-sensitive graphical presentation variation, history traces, overviews, time stamps and footprints (Nielsen, 1990). Some tasks have the property of 'cognitive viscosity' (Young, 1991) where some alteration has implications for other parts of the system. For example, changing a variable name in BASIC means that all instances of that variable need to be updated. Viewdata editing can have the same consequences since pages within the system may be related. If information on one page is altered, it may entail updating for others.

Young and Harris (1986) report on the design of an assistant which maintains and displays a list of implications, based on the edits performed with such a viewdata editor and thus reduces the load on the user's short term memory. The assistant also orders these tasks for convenience. Mode error differs from context error (as defined here) in that it is a semantic, rather than pragmatic, phenomenon as can be discerned by the use of the phrase *mode ambiguity*. A situation of mode ambiguity arises when an action has more than one meaning depending on the state of the system. Context errors are not caused by multiple meaning. Mode errors are therefore not seen by the system as errors at all. They are errors in that the user and the system assign different meanings to a token because of the mistaken belief concerning the system state. Mode errors can be reduced by removing mode ambiguity either by increasing the 'width' of the interface or restructuring the functionality of the system. The problem may also be addressed by the use of mode signalling techniques (Monk, 1986)

5.9 The Design of an Intelligent Context Error Assistant

Jackson and Lefrere (1984) suggest that an explicit representation of user intentions is required for intelligent help and that the best possible context for interpreting user actions is the current state of the user's plan. They offer a rule-based approach to this endeavour, based on a view of effective interaction being determined by 'communicative competence' characterised as the 'appropriateness of expressions in situations'. These views find some sympathy here. However, while Jackson and Lefrere address the business of interpreting user actions with reference to the context of the user's plan, it should be stressed that the ascription of intention can only be an hypothesis and is entirely bound up with the ascription of belief and the assignment of interpretations to expressions.

In other words, assigning interpretations to another's utterances and ascribing beliefs and intentions are parts in a single project. One cannot complete one part without performing the other. (Davidson, 1984: p.127). Someone who drinks a glass of water may be interpreted as believing the glass to contain water and being thirsty or believing the glass to contain poison and being suicidal, along with any number of other interpretations. The action cannot be made sense of without reference to intentions and beliefs. Thus to form hypotheses about intended meanings by making assumptions about non-linguistic intentions is just to choose a particular interpretive strategy. The strategy adopted here is to ascribe beliefs by making assumptions concerning intentions. The assumptions made are that the expressions used can be assigned literal interpretations and that their perlocutionary force (Austin, 1962) reflects the intentions of the user/speaker. The non-linguistic context interacts with that presupposed by the utterance to allow inferences to be made about beliefs concerning that context. The reader is invited to compare this linguistic description of the process with that given in terms of mental models and possible worlds in Chapter four.

Another aspect of Jackson and Lefrere's approach which finds favour

here is the use of rule-based techniques. As they point out, although large rule-based systems present problems of control, they also offer great flexibility, both in being less context-bound than higher level knowledge representations such as scripts and frames and in being construable variously as descriptions of expert knowledge or system rules, causal explanations or plan generators. We have earlier described the problem of context error in terms of pre- and post-conditions on user actions and so the rule-based approach seems most appropriate for the representation of this kind of knowledge. There is also the prospect of being able to use this knowledge base both for the diagnosis of user misconception and for the provision of help information on user errors, although the latter aspect is a non-trivial issue and is not considered central to the present work.

The help system under consideration requires access to two basic kinds of knowledge: knowledge of the system state and knowledge of the system 'rules', in terms of the requirements imposed on the system state by commands issued by the user and the effects or changes which successful commands achieve; the pre- and post-conditions of commands. It also has to know how to interpret commands. A command can be seen as a speech act conveying an attempt to change a state of affairs, to make some proposition true. In the context of a command language file system, this may be an attempt to alter the system state in terms of the file and directory structure or attributes or to alter the state of the display in some way, as when a directory is listed or the current working directory (*cwd*) identified.

There are a number of formalisms which might be employed for the model of this knowledge; predicate calculus for the modelling of system state and production rules or Horn clauses for the modelling of system rules. However, the logic programming language Prolog provides a convenient notation for both these models as well as providing a way of expressing the altering of states of affairs via the *assert* and *retract* functions. Thus a command can be treated as an attempt to *assert* a particular proposition (or perform a *write* to the screen) but while Prolog

itself will allow any legal proposition to be asserted, our sub-system will impose the conditions applicable to the UNIX® file system on what can in general be asserted, how it must be asserted and what other propositions must also hold for a particular assertion to succeed. Throughout the following discussion, therefore, the Prolog notation will be used as a formalism for the knowledge domain in question. English renditions of Prolog clauses are provided for the reader unfamiliar with this notation.

Representing the system state presents the simplest problem, since this can be achieved with a set of Prolog *clauses*. For example, the existence and attributes of an extant file can be indicated by an instantiation of the following clause:

file(Name,Mode,Parent).

As a query, this would unify with a 'fact' in the database such as

file(letter,777,letdir).

This clause represents the situation that there is a file called 'letter' in the directory 'letdir' which is not read, write or execute protected for any user. Prolog uses names beginning with upper case letters as variables and those beginning with lower case as constants. By asserting similar clauses for directories, the state of the whole directory structure can be inferred from the links between parent and child. Other clauses will represent such things as the identity of the *cwd*. Of course, the above clause, by the nature of the file system, entails that there is a directory called 'letdir' which might be represented by the clause:

directory(letdir,777,root,[letter]).

This is not the place to consider details of implementation but it should be pointed out that the reciprocation or redundancy involved in showing

'letter' as one of the children (actually, the only child) of 'letdir' as well as showing 'letdir' as the parent of 'letter' is not logically necessary but may be required to reduce processing and thus speed up inferences. However, redundancy has its costs and a valid command indicating deletion of the file 'letter' should result in the removal or *retraction* of the above *file* clause **and** a modification of the *directory* clause to remove 'letter' from the list of children. There is a great deal of freedom and flexibility in the way that knowledge can be structured using such techniques and decisions have to be made on the basis of achieving control objectives and architectural elegance and clarity. However, only issues directly governing the effectiveness of a design in terms of the central aims of this project are considered here.

The rules of the system are a good deal more complicated to express and must include reference both to the preconditions which must pertain for a command to succeed and the effects or consequences of successful and unsuccessful commands. Let us suppose that the UNIX® command is translated into the notation of a Prolog query (a fact presented for unification or pattern matching to the Prolog database). For example, the preconditions relating to changing the current working directory with the command "cd <directory>" might be represented as follows:

```
cd(X):-
    not cwd(X),
    directory(X,_,_,_),
    cwd(Y),
    parent(X,Y); child(X,Y), execute(X).
```

This states that

"cd(X)" is true or succeeds if
X is not already the current working directory
and X is an existing directory
and where Y is the current working directory
X is the parent or child of Y

and X has execute permission...

The satisfaction of these preconditions must lead to the appropriate consequences for that particular command. In this case, that would mean the retraction of the clause relating to the identity of the current working directory and the assertion of a clause to the effect that the directory currently instantiated in the variable X is the new current working directory:

```
retract(cwd(Old)),
assert(cwd(X)).
```

If one of the preconditions fails, then it is important that its characteristics are noted as this constitutes the system's view of the context error and the starting point for the identification of user misconception. For example, if the *cwd* turns out to be neither the parent nor child of the 'target' directory, then it may be assumed that the misconception concerns some aspect of the directory structure, identity of the *cwd* (state misconception) or the rules for changing directory themselves (function misconception). Ideally, it would be desirable to be able to address multiple precondition failures so that all incorrect aspects of a command could be indicated at once, in line with the demands of cooperative communication but, although this ideal should almost be obligatory in any real development, it is not considered imperative in the present research context as its contribution to improved dialogue would seem to be a separable component.

Any failed precondition will, of course, effectively block the consequences or post-conditions of the command. In this example, the *cwd* will not be changed. The failure will also trigger the help system into an attempt to diagnose the misconception lying behind the error. As described in Chapter four, this consists firstly in generating a set of misconception hypotheses and secondly in attempting to cull evidence in support of one of

these over the others. The former involves the generation of models of the system which are compatible with the command. These possible states of the system are not only ones where the unsatisfied preconditions are met but also ones which would otherwise render the command acceptable and, like the unsatisfied preconditions, can be generated using the rule base itself. By substituting variables for constants in the rules governing directory changes, we can generate the instances when the command would succeed. For example, if the argument supplied to the *cd* command (say x) remains the same but the argument for the *cwd* is left open as a variable, then the rule governing directory changes will unify that variable with all of the possible successful combinations of *cwd* with x. Actually, this is an exploitation of the *polymodal* properties of Prolog and is not a feature of rule based systems as such but this use of 'rules' for a number of purposes is as described above.

Suppose then that a set of such counterfactuals is generated. It remains for the assistant to attempt to determine which of these 'distortions' of the actual system state corresponds to the user's mental model and so identify the misconception. As detailed in Chapter four, for state misconceptions, the strategy adopted here is to look for evidence that the user is more likely to believe one scenario over the others. One kind of evidence comes from the log of past commands which is maintained by the assistant system. If the counterfactual system state is found to correspond with a previous system state, then this is counted as evidence for belief that that state pertains. This is motivated both by casual observations of users and by general features of short term memory. The log really represents a record of past system states. In practice, it is easier to maintain a record of state changes in the form of a representation of successful commands than one of previous system states, since the latter would involve a great many more 'facts'. It is a simple matter to check through such a record for a command which altered the system state in the relevant way.

We have noted that a context error can be caused by a misconception concerning system functions as well as system states. The directory change

failure could arise from a misconception about the preconditions applying to the *cd* command. In such cases, it may be adequate merely to state the violated misconception rather than attempting to diagnose the precise nature of the misconception. As the rules of the system are fixed, there is not the possibility of a misconception corresponding to a possible system rule and therefore less available evidence in support of one possible misconception over another. A library of typical misconceptions might be useful for this purpose. Indeed, the user protocols recorded in Chapter 7 tend to indicate typical misconceptions concerning, for example, the construction of pathnames. Neither would the linguistic paradigm rule out the relevance of this kind of knowledge or knowledge of user expertise to the interpretative process. However, these lines of inference are not pursued further in the current project.

As noted in Chapter four, in many cases it may be adequate to provide information regarding the actual system state or rule relevant to the unsatisfied precondition in order to address the misconception, allowing the user to make the necessary connections. There is evidence to support the idea that users' learning can be aided by being encouraged and guided in thinking about the system's workings rather than always being 'led by the nose' (Carroll & Mack, 1984; Black & Carroll, 1987). In some cases, it will not be possible to determine the precise nature of the misconception, as can happen in everyday conversation. The assistant can only help according to the quality of information at its disposal and, on occasion, the only option will be to present information which addresses a number of possible misconceptions, perhaps inquiring first if the user would like assistance.

Where there are a number of competing candidate misconceptions, the heuristics outlined in Chapter four may be called upon. The assistant is unlikely to be able to form misconception hypotheses for any user with a radically distorted view of the system, since there is likely to be little discernible connection between the system, the user's conceptual model and the erroneous command. In such situations, the assistant will be obliged to present information which addresses all likely misconceptions, requiring

the user to identify the appropriate one.

However, the assistant has to treat the user as a (fallible) rational agent in treating the user's commands as rational responses to a combination of intentions and beliefs. Overall, there is a tendency or *entropy* towards rationality in that the more radically someone's beliefs about the world differ from ours, the less likely it is that we will be able to make sense of his utterances. We cope with discrepancies in someone's use of our language so long as we can interpret that use against a body of assumed shared belief. Similarly, the assistant's hypotheses will tend to maximise this shared belief, in that hypotheses concerning user misconceptions will be rated on a scale relating to the amount of distortion from reality they involve. Thus, misconceptions about the identity of the *cwd* will be rated highly since these consist in minor distortions in the system state, whereas changes in the directory structure involve greater degrees of distortion. In this way, logical and psychological proximity (see Chapter four) are the rationalising principles which guide the assistant's reasoning.

The next chapter describes a concrete implementation of these ideas aimed at evaluating their application and effectiveness in live interaction between users and a computer system.

6. Implementation

The implementation of a test bed, prototype interface, as part of a simulation of the UNIX® file system is described in this chapter. This simulation is used for evaluation purposes in the research. The processing which the user support component undertakes on user errors is explained, as are any deviations from the design. Some explanation of the program code found in appendix A is included to enable the reader to refer easily to this if required.

6.0 Introduction

Since the research findings which provided part of the motivation for this project were based on studies of UNIX® use and in order to be able to evaluate the approach advocated here, a simulation of the UNIX® file and directory system was designed, to which could be attached any data logging software, as well as the proposed help system implementation, based on the analysis of context error detailed throughout this text.

Following the rule-based methodology described in Chapter five, all of the software was written in AAIS Prolog on an Apple Macintosh II. Prolog was chosen because it provides a clear and easy way to represent rules and states of the simulated system. It also allows the possibility of generating counterfactual states and rules in line with the possible worlds approach to misconception identification outlined in Chapter four. The code is portable apart from one or two routines which use Macintosh operating system calls in order to disable the mouse pointer and menu bar and read the system time and date. The complete code can be found in Appendix A.

6.1 The UNIX® File System

UNIX® has a hierarchical directory structure wherein directories are nested to an arbitrary level. Files are always leaves in the directory tree, whereas directories may also be branches. UNIX® provides a number of commands to allow the user to set up, view and manipulate the directory and file structure to taste. The simulation does not provide all of the UNIX® facilities and omits those applying recursively, most of the command options and restricts copying and moving to files only. It was not necessary to include all of the UNIX® functionality or to follow the UNIX® design slavishly, since evaluation subjects were to be chosen from a novice population who would not be experienced enough to use advanced features.

Certain variations were also included so that any experience of UNIX® would not provide a subject with complete familiarity with the simulation. The copy (*cp*) and move (*mv*) commands were restricted to take only files as arguments and not directories. This was done in order to complicate the users' task of restructuring the file system so that errors would be more likely. The simulation was designed to be complex enough to present a real challenge to novice users, without overwhelming them with an extensive range of complex operations. Thus, in the simulation, there are only ten available commands which are described in the simulation command summary given to subjects in evaluation exercises and reproduced in appendix G.

6.2 The Simulation

At the top level, the software comprises a prompt generator and control section. The control section sets up the environment and calls other program components which perform various functions within the simulation and the help system. The first simulation component is the command parser. The parser is actually comprised of the files 'input', 'service' and all of the '...server' files. The parser has to take the input

stream and divide it into command line components. Since the application of rewrite rules are restricted by the presence of certain symbols, the grammar of the command line is context sensitive and can be formally represented as follows:

commandline --> lone
commandline --> single + " " + path
commandline --> double + " " + path + " " + path
commandline --> "chmod" + " " + mode + " " + object
path --> filename | dirname | dirname + "/" + path
object --> filename | dirname
filename --> string
dirname --> string
string --> alphanum | alphanum + string
alphanum --> {"a"-"z","0"-"9"}
lone --> {"ls", "ls -l", "cd", "pwd"}
single --> {"ls","ls -l", "cd", "mkfile", "mkdir", "rm", "rmdir"}
mode --> {"000","100","200","300","400","500","600","700"}
double --> {"cp","mv"}

The parser turns command lines into Prolog lists so any paths can be processed recursively until a target object is identified at the end of the path. The command is processed by the **validcomm** module which identifies it as valid or not. If the command is valid the rest of the command line is passed to individual modules dealing with arguments relating to that command. Facts for error type and erroneous arguments are asserted by each processing routine if an error is encountered. The possible error types have been identified as follows in accordance with the error typology detailed in Chapter three.

6.3 UNIX® Error Classification

Lexical error - The command word itself is not recognised. Lexical errors cannot apply to object names as these are user defined. Mis-spellings and typing errors may show up as syntax errors or reference failures.

Syntactic error - The command line does not follow the correct syntax of the above grammar. Too many or not enough spaces and incorrect path separators fall into this category.

Logical or Semantic error - The action specified does not 'make sense'. Attempts to list a file with the "ls" command or remove a directory with the "rm" command are of this type.

Context or pragmatic error - The system is not in a state for this otherwise correct command to succeed. These fall into four sub-categories:

Reference failure - the specified object does not exist in the specified directory.

Privilege failure - the specified object is protected from the specified operation by its current mode value setting.

Non-empty - an attempt to delete a non-empty directory has been made.

Duplication - all object names must be unique. An attempt to create a duplicate name gives this error.

6.4 The Database

The database that the simulation maintains does not include 'real' files corresponding to those manipulated by the user. Instead, the directory structure is simulated as a set of Prolog facts. This is updated as specified by each successful command from the user. All of the user's commands are time stamped via a Macintosh operating system call and recorded as Prolog facts and this part of the database is automatically written to disk when the user logs out, along with the directory structure clauses, counts for the different types of error committed and elapsed time since the user logged

on. These log files may then be scrutinised at a later time or could be used to replay the user's whole interaction if required, although this was not considered necessary for the planned evaluation exercises. Erroneous commands are marked in the database with their error type, the current working directory and the arguments which caused the error. Again, this aids in the later analysis of errors.

6.5 Error Analysis

After each command line has been processed and the command recorded, the **analyser** module is called. If an error has been signalled, the analyser first checks the kind of error. If the error is of the type falling under the class of context error, then the analyser scrutinises the interaction log, via modules dedicated to finding evidence of particular kinds, for a past action which might constitute user justification for believing the erroneous command to be compatible with the system and its current configuration.

In theory, the analyser should generate a set of total system states (possible worlds) compatible with the erroneous command and then select from these candidates for the user's mental model of the system, using the principle of conservation or logical proximity (preferring states most similar to the actual state) and looking for evidence that might indicate one over the others. In practice, a heuristic is applied to lessen the amount of processing required to produce a misconception diagnosis. This implementation of the process of analysis is still as strong as the analysis criteria as they stand. Any development of the criteria might, of course, require modifications to or perhaps even abandonment of these heuristics.

Any past action whose undoing would rehabilitate the erroneous command is taken as evidence for the hypothesis that that action's effects have not registered in the user's mental model. This evidence thus provides support for the idea that the user's mental model corresponds to the possible world in which the effects of that action are negated. In order for the state to change, there must have been a user action which changed it and

this action will be in the interaction log. Thus this action can be found by the analyser and determined as the (probable) causal antecedent of the error indicating that the user's mental model of the system is out of correspondence with reality in this respect.

An output routine notifies the user of the mismatch and the action which it deems to be the likely cause, quoting the actual arguments used in the erroneous and causally antecedent commands. This is important in three respects. Firstly, it presents error messages in terms the user is familiar with i.e. the names of the objects being manipulated, rather than in system terminology abstracted from those objects e.g. 'pathname'. Secondly, the user is informed of the connection between the antecedent action and the current state, thus demonstrating or reinforcing a correct device model by illustrating the relation between the effects of one command and the preconditions of another. Thirdly, this 'tutorial' information is presented at a time when the user is focussed on its subject matter, in actually attempting a task which it relates to.

The analyser does not currently address misconceptions concerning system rules or those having no relation to some causal antecedent user action. The help system would no doubt be strengthened considerably by these additions. A fully developed version of the help system would also eschew the heuristics used here and follow the design more closely in generating system state hypotheses and using the stated selection criteria. However, this has not been possible owing to time constraints.

6.6 Error Messages

The error messages that the help system (QDOS) generates are designed on the basis of a three part strategy. This is intended to compensate for the fact that rule-based errors are not addressed in this implementation and to support the conceptual information with a little of the procedural kind. Thus, a context error will first be signalled via the standard UNIX® error message, e.g. 'file not found'. Then the first part of the QDOS error messaging begins with a statement of the rule that is involved in the error,

e.g. in the case of an attempt to duplicate a filename, 'all names must be unique'. Next, specific information about the error is given using the object names involved along with details of the antecedent action which caused the system state to be such that the current command failed. Lastly, where possible, advice is given, again in specific terms, as to how the command might be rehabilitated. This set of messages corresponds with the 'blanket' response to suspected misconception described in Chapter four. Ideally, the specific state or rule based misconception would be determined, if possible, and extraneous information omitted. If users are burdened with material superfluous to their specific needs, they will tend not to rely on its provider and may dismiss useful information along with that which they consider irrelevant (Carroll & Aaronson, 1988).

6.7 The Program Modules

What follows is a short description of the main components of the UNIX® simulation and help system QDOS. The code itself is reproduced in Appendix A. Figures 6.1 and 6.2 illustrate, respectively, the program structure hierarchy and control flow.

Qdos is the main calling module. It loads (reconsults) all of the other files into the Prolog database and controls the overall flow of the program via a 'repeat' call to the main calling clause.

Startup sets all of the initial values for the interaction. All error counts and time values are initialised and the startup directory structure is set up. The events which created this structure are also asserted so that a pre-interaction history is present to explain the existence of the starting state, should a context error occur involving one of these actions. The analyser then has actions occurring prior to the experimental subjects' interaction to draw on as explanations of current states.

Output contains all of the clauses which provide UNIX® like feedback in the way of directory listings in long and short format and responses to the *pwd* command. UNIX® numeric modes are also converted to symbolic modes here for long lists, e.g. '700' --> 'rwx'.

Mylib.h contains the definitions of Prolog clauses, which Qdos uses, and which bind to the Macintosh operating system and toolbox.

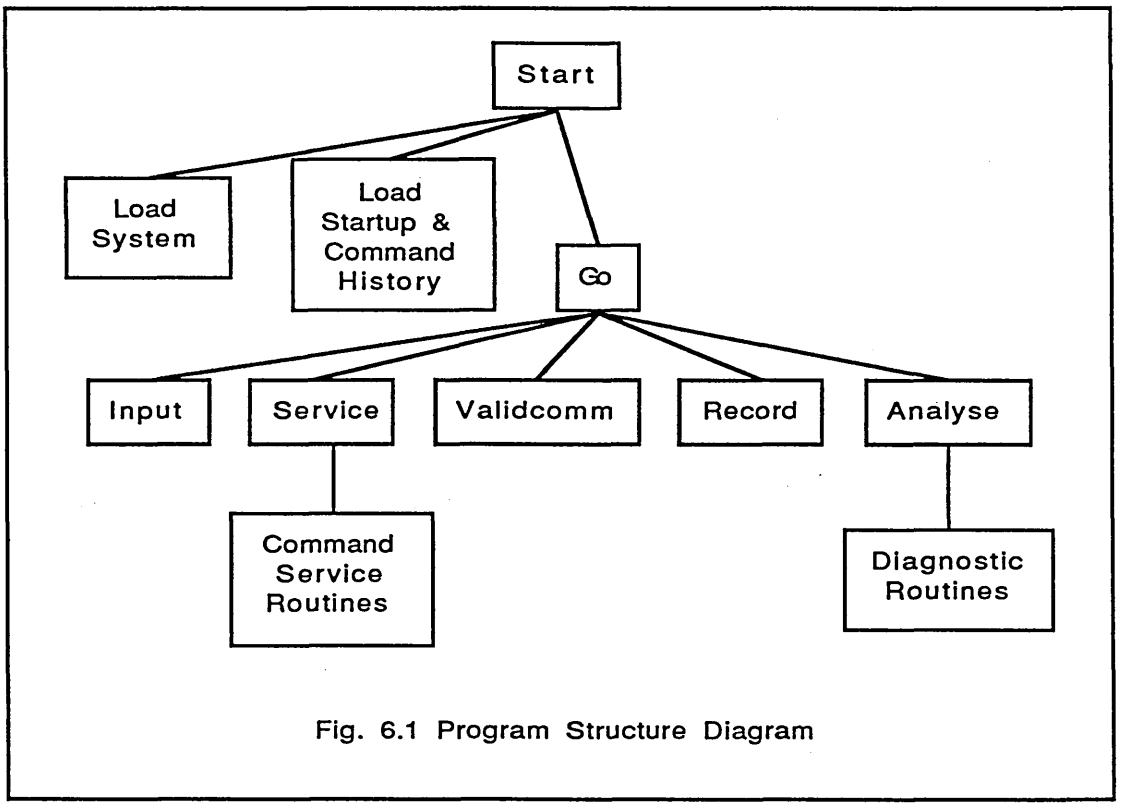


Fig. 6.1 Program Structure Diagram

Qdos uses only a fraction of the available system calls so that it would be wasteful of memory and processing resources to have the full complement of toolbox definitions resident in the database.

Utils contains a number of generic routines used to add and delete items from lists and to check if an object is another's ancestor in the directory structure.

Analyse first checks if an error condition has been asserted. If there is an error, it checks if the error is in the set designated as pragmatic and if it is, it determines the kind of context error and calls other routines which look through the interaction log for evidence associated with that particular error type. A reference error results in a call to **lastrmvcheck** and **lastcdcheck**. A read, write or execute violation results in a call to **lastreadcheck, lastwritecheck** and **lastexecheck** respectively. A duplication error results in a call to **lastmkmvcpcheck** and a nonempty error results in a call to **nonemptyprecond**. All of the above calls are

followed by a call to an appropriate **precond** routine which delivers information to the user concerning the precondition associated with the error.

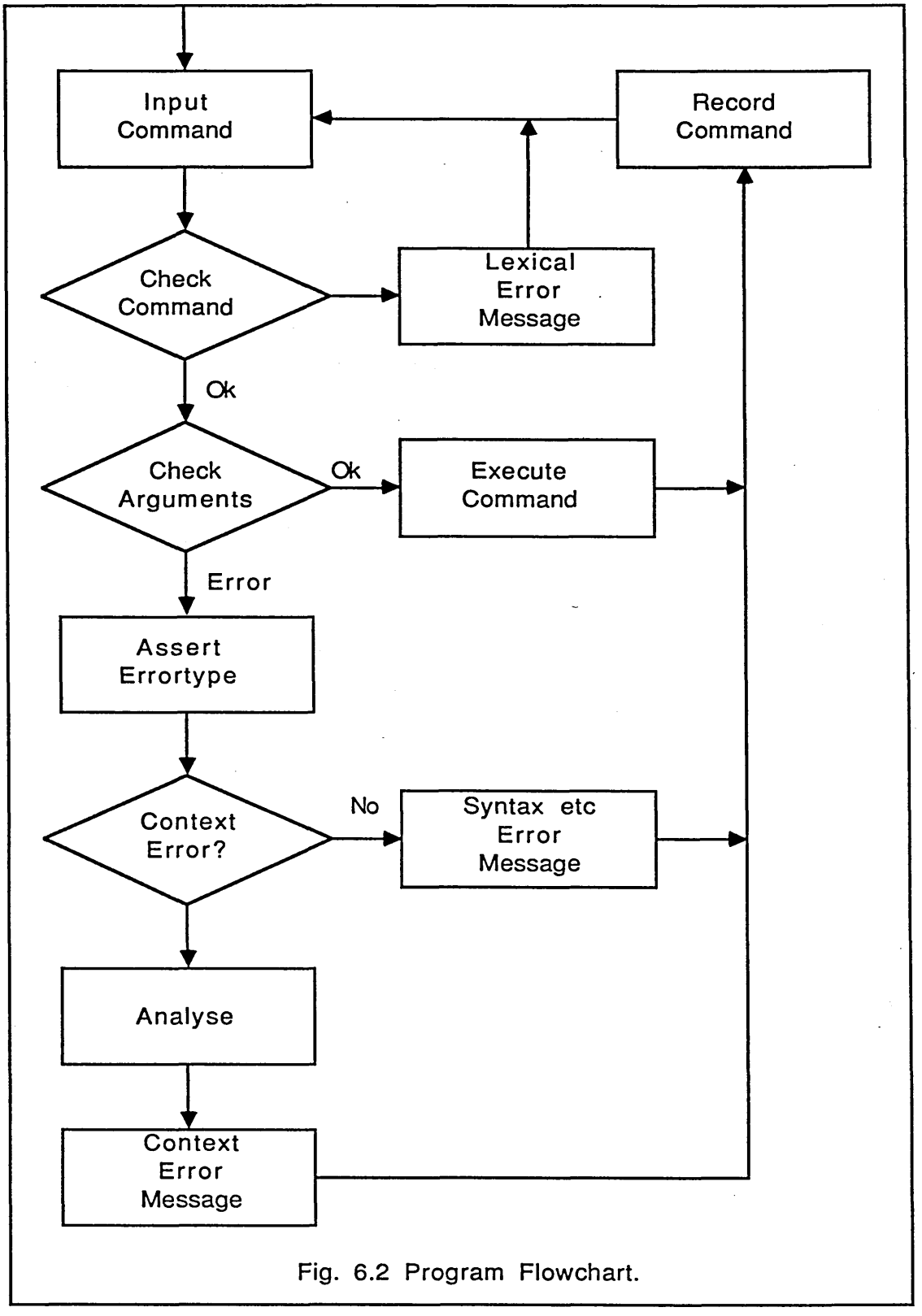


Fig. 6.2 Program Flowchart.

appearance sets the window title to 'QDOS', disables the mouse and clears the menu bar, thus disguising the AAIS Prolog display screen.

cd processes the UNIX® *cd* command. The command is checked for syntax errors and if there are any, a syntax error is asserted and processing stops. If no argument is provided, the *cwd* is changed to 'home'. If a path is supplied, it is traced and checks are made that each item being a directory and that each is a child or parent of its predecessor. If the path is faulty, a reference error is asserted with the identities of the problematic arguments. If an item in the path is a file, a logical error is asserted and a brief error message to this effect is output. If all is well, **change** is called.

change first checks the access mode on the target directory and asserts an execute error if it is protected. If not, the target is asserted as the new *cwd* and this change is recorded in the database for quick reference by the **analyser.**

chmod processes the *chmod* command. It first checks the command for syntax errors before testing the mode argument for validity (000 - 700). The path is traced, checked and errors flagged as in **cd** and, if all is well, the mode setting is altered for the target item. This is unless the item is the root or home directory. In that case, the information is provided that the user is not the owner of these directories and cannot therefore alter their access mode settings. This is discounted as a context error, since it involves system characteristics which are beyond the control of the user.

cp processes the *cp* command. The syntax and path references for source and target arguments are checked in the manner described above and appropriate errors flagged if necessary. A logical error is flagged if the source target is a directory, since this implementation only allows the copying of files. An existing file of the same name is overwritten (actually no change is necessary) if it is in the target directory. A duplication error is signalled. Duplication is also indicated if a directory of the same name as the destination target name exists. Access violations are also checked for and recorded if discovered. All being well, the records are altered to

reflect the existence of the new file.

Input deals with the feeding of command lines into the simulation. Characters are read from the keyboard until an argument separator (space or "/") or commandline terminator (return) is encountered. The resulting lists of characters are transformed into a list of atoms which the other processing routines treat as data. The form of these commandline lists is observed in the log files presented in appendices I and J.

lastcdcheck is used in the analysis of reference errors. Its function is to check the contents of the previous working directory for the presence of the erroneous item in a reference error. It is thus testing the hypothesis that the user has forgotten the last directory change or does not know the effects of the *cd* command and therefore holds the misconception that the previous working directory is the current working directory. This hypothesis is one which would inevitably arise as a possible world, under the criterion of logical and psychological proximity. The alteration to the system state is minimal and the erroneous command would be acceptable had the last directory change not occurred. If the test is positive, the hypothesis is taken to be proved and a reminder of the directory change and its effects is presented to the user.

lastexecheck is also used in the analysis of errors. It searches the user log for an instance of the *chmod* command which had the erroneous argument as its target. If found, such an occurrence is taken as evidence for a state based misconception involving the state wherein that command had not been invoked. In this case the criteria of logical and psychological proximity are less reliable, owing to the less obvious connections between the setting of execute protection and the actions which are consequently outlawed. The possibility of the cause of the error being a rule based misconception, even given evidence under the stated criteria, is not as remote as in other cases of context error.

lastmkmvcpcheck is used in the analysis of duplication errors. It searches the user log for instances of *mkfile, mkdir, mv* and *cp* command whose argument matches the erroneous argument in the duplication error.

Again, a positive test is taken as evidence for a state-based misconception and a reminder of the causal antecedent action presented. If no such antecedent is found, state and rule-based information is generated.

lastreadcheck and **lastwritecheck** operate in the same fashion as **lastexecheck.**

lastrmvcheck is utilised in the analysis of reference errors. The user log is searched for the presence of an instance of the *rm*, *rmdir* or *mv* command whose argument matches that of the erroneous command in the reference error. If such an occurrence is found, this is taken as evidence for a state-based misconception involving the state which would have obtained had that command not occurred. Otherwise, rule-based information, concerning the preconditions of the failed command are presented.

ls -l processes the *ls* command in its 'long' option format. The usual syntax, access and path reference checks are made and if no errors are found the target directory contents are listed with their access modes, owner and size attributes.

ls is essentially the same as **ls -l** but in this case only the names of the target directory contents are displayed.

mkdir services the *mkdir* command. Syntax, reference, access and duplication checks are made and appropriate error attributes asserted if any context errors are found. If the command is sound facts are added to the Prolog database representing the addition of the new directory.

mkfile processes the *mkfile* command, which does not actually exist in UNIX® but which was created to enable files to be created without the use of an editor or other application program. Its function is largely the same as *mkdir* except that its targets are files instead of directories.

rperm, wperm and **xperm** are the clauses which check file and directory command arguments for read, write and execute protection. If arguments fail any of these checks, the error characteristics are asserted and the **modefail** predicate is set to true to signal other modules to cease processing the command.

mv processes the *mv* command. Syntax, reference, duplication and access tests are made on the arguments, which may only be files in this implementation. Errors are signalled to the analyser in the usual way. If no errors are found the relevant facts are retracted and asserted in the Prolog database, representing the displacement of an item and its renaming where necessary.

writelist, listconts and **longlistconts** provide output routines for the UNIX® *ls* commands and error messages which require the output of data stored in Prolog list format.

transmode translates UNIX® numeric mode formats into character based codes for output in response to the *ls -l* command (e.g. 400-->r-x).

precond provides output of general information relating to the preconditions of particular commands. If the analyser decides that a hypothesis for a state-based misconception cannot be supported, the possibility of a rule or function-based misconception is suggested. For example, if no evidence is available in the user log to support the hypothesis that the user holds the misconception that a directory is empty when it is not, then the possibility that the precondition of emptiness on the *rmdir* command is unknown to the user is met with information to this effect.

pwd serves the *pwd* command and outputs the path from the root directory to the current working directory listing every directory node on the way (e.g. root/home/mail/letters).

start is the main routine in the suite of Prolog modules. It reconsults (loads) all the other modules and sets up the QDOS environment detailed in the **startup** module, zeroing all counts, setting the start time, looping through the main calling sequence of the program and supplying the prompts.

record is the module which provides all of the logging facilities in the simulation. All user commands are asserted into the Prolog database in the form of **event** or **nonevent** clauses. Error counts, final directory configurations and statistical data is also recorded here. When the user logs

out of the system these clauses are written to disc and are presented in Appendices I and J.

rmdir processes the *rmdir* command. checks are made for syntax, reference and access errors and also for logical errors such as the target being a file and that the target is an empty directory. Error characteristics are asserted appropriately if one is found and if not, the facts representing the existence of the target directory are retracted.

rm is essentially the same as **rmdir** but its target must be a file.

service, validcomm and **command** form part of the command line parser. They essentially check that the command itself is recognised and call the appropriate service routines which process the arguments to the command. The scope of context adopted in the general approach does not allow the occurrence of alien commands to be considered as context errors and so any errors in commands are flagged as lexical errors.

startup is a file containing all of the environmental settings to be loaded when the user logs on to the system. This includes the initial directory structure (see Appendix D) and the **events** which brought it about so as to provide an interaction history on which the analyser can work to generate error messages relating to causal antecedent actions as soon as the user starts the task. This allows the effects of previous actions to be related to the current error situations and is intended to strengthen the user's understanding of the causal structures inherent in the system's functionality.

tellkids was intended to maintain consistent reference to a directory name in all its child records if the directory name was altered. Once the decision was made to disallow directories as arguments to the *mv* command, this turned out to be unnecessary.

getdatetime ascertains the system date and time via a operating system call. This value is converted to 12-hour and day-date-month-year format and is used in error messages and for timestamping the user log records.

timelapse calculates the elapsed time between two given times. It is used to calculate the total time spent logged in by any user and in error

messages which state how long ago some change was made to the directory structure, which is intended to provide users with some chronological orientation within the error situation.

We next move on to a discussion of evaluation techniques and a description of the evaluative exercises conducted within this research using the QDOS system described above.

7. Evaluation

This chapter deals with interface evaluation techniques and their relative merits. The importance of evaluation is identified and three basic types of evaluation are discussed. The evaluative methods and exercises conducted within the project are described and results presented. Conclusions and observations on these results serve as an appraisal of the success of the approach adopted and its application to the aspect of interaction studied.

7.0 Introduction

The outcome of good interface design is the production of systems which are easy to learn, pleasant to use and which facilitate the completion of user tasks with the least effort. However, the task of translating these *desiderata* into specific interface characteristics is far from simple, as is that of extrapolating from specific characteristics to estimates of usability. Thus an important part of the design process is the evaluation of design decisions. Evaluative measures can be undertaken at any point within the system life-cycle but it is clear that the earlier that design decisions can be evaluated the better, since the costs of altering a design increase at each stage of the product life-cycle.

Without design criteria and evaluative techniques, it is all too easy for designers to succumb to common pitfalls. These include the reliance on 'common sense' which although useful, can lead to design errors. For example, common sense would probably not lead one to suppose that speech recognition in noisy environments could be improved by the use of ear plugs, yet this is in fact the case. Another possible mistake designers

can make is to assume that they are somehow typical or representative of the population of users. This is always false. Designers have a particular and specialised view of the system not normally available to users.

Design 'traditions', appeals to authority and unanalysed assumptions about the user population or context of use can also mean that design faults are introduced. A system 'beep' may not be audible in a noisy environment or by hearing-impaired users. Sometimes what seems like a 'bright idea' will become a design decision without proper consideration of its implications or testing of its consequences and in general, evaluation may be postponed for convenience, making design alterations all the more difficult and costly when they are eventually recognised as being necessary.

Even when evaluation is undertaken it is important that representative samples and contexts are utilised and that the experimental data gathered is analysable to give meaningful measures of usability within the parameters of the evaluation technique used (Downton, 1991). Evaluation techniques fall broadly into three categories. These are formal methods, 'classical' experiments and observational methods.

7.1 Formal methods

Formal methods offer the potential to evaluate designs from the very earliest point in the design process since they do not necessarily require an implementation of the design to work with. The great problem with formal methods is their relative remoteness from the context of use and the vagaries of human performance, mostly assuming expert knowledge or performance, a factor which is bound to reduce the degree of confidence to be held in their results. Few approaches attempt to cope with individual differences in learning styles or cognitive strategies and user motivation (Wilson *et. al.*, 1988). However, they can enable many kinds of design faults to be 'weeded out' at an early stage.

In evaluating an interface design the concern is with the demands, in terms of knowledge requirements, cognitive processing and physical action, which the design places on the user in performing tasks and the level of

support that the design presents to the user. Any formal method used in evaluation will therefore adopt some model of human cognitive characteristics in order to predict these qualities. Thus there is a substantial overlap of concerns between this chapter and chapter four, dealing with user modelling.

Early models follow the linguistic paradigm prevalent in command language interfaces of many early interactive systems. Reisner's 'psychological BNF' (Bachus-Naur Form) (Reisner, 1984) represents cognitive processes and overt actions in an interaction grammar. User performance is taken to be related to the number of terminal strings in the grammar and the overall number of productions required to describe the dialogues. Thus a dialogue with many rules and terminals would be seen as being more difficult to learn and harder to use than one with fewer rules and terminals. This criterion can have useful application but the construction of grammars is a somewhat, arbitrary process and different predictions of usability for the same design can be generated simply by the choice of grammar (Fountain & Norman, 1985; p. 10)

Payne and Green's (1986) Task Action Grammar (TAG) adds to the grammatical model a notion of the relatedness of tasks and the consequent groupings of grammatical rules into sets. This simplifies grammars and is claimed to have more psychological validity in predicting ease of learning, since users are assumed to abstract and make use of similarities between tasks. TAG can reveal design faults related to the concern for consistency, in that simple tasks which share features should be reflected in the design as instances of a more generic task. However, TAG gives the same status to small and large inconsistencies which it may unearth in a system (Booth, 1990c) and has difficulty in defining a *simple task.*

External-Internal Task Mapping Analysis (ETIT) (Moran, 1983) is intended to predict learning complexity and ease of transfer of skills, by mapping tasks in users' terms (external task space) to system orientated (internal) task descriptions. A greater number of mapping rules indicates more complexity and therefore less ease of use. ETIT has not been used 'in

anger' and there is some difficulty in determining a user's 'external task space', apart from abstract specifications or those based on the devices being considered.

The above approaches essentially model the user knowledge or competence required to operate a device as opposed to the mental processing required. In general, these methods tend to suffer the limitations of assuming a simple relationship between knowledge and behaviour and make little reference to cognitive processes of interpretation and problem solving and general limitations on cognitive processes such as memory retrieval. The following approaches attempt to deal more directly with these latter issues.

Card *et al.*'s (1983) Keystroke Level Model makes explicit assumptions about human cognitive processes in order to derive time comparisons based on expert users performing familiar tasks. In this model different human cognitive processes are seen as autonomous facilities with their own characteristics on which performance calculations can be derived. Although the view of human beings as information processors will not do as a general cognitive model, for example, this model does not predict conceptual difficulties, the Keystroke Level Model can guide the construction of expert user time performance models for simple tasks.

Norman's (1986) identification of seven stages of mental activity (goal formation, intention formation, action specification, execution, perception, interpretation and evaluation) does not make quantitative predictions but facilitates the understanding of the consequences of design decisions. For example, menu systems can be seen as supporting goal formation and action specification. In these terms, the overall aim of interface design is seen as facilitating each of the stages of activity as they arise in the course of interaction.

Moran's (1981) Command Language Grammar combines a conceptual and a process model of a system and is intended to elucidate the relationship between the conceptual model of a system and its command language in order to identify conflicts between the two. It has been used

with limited success to predict the incidence of error from the mismatches between conceptual and procedural system models (Davis, 1983b).

GOMS (Goals, Operators, Methods & Selection Rules) (Card *et. al.* 1983) which, like CLG, hierarchically decomposes goals into 'unit tasks', in order to make performance predictions, is a coarser grained version of the Keystroke Level Model. Again, reasonably good estimates are possible but GOMS has difficulty in defining the concept of 'unit task', in that no planning component exists to show how simple tasks are combined into unit tasks, and problems in explaining how selection rules are related to one another and the rest of the analysis (Green *et. al.* 1988).

Kieras and Polson (1985) present a two part formalism intended to represent users' 'how to do it' knowledge and the device itself. The former part is GOMS based and the latter uses generalised transition networks. The aim is to show ease of learning and use and mismatches between the device model and the production system. Rules which belong to more than one method are assumed to require no extra learning overheads and thus to facilitate learning. The depth of the goal stack generated is taken as an indication of ease of use, in terms of the consequent demands on short term memory. This model can provide reasonably accurate predictions of learning and execution times.

7.2 Experimental Methods

Evaluation may be undertaken for the ratification of specific interface designs or in the attempt to discover particular design principles. Lessons may be learned from testing particular designs but often useful principles are 'buried' among the mass of data derived. In any case, the main goal of HCI to date has been the delivery of such principles. The basic methodology of experimental evaluation comes from experimental psychology where a wealth of techniques and experience in empirical evaluation have accrued, to the extent where there are standard procedures for designing and evaluating the results of experimental settings.

The first important decision in experimental design is the selection of

tests, the object of which is to expose the experiment to the most realistic context possible in terms of tasks and environment, at least to the extent of aspects which could reasonably be expected to influence the outcome. Within these terms, it may be desirable to work with the least number of subjects for the least amount of time required to generate significant results.

A *variable* is a factor which may be expected to influence the result. Ideally, an experiment will attempt to observe effects on just one variable in order to provide the simplest statistical model and therefore the most potentially compelling result. In practice, it is rarely possible to achieve this and the aim of realism simultaneously, since everyday situations naturally involve many variables. The variables in which the experimenter has no interest are *nuisance variables* whose effects must be somehow eliminated from the results. Differences between subjects in terms of ability, mood or motivation fall into this category. The use of a *within subjects* design, where each subject is exposed to all test conditions, as opposed to an *inter subject* design, where two or more sets of subjects are exposed to differing conditions, may be used to minimise the effects of some of this kind of variable. However, within subjects designs have their own problems.

Other nuisance variables may be introduced into the experiment with the procedures adopted. Learning effects, fatigue and the like can influence results and so it is important that the experimental procedure is designed to minimise these factors as much as possible by balancing the sequencing of tests across subjects, replicating the experiment with sequences reversed or randomising aspects of the procedure in the expectation that randomness will produce 'even' results. The mere fact of being measured at all can affect a subject's performance; the so called 'Hawthorne' effect.

Supposing that an optimal experimental design has been achieved and the data gathered, it is then necessary to analyse this data in order to detect any significance in them. A variety of methods are available and determining which is the most appropriate method depends upon such

factors as the number of subjects, the number of experimental variables and whether the data can be taken to be representative of a random sample from a *normal distribution.*

To say that some finding is statistically significant is not to say that it is of any practical significance. If some measure of the quality of some interface design or design feature is shown to be statistically significant to some degree it does not necessarily follow that it is a worthwhile option, particularly if it happens to be an expensive one. The major problem suffered by the experimental approach concerns the issue of assessing the generality of results (Monk & Wright, 1991). In practice, the kinds of computer systems and the tasks and problems with which they aid users are large and complex. Tasks may involve many simultaneous and successive cognitive processes and require highly detailed interactive procedures. This has two consequences. Firstly, any of the details of either cognitive processes or interactive procedures, may influence the performance of the others, rendering established research findings concerning individual or small numbers of details dubious in such a complex context. Secondly, measures of the usability of the total system will be of minimal generality, since results will be applicable to only those contexts which are all but identical to the one tested. These factors have been instrumental in the growth in interest and development of the third category of evaluative methodologies.

7.3 Observational Methods

Observation has always been an important, if informal, part of interface evaluation but a growing dissatisfaction with formal and experimental methods, for reasons outlined above, has led to an increasing interest in developing means of assessing usability in context which amount to more than casual observation.

Part of this dissatisfaction also develops from a feeling that the theory or principle being tested is often put at the centre of concerns, rather than the user. This has led to a number of user-centered approaches to

evaluation. As a beginning, one can solicit user responses and opinions via interviews and questionnaires. However, the usefulness of this approach depends upon such factors as the amount of bias or ambiguity inherent in the questions themselves and possible bias in the questionnaire respondents. Questionnaire and interview results cannot be counted as objective data but can be useful as a backup to experimental findings.

A more radical approach to user-centred evaluation, akin to the ethno-methodology of certain anthropologists, eschews all artificial constraints and allows free use in work contexts. The philosophy behind this approach is that the user's experience is central and that all interpretation of it by the observer is invalid. Evaluation is conducted by the user with a 'co-evaluator' and the data are the discussions which ensue (Whiteside *et. al.*, 1987). Such methods, however, do not normally and are not intended to provide generalisable results. Contextuality and generalisability are, as usual, in inverse proportion.

Another radical departure from the 'traditional' view in HCI that designs are arrived at through the application of validated principles stems mainly from two observations. The first of these is that innovation almost invariably precedes the development of theory. The second is that when an 'artifact' is introduced into a work context, it invariably alters that context and this tends to invalidate the design based on the previous context. Any subsequent alterations suffer the same fate and so the whole business of design is necessarily an iterative process; evaluating artifacts in terms of how they support user tasks in the context of use and modifying them accordingly. Any theory inevitably embodied in an artifact arises out of this process rather than preceding it. This kind of position is espoused by Carroll and others (Carroll *et.al.*, 1991). Approaches of this kind, unlike the ethno-methodological account, still carry a minimal amount of psychological 'baggage' in terms, for example, of being structured around Norman's generic task model (Carroll *et. al., op. cit.*).

One of the problems with debriefing interviews with users is that subjects may not accurately remember significant aspects of their

experience. A way around this is to get them to provide a verbal protocol while the interaction is proceeding. In this way, the intentions and events which concern the user most and responses to them will naturally enter into their monologue and the researcher will gain some degree of access to the user's 'live' thought processes. However, it should be added that the advantage of this must be set against the occasional unwillingness of some experienced subjects to share their expertise or inability to explain actions which have been 'internalised' (Diaper, 1989).

Generally, protocol analysis is a method for gaining insight into the psychological processes of an individual engaged in some task or activity. It consists in encouraging the subject to 'think out loud' by way of explanation of the thought processes which underlie and motivate current actions and psychological responses to stimuli received in the course of the activity. Since it is impractical to attempt to analyse these 'monologues' concurrently or recall them for later analysis, it is usual to undertake audio and video recording and sometimes to electronically 'log' user actions and system responses in a computer file.

In addition to the prior request for a verbal protocol, the researcher may prompt the subject from time to time to reveal current mental machinations if the subject should lapse into quiet performance or if a particularly interesting episode should occur. Of course, not all subjects will find it easy to provide such commentaries and the necessity for doing so can easily disrupt task performance. For this reason an alternative method is to have the subject provide a commentary over a recording of their prior performance and to record this commentary. The disadvantage here is that subjects may tend to 'over-rationalise' their previous actions and again not remember their mental machinations accurately enough to be useful.

With either method, another problem can be shyness or reticence on the part of the subject and every effort must be taken to remove any feelings of being under examination (Monk & Wright, 1991). This kind of approach is especially useful for identifying communication breakdown between user

and system. It allows the researcher to focus in on fairly minute aspects of the interaction and to tease out details of miscues and misconceptions. This kind of analysis is extremely laborious and time consuming to perform, since hours of video tape and log records can result from a relatively small number of subjects. Again, results will rarely submit to any significant generalisation, although attempts have been made to identify 'interaction scenarios', which describe typical situations of computer use, as basic units of interaction which do generalise across contexts (Carroll *et.al., op. cit.*).

For the purposes of evaluating the research presented here, it was decided to utilise two approaches. A 'traditional' experiment was devised to try to compare two alternative interface designs and a protocol analysis was conducted so that fine details of interaction might be studied so as to evaluate the hypothesis proposed concerning the cognitive causes of context error.

The software simulation of part of the UNIX® file system described in chapter six was used for the experiment. Although it might have been possible to sample 'real' users, perhaps over a prolonged period, by monitoring them in a live UNIX® environment, it was decided to use a simulation so that greater control could be exercised over the facilities and functions of the system. The experimental variable was the presence or absence of the QDOS context error module in the simulation. At the time when the experiment was conducted, the version of QDOS developed was not capable of reasoning about process-based but only state-based context errors.

The simulation logs user commands so that this record can be scrutinised for causal antecedent actions in context error situations (in theory, the error analyser would look for prior states implicated by the error but in practice, it is easier to record actions which alter states and to look for these actions. This also has the effect of limiting the set of possible worlds considered to those most like the actual world, as described in chapter four). When context errors occur, the help system checks the log for a likely cause. For example, if a file is not found, the help system

checks for deletions, moves or renames of the file and also for its presence in the previous working directory. If such a causal antecedent is found the action is notified to the user as a definite or likely reason for why the current action failed. The simulation also logs all errors, so that an analysis of their relative frequency can be made, and the final state of the file system, so that a measure of task completion can be calculated.

7.4 The Experimental Task

A task was designed for the experimental subjects. The experimental hypothesis being tested was that the experimental group would perform more of the task, in less time and with fewer errors than the control group, owing to the activity of the help system. An aim was to provide a relatively realistic working situation with sufficient task complexity to make the production of errors likely in a comparatively short task duration. Subjects had to take an existing directory and file structure and transform it according to a written specification. This specification is presented in the guise of an informal memorandum and was made intentionally disorderly so that users would have to devise their own task structure in terms of the ordering of tasks. The startup and target file system configurations can be found in Appendices D and E and the printed materials provided to subjects in Appendices C and G.

Pilot trials were conducted with two subjects to validate the experimental design. In any case, the performance results were to be time related to increase performance variability and sensitivity of measure as much as possible.

7.5 Subjects

The subjects were recruited from the third year of a degree course in communication studies. A questionnaire was administered (see Appendix B) to provide information so that two groups could be balanced for relevant experience, the aim being to have two groups with roughly equal experience. Although it would have been preferable to also have some

measure of subjects' aptitude, as there was little opportunity for prior contact with the subjects, balancing of the experimental groups had to be undertaken on the basis of previous experience in using computers, using relevant software applications and typing skills, as obtained from the aforementioned questionnaire. Prior experience was deemed the best available balancing variable to correlate with the independent variable.

In all, twenty subjects were recruited; ten in each group. Unfortunately, only ten of the volunteers actually attended the appointed sessions, a disappointing fact which probably marred the whole exercise. As the experiment was conducted in two sessions, it could not be known how many subjects would attend and so the process was not aborted, in the hope that a reasonable number would eventually attend.

7.6 Experimental Design

It seemed clear that a 'repeated measures' experimental design would not be appropriate for the purposes of this exercise for two reasons; firstly, subjects had little time to spare for the exercise and secondly, it would be unsafe to assume that the considerable learning effects would operate in the same way over the two condition sequences. It also seemed unlikely that an 'independent subjects' experimental design would, with the number of subjects involved, eliminate significant constant errors relating to aptitude and experience among subjects. Thus, the 'matched subjects' design was chosen and the results were organised into two groups of five subjects each in such a way as to balance them for previous experience, as ascertained from questionnaires.

One group (experimental group) attempted the task using the simulation with the help system operational while the other group (the control group) received only UNIX® type error messages. Subjects were given a brief description of the UNIX® file system and its commands before the experiment began and a supervisor was on hand to give advice on the task but not on the system being used. Subjects were unable to give more than about two and a half hours of their time because of other commitments.

7.7 Results and Evaluation

	Ques.	Time (hrs)	Correct Comms	Error	Total Comms	No. Context Errors	% Context Errors	% Alt. Score	Task Perf.	Error Perf.
S1	40	1.35	76	47	123	31	66	85	63	62
S2	30	2.40	163	78	241	47	60	83	35	68
S3	40	1.24	78	89	167	74	83	3	2	47
S7	25	0.70	50	22	72	10	45	1	1	69
S10	15	1.35	25	80	105	55	69	3	2	24
Avge	30	1.4	78	63	142	43	65	35	21	54

fig.1: Individual Results For Helped Group

	Ques.	Time (hrs)	Correct Comms	Errors	Total Comms	No. Context Errors	% Context Errors	% Alt. Score	Task Perf.	Error Perf.
S11	35	2.42	201	124	325	91	73	79	33	62
S12	25	1.58	36	64	100	33	51	0	0	36
S13	20	2.48	100	66	166	39	59	64	26	60
S14	45	1.89	168	145	313	105	72	83	44	54
S15	25	1.69	105	178	283	116	65	24	14	37
Avge	30	2.0	122	115	337	81	64	50	23	50

fig. 2: Individual Results for Non-helped Group

Figs. 1 and 2 above show tables of results derived from the experimental log files in appendix I. The tables show, by group and from left to right, each subject's questionnaire 1 score, total time taken, number of successful commands, number of errors, total number of commands, number of context errors, context errors as a percentage of all error types, alteration score (reflecting the extent to which the task had been completed as a percentage of total task), task performance (alteration score/time taken) and error performance (errors as a percentage of total commands).

The results were evaluated in terms of performance and error counts. A scoring system was devised in order to give a value for the percentage of task achieved for each subject. This was achieved by rating each subtask in terms of its complexity. For example, the sub-task of moving an empty directory attracted a score of one unit, whereas a directory with three files attracted a score of four units. A maximum score could thus be calculated for the whole task and percentages calculated on the basis of the file system configuration recorded by the simulation. A rating of performance was obtained by dividing the percentage of task achieved score by overall time taken. Error performance was calculated as correct commands as a percentage of total commands.

Scores varied widely between subjects in both groups; Standard Deviation for the helped group task and error performance was 27 and 19 respectively and for the 'unhelped' group 17 and 12. This would appear to be accountable to a floor effect in the experiment, in that subjects 3, 7, 10, 12 and (arguably) 15 failed to achieve a significant percentage of the task. If these subjects are dropped from the data and the group scores are recalculated, task performance averages of 49 and 34 are achieved for the helped and unhelped groups respectively (49 and 29 including subject 15); an improvement of 44% for the helped group (or more if subject 15 is to count). No statistical significance can be claimed for this 44%, since there are now only two subjects in one of the groups. However, this difference does not appear to correlate with subjects' experience, as indicated by the first questionnaire, where the helped group showed only a slight experience

advantage. These results indicate that the experimental task was too difficult and that more piloting, with typical subjects, would have been advisable. This fact, along with the disappointing turnout would appear to have undermined what might have been a successful outcome. However, the unavailability of resources for repeating the experiment, both in terms of time and subjects, dictate that a demonstration of the effectiveness of the approach to context error advocated here will have to be achieved by different means.

In any case, error analysis of all subjects reveals that around 65% of all errors were context errors, mostly reference failures, and that 25% of this 65% had significant causal antecedents. Errors constituted over 47% of all attempted commands. In general, subjects found the UNIX® style file system and its command line interface hard to understand and frustrating to use. It is encouraging to note the ubiquity of context error in the results, bearing out other UNIX® studies, although it has yet to be demonstrated that the proposed approach to dealing with it is significantly effective.

7.8 Protocol Analysis

The study was conducted in order to test certain hypotheses concerning the causes of context error. These hypotheses provide part of the justification for the particular approach to dealing with context error proposed in this thesis. A detailed discussion of the hypotheses and the remedial approach adopted has been presented in earlier chapters. Briefly, it is suggested that these errors are caused by mismatches between the system and the user's conceptual model and that they highlight a failure of the interface in aiding the user to build and maintain an appropriate conceptual model in the course of the interaction. It is also suggested that the problem is exacerbated by the interface's failure to meet warranted expectations of communicative cooperation. The study is intended to demonstrate that the kind of approach proposed in this thesis, adapted from linguistic pragmatics, would be effective in determining and remedying user misconceptions about the computer system being used.

7.9 Methodology

For the purposes of this study, a video camera was used for recording both the occurrence of typed commands on the computer monitor screen and the subjects voice, via a lapel microphone. The software in use was a modification of the UNIX® simulation software used in the experimental exercise, designed to record all users' successful and unsuccessful commands as well as noting the error type and some of the contextual features of the error, including its exact time of commission. The video tape was time stamped so that the log files and the video could subsequently be correlated.

In this case, the QDOS context error analysis module was rendered inactive so that subjects would not be pre-empted in their speculations about any errors occurring. The same task and preparation was used as in the experimental setup but questionnaires were not required.

7.10 Subjects

Four subjects took part, two of whom were computer science graduates and all of whom were occasional computer users. All were familiar with the basic notion of hierarchical file systems before the study but none was a recent or regular user of such. Each subject was given a brief pre-task description of the system and a written list of commands and their effects to which reference could be made while completing the task.

After all subjects had been recorded, the video tapes were scrutinised at or near the time points indicated by the log file to be instances of context error. Any comments made by the subject at these times were noted, along with a description of the circumstances of the error and any comments felt to be relevant to the situation. What follows is a set of observations based on these notes, grouped according to similarities in the analysis of the errors. A detailed account of some of the more significant error situations, along with explanations of the way that the proposed help system would interpret them is included from appendix K.

Locational Disorientation:

Subjects often made errors because of misconceptions concerning the identity of the current working directory (*cwd*). Below are some examples taken from appendix K.

S1

nonevent([cd, ' ', persaddr], reference, persaddr, persaddr, persaddr, monday, 20, january, 1992, 18, 16, 34).

Description: 'persaddr' was already the *cwd* when the subject tried to make it so; the classic "please close the window" error.

Commentary: In this case, there is almost certainly a previous directory change which will operate as evidence for the state misconception that the previous *cwd* is the *cwd*. QDOS's response would be to inform the user of the relevant state and the prior action which brought it about, thus correcting the state misconception, reminding the user of the forgotten prior action and indicating the consequences of that action. QDOS's error message would read:

"That command failed because you changed directory from <previous-cwd> to 'persaddr' <timelapse> ago."

S2

nonevent([cd, ' ', addresses], reference, mailin, addresses, mailin, tuesday, 21, january, 1992, 17, 36, 34).

Description: This was an attempt to change the *cwd* to a sibling directory 'addresses' from 'mailin', without supplying the parent as the path. The user said at this point "I think I've lost where I am" indicating that he did was unsure of identity of the *cwd* at this point.

Commentary: The last change of directory from 'home' to 'mailin' would be taken as evidence for the diagnosis of the misconception that 'home' was the *cwd*. The present *cwd* and the location of 'addresses' would thus be deemed as sufficient information to enable the user to recover and correct the misconception. The error message would thus read:

"That command failed because you changed directory from 'home' to 'mailin' <timelapse> ago."

S4

nonevent([cd, ' ', business], reference, home, business, home, tuesday, 11, february, 1992, 18, 36, 36).

Description: The subject attempted to change directory from 'home' to 'business' without supplying the correct path. She was expecting the command 'cd' to find the directory for her, which amounts to a functional misconception concerning the rules for accessing the directory structure.

Commentary: The particular precondition violated is that an item in the path must be a parent or child of its predecessor (the *cwd* being assumed to be the path start). There is no evidence to support the idea that the subject held any particular state misconception and so general forms of correction are indicated. Assuming that the information on relevant system states (locations of *cwd* and target items) and rules for path construction could be assimilated, the problem could be overcome. QDOS's error messages would read:

"The current directory is 'home'."
"The item you apply the command to must exist in the specified directory or in the current directory if you do not specify one."
"Specify the target directory by supplying a path from the current directory to the target directory via all the

intervening directories, i.e. 'mail/business'."

S3 worked almost exclusively from the 'home' directory and therefore was not prone to this misconception. The commandline interface implemented in the simulation does not display the *cwd* name. The user may be able to infer the *cwd* from the command history left on the screen as the display scrolls up but often this information is not absorbed. In the majority of instances, this problem arose from simple memory lapse in that a previous directory change was forgotten but S4 also seemed not to understand that if permission to change directory is denied (because of access code) the current working directory does not change.

Subjects also complained of not being able to remember where items were and what the contents of a particular directory were: S1 19.36.45 - "I can't hold in my head what's in anything at all." As might be expected, problems of maintaining awareness of highly transient or ephemeral system states were exacerbated by necessary attention to sub-tasks, especially if they were unexpectedly encountered. This phenomenon would appear to be accountable to the characteristics of short term memory.

General Misconceptions

All subjects held misconceptions concerning the correct construction of pathnames. These included the belief that it is possible to move 'sideways' in the directory structure, in other words to 'cd' directly to a sibling directory, the assumption that all pathnames start from the 'home' directory instead of the *cwd* and that the current working directory had to be specified as the start of the pathname. All these led to reference errors for which the standard 'x not found' error message was received, where 'x' is the problematic item in the path. This message aided very little in correcting these misconceptions which meant that they had to be corrected verbally by a supervisor in order for the subjects to be able to continue with the task.

S1

nonevent([mv, ' ', mail, (/), business, (/), busout, (/), enquirout, ' ', home, (/), 'temp]'], reference, home, home, home, monday, 20, january, 1992, 19, 14, 6).

Description: Here, the *cwd* ('home') was placed at the head of a path. Interestingly, the first argument path correctly omits the *cwd*, indicating that this was not a genuine misconception. The consequent error message was greeted with "'Cos I'm in 'home', stupid girl!" and the error was quickly detected, which also supports the idea that this was a 'slip'.

Commentary: The 'undoing' of no single prior action would have rehabilitated the command, so that state and functional information would be presented. In this instance, a mere reminder of the identity of the *cwd* would have been sufficient but it is better that the user has a little too much information rather than not enough. QDOS's error messages would read:

"The current directory is 'home'."
"The item you apply the command to must exist in the specified directory or in the current directory if you do not specify one."
"Specify the target directory by supplying a path from the current directory to the target directory via all the intervening directories, i.e. 'temp'."

S2

nonevent([mv, ' ', scrap, (/), tojim], reference, persmail, scrap, persmail, tuesday, 21, january, 1992, 18, 12, 57).

Description: This compound error arose from the twin misconceptions that (a) all pathnames start from the home directory, whatever the identity of the *cwd* and (b) *mv* is like *ls*, in that the *cwd* is the

default argument.

Commentary: This should also have been signalled as a syntax error to the effect that *mv* requires two arguments; one for the source and one for the destination. The reference error would be dealt with without the benefit of any evidence from the log and help information would thus be based on relevant aspects of the current state, such as the *cwd* and location of the target, and violated preconditions on referencing via paths thus:

"The current directory is 'persmail'."
"The item you apply the command to must exist in the specified directory or in the current directory if you do not specify one."
"Specify the target directory by supplying a path from the current directory to the target directory via all the intervening directories, i.e. 'personal/home/scrap/tojim'."

S1

nonevent([mv, ' ', complaintin, ' ', temp], duplication, home, temp, busin, monday, 20, january, 1992, 19, 18, 32).

Description: Because of the dual function or ambiguity of the 'mv' command in UNIX; both in moving and renaming items, it is impossible for the system to tell if this is a duplication or reference error. It is treated as the former, when in fact it is the latter. This obviously makes for highly confusing error reports, since the error message bears no relation to the user's intentions. The subject was attempting to move a file ('complaintin') from the *cwd* ('busin') to a 'great uncle' directory ('temp'), without supplying the correct ('business/home') path. The system sees this as an attempt to rename 'complaintin' to 'temp' and rejects this name duplication.

Commentary: One answer here is to remove the inherent ambiguity, so that the system at least has access to the meaning of the action. The

pragmatics of the action can then be addressed more accurately. Alternatively, the classification of error type would follow and depend on the possible worlds analysis, so that both interpretations of the error could be pursued and compared. In this case, there is no evidence to support either option over the other, which would lead to information relating to both being provided as alternative user misconceptions. The user could then choose the most appropriate interpretation. As it is, QDOS's error message would address the duplication interpretation with reference to the prior action creating the original 'temp' directory:

"That command failed because a directory 'temp' was created <timelapse> ago."

Straightforward Context Errors

Many context errors were committed merely because subjects were unaware of some feature of the system, for example, that some directories were protected or that an item existed somewhere in the directory structure with the same name as an item being created. In these cases, subjects simply assume that an action will succeed, without information to the contrary. Similarly, without any evidence available for the subjects belief in any but the most 'obvious' possible world (that closest to the actual world), the proposed QDOS analysis would default to that world, e.g. the world where the protected item is unprotected.

S1

nonevent([cd, ' ', persaddr], execute, addresses, persaddr, addresses, monday, 20, january, 1992, 18, 4, 44).

Description: This was an attempt to make the execute-protected directory 'persaddr' the working directory. The subject was not aware that having execute permission is a precondition of this action.

Commentary: A prior action in the log, removing execute permission from the directory would provide evidence for a state misconception here. However, this mode had been set in the startup file and was not one of the subject's own actions and is thus misleading. The absence of such evidence would lead to the provision of state and functional information regarding the mode set on 'persaddr' and the precondition of execute permission on directory changes, along with directions for using the 'chmod' command. Such would seem adequate in the current situation:

"'persaddr' is execute protected."
"You must have execute permission on the item for that command to succeed. Use the 'chmod' command to gain execute permission."

S1

nonevent([mkdir, ' ', business], duplication, home, business, home, monday, 20, january, 1992, 19, 4, 4).

Description: This command constituted an attempt to duplicate a directory name. The subject was unaware of the bar on duplicated names.

Commentary: The existence of an action in the log, creating or naming the directory 'business' would be taken as evidence for a state misconception concerning the existence of that directory. This could produce incorrect diagnoses, in that the creation of an item does not imply knowledge about rules concerning duplication. This indicates that a possible extension to the criteria for diagnosis might be some kind of representation of the implications for user knowledge of the various commands which might be found in the log. The use of the 'chmod' command might indicate some familiarity with the use of protection codes and their effects, whereas the use of 'mkdir' does not indicate any familiarity with the associated rules of duplication. We have indicated that this kind of inference is rather

tenuous but it could prove a useful source of negative evidence to suppress incorrect diagnoses, rather than as support for particular diagnoses. As it is, QDOS would fail to *directly* address the rule misconception:

"That command failed because a directory 'business' was created <timelapse> ago."

S1

nonevent([rmdir, ' ', busaddr], nonempty, nul, nul, addresses, monday, 20, january, 1992, 19, 38, 20).

Description: The subject was not aware of the rule that a directory must be empty before it can be deleted.

Commentary: The 'undoing' of no single event in the log would have rendered this command executable, therefore state and functional information would be presented. This would seem to address the problem:

"busaddr is not empty."
"A directory must be empty before it can be deleted."

S2

nonevent([cd, ' ', persaddr], execute, addresses, persaddr, addresses, tuesday, 21, january, 1992, 17, 38, 19).

Description: An attempt was made to change directory to 'persaddr' which was execute protected. The user was unaware of this protected state. It had been set in the startup file as part of the task definition and not set by the present user.

Commentary: The presence of the command in the log, setting the protection on that directory, would be taken as evidence for psychological proximity of a possible world in which that command had not occurred and in which the directory was therefore not protected. It is assumed that the user himself set this value as the command is in his log. Information designed to correct this misconception would be presented:

"That command failed because 'persaddr' was execute protected <timelapse> ago."

S2

nonevent([mkdir, ' ', busaddr], duplication, business, busaddr, business, tuesday, 21, january, 1992, 18, 29, 4).

Description: An attempt to create a directory 'busaddr' when one already existed. On encountering this error the user exclaimed "Where?" and on finding the offending item "...I forgot to take that directory out."

Commentary: The creation of the original 'busaddr' would be evidence for the psychological proximity of a world in which this directory did not exist. The prior event would be cited, implying the relevant current state:

"That command failed because a directory 'busaddr' was created <timelapse> ago."

Misinterpretation

On many occasions the problematic situation was not resolved by the consequent output of an error message. In order to recover, users must be able to interpret the error correctly. A reference error can occur for many reasons and the terse UNIX® 'x not found' error message leaves the user

to speculate among them. Thus typing errors could engender speculation about the correctness of the pathname (**S4** - 19.10.02).

It was also possible for subjects to infer a correct conclusion for the wrong reasons: **S2** (17.43.51) incorrectly interpreted a reference error caused by a naming mistake as the (correct) rule that directories cannot be moved. This resulted in him not noticing the name error ('address' for 'addresses') and committing the same name error later on (17.58.21). This happened again when a path error was interpreted in the same way (18.12.57). In both cases, the true nature of the error went unnoticed. As QDOS is not designed for lexical errors in open-class terms, this kind of error would be assumed to be corrected by a lower level of help function.

Compound Errors

Commands sometimes embody more than one error, while the error message received would only address the first error to be 'parsed'. This meant that on recovery from the signalled error, the subject would inevitably recommit the others, requiring the same action to be attempted three or possibly four times instead of just twice. A similar kind of situation occurs when the user receives a 'permission denied' error on trying to enter a directory. In some cases, the user would change the access code of the directory to '--x' (execute allowed) but then be thwarted in listing the directory because of the lack of read access.

Feedback Problems

All subjects were mystified by the system's practice of displaying nothing on the screen when an empty directory is listed. S4 interpreted this as meaning that the item was a file until she was disabused of this belief when she tried to list a file and received the 'x is a file' message (18.39.23).

Subjects often failed to assimilate information which was on the screen, even though it was directly relevant to the current action and could have been instrumental in avoiding a subsequently committed error.

Other Observations

S4 used 'cd x' to check or ensure that 'x' was the *cwd* and took the 'x not found' as a confirmation of this. This strategy worked most of the time as she was typically correct in her guess concerning the *cwd*. However, this led to a major disorientation later when she had forgotten a previous 'cd' which took her to the 'home' directory, two levels up from directory 'x'. Her usual 'cd x' command and its following 'x not found' message convinced her that 'x' was the *cwd* but none of x's contents could be accessed there. (19.44.13) "I'm home!??...How did I get there??" was her response when she finally discovered why. QDOS's more detailed error messages would have prevented this strategy's misleading consequences.

The 'ls' command defaults to 'ls *cwd*' when no directory is specified but this is not true for other commands. Another common error was to assume *cwd* to be the default if no directory is specified for all commands because this is true for 'ls'. This misconception presented itself in the form of syntax errors. One subject used MS DOS command format on one occasion (18.39.07).

7.12 Summary

There were thirty five context errors in **S1**'s session log. On three occasions, the inferences drawn from the diagnostic criteria were inappropriate to the user's particular situation, in that some of the information germane to the error would not have been provided or that some misleading information would have been given. On three occasions, the user would have received information relating to system rules, as well as relevant state information, when the misconception was state-based. On the other twenty nine occasions, the misconception diagnosed via the stated criteria broadly matched that apparent in the analysis of the recorded protocol and log file.

There were twenty six context errors in **S2**'s session log. On two occasions, the inferences drawn from the diagnostic criteria were inappropriate to the user's particular situation. On seven occasions, the user

would have received information relating to system rules, as well as relevant state information, when the misconception was state-based. On the other seventeen occasions, the misconception diagnosed matched that apparent in the analysis of the recorded protocol and log file.

There were eighteen context errors in S3's session log. On one occasion, the inferences drawn from the diagnostic criteria were inappropriate to the user's particular situation. On three occasions, the user would have received information relating to system rules, as well as relevant state information, when the misconception was state-based. On the other fourteen occasions, the misconception diagnosed matched that in the analysis of the recorded protocol and log file.

There were thirty three context errors in S4's session log. On five occasions, the inferences drawn from the diagnostic criteria were inappropriate. On nine occasions, the user would have received information relating to system rules, as well as relevant state information, when the misconception was state-based. On the other nineteen occasions, the misconception diagnosed via the stated criteria broadly matched that apparent in the analysis.

There were one hundred and twelve context errors in all users' session logs. On eleven occasions, the inferences drawn from the diagnostic criteria were inappropriate. On twenty two occasions, the user would have received excess information. On the other seventy occasions, the misconception diagnosed via the stated criteria matched that of the analysis of the recorded protocol and log file. The overall success rate in approximate percentage terms is thus respectively: fail = 9%, verbose = 21%, correct = 70%.

7.13 Conclusions

As can be seen from the detailed description of context error situations encountered by the users in this exercise, it is estimated, by comparing the projected diagnosis from the stated criteria against an appraisal of the problem, gained from a study of the video tapes and log files, that the

proposed help system could be effective in accurately determining user misconception in seven out of ten cases. In two out of three of the remaining cases, information in excess of that required, relating to preconditions on the command involved, would be generated and in only one case in ten would a totally spurious diagnosis be made.

Although it is admitted that the given criteria are not invariably adequate to the task of diagnosing user misconception, it must be said that, in many cases, it was not possible for a human observer to infer the underlying problem merely from watching the interaction. It is therefore concluded that the success rate can be counted as notable and significant.

As they stand, the criteria for diagnosing user misconception mean that evidence for a state misconception is searched for and, if found, indicates that the misconception is state-based. As there are, as yet, no positive criteria for diagnosing function-based misconceptions, errors generating hypotheses of state-based misconception which are evidentially unsupported are treated as if possibly state-based and possibly function-based. This is not an ideal state of affairs, as it means that users will often be presented with information of which they might already be in possession. Positive criteria supporting function-based misconception hypotheses would therefore be extremely useful. It has been shown that the notions of logical and psychological proximity of possible world models of user misconception are useful in the selection of candidate hypotheses. It remains to be seen if other criteria are available in order to enhance the design of the help system.

In many instances, the help system is able to reinforce the user's understanding of the system in relating, by their effects, past actions to commands involved in current errors, making explicit the causal relations between the effects and preconditions of different commands. However, as a basis for determining misconception, these relationships are not always reliable, since some of them are less obvious than others. Thus, while it apparently works well to use the fact that a file was moved and that this action had been forgotten, in order to explain a failed attempt to access the

file in its old location, this strategy fails to produce accurate results where effects and preconditions are less obvious, such as when the creation of a directory means that its name cannot be used again (duplication). This phenomena led to more than one of the erroneous diagnoses in the protocol analysis exercise.

The experience has also shown how inconsistency and ambiguity can lead to unfulfilled expectations and greatly complicate the resolution of error situations. The default arguments for *ls* and *cd* caused errors and the ambiguity of the *mv* command led to error misinterpretation by effectively doubling the possible interpretations.

The proposed help system is as accurate as it can be, given the level of information and evidence to which it has access. The less evidence there is, the less specific it can be about the misconception and the more generalised the information it is able to provide. However, this has the beneficial side effect of rewarding experimental behaviour on the part of the user, since such behaviour will tend not to relate so much to past actions and therefore engender more general information in the form of state-based and function-based diagnoses.

It has been noted that users may entertain general misconceptions, unrelated to any particular command, for example that pathnames always start with the 'home' directory or that failed commands somehow still achieve their intended end. It is feasible that the help system could be extended to address this kind of misconception, although no attempt is made to do so here.

It has also been noted that components of compound errors deserve to be treated concurrently and prioritised so that: a) users are not led to commit component errors again and b) those which preclude the objective of another component take priority. For example, an attempt to apply the *ls* command to a file, which also embodies a bad reference to that file, should be failed on logical grounds, rather than referential grounds.

All in all, it is estimated that the evaluation conducted has provided strong vindication of the proposed approach both in terms of its analysis

and prediction of interface characteristics which tend to promote the kind of errors discussed and its success in diagnosing many situations of user misconception. Chapter 8 considers these claims in more detail.

8. Conclusion

This chapter attempts to draw together all of the strands of the others and to evaluate their strengths and weaknesses as they affect the initial aims of the research.

8.0 Review

The principal aim of this research has been to demonstrate the relevance of certain characteristics of NL communication to the HCI context, motivated by a) the conviction that attention to such a relation can help to improve the usability of computer systems and b) resistance to the idea that the graphical user interface embodies a type of communication between user and system which bears no relation to that of NL. It has been an intention to show that communicational issues in HCI cut across issues of interface style.

To this end, situations of communication breakdown or dialogue failure have been suggested as being particularly perspicuous in highlighting important aspects of dialogue design and a significant class of contextual error has been identified and analysed by applying certain theoretical constructs, adapted from NL pragmatics. It has been argued that such situations occur, regardless of particular interface styles, as a result of the inability of typical systems to adequately enable the user to build and maintain an appropriate mental model of the system, owing to the lack of attention to the principles of cooperative communication. It has also been argued that, in order to rectify this situation and provide a cooperative

response, a degree of intelligence is required in the interface to reason about the interaction as it proceeds.

A model of user misconception has been proposed, based on the notion of possible worlds, and mechanisms have been suggested for selection between alternative candidate possible world models of user misconception, following Gricean principles of cooperative communication. Such representations and mechanisms have been combined in the design of a rule-based system intended to act as an intelligent assistant for the diagnosis and resolution of contextual errors in interactive command line systems. The assistant is intended to provide appropriate cooperative responses, lacking in a typical system's feedback mechanisms.

A simulation of the UNIX® file system has been described, along with a restricted implementation of the proposed help system, designed to address contextual errors arising in users' interactions with the UNIX® file system. This software was produced for utilisation in evaluative exercises to be conducted on the proposed help system design and the theory supporting it.

Two evaluative exercises have been conducted in order that some measure of the effectiveness of the proposed help system design and its underlying theory might be made. The first of these took the form of a classical experiment using balanced groups of subjects. The second involved four subjects and used the method of protocol analysis in order to gain a more detailed, if more subjective, account of users' interactions in situations of contextual error.

8.1 The UNIX® Experience

The experimental exercise cannot be considered as anything but inconclusive. Even extrapolating the results does not demonstrate their significance. It is probable that a major contributory factor to this outcome was the disappointingly low turnout of subjects. However, it must also be said that the experiment could have been improved if it had been possible to prolong the subjects' system contact time, to determine some measure of ease of learning and to use a more developed version of the help system

software. On the positive side, useful data on the commission of various error types was produced.

The protocol analysis was an altogether more successful enterprise, producing many useful insights into the UNIX® file system design, as well as users' interaction with it. The detailed analysis contained in Appendix K, summarised in the last chapter, demonstrates the viability of the proposed approach to pragmatic error, at least within the context of this application type. Although it is admitted that the criteria for selecting between alternative hypotheses of user misconception need strengthening in order to render this process more robust, the success rate of diagnosis is high, given the relative simplicity of those criteria.

Hypotheses concerning the causes of context error have, to a large extent, been confirmed by the exercises here. However, the experience has also brought to light unexpected interaction behaviour which would certainly influence any development of the general approach. In particular, user actions based on 'high-level' or 'generalised' misconceptions about system functionality, rather than those concerning single commands, suggest that system responses should reflect this difference, so that users are not led to believe that general 'rules' apply only to particular commands.

A point to be made in this regard is that UNIX® error messages do not always make clear what aspects of the problem are within the user's control and which are not. For example, in UNIX®, the 'permission denied' error may be generated when the user has set an access mode or when the system supervisor has done so. The ability or inability to perform an action and the responsibility for a problematic state of affairs may or may not be in the control of the user. Thus it is important that these two cases are distinguished for users so that they are able to take responsibility for what they can control and do not attempt to influence matters beyond their control

It has been confirmed that context errors are caused by the inability of subjects to maintain an adequate model of transient system states involving,

for example, the identity of the *cwd*, the access protection, existence or location of items and so on. One possible solution to this kind of problem involves some method of lessening the burden on users' short term memory by making it unnecessary to recall such details. To some extent, this can be achieved in the WIMPS style of interaction (exemplified in the Macintosh Finder) in that features of the current system state, as mentioned above, are displayed graphically as icons and their attributes on the 'desktop', while the direct manipulation style of interaction ensures that the users' attention is drawn to the relevant information. One of the important characteristics of the WIMPS interface is that it reduces the 'command space' to the set of available commands (in the menu system) and changes the burden of recall to one of recognition for the user.

However, as has been suggested earlier, this approach has its limitations. Firstly, there will always be a limit to the useful amount of information that can be displayed at one time and therefore the information lying outside of the displayed domain is subject to the same recall constraints and difficulties of the command line interface. An item must be brought into view before it can be manipulated and so the user must accurately recall how this is to be achieved if a possibly extensive search is to be avoided. In the case of a previously deleted or renamed item, a search of the whole file system might have to be undertaken for the absence of the said item to be established. A similar story could be told for items having undergone relocation in the file system. Other criticisms of the WIMPS style interface can be found in Ehrich *et. al.* (1986; p. 173).

In short, unlike the command line interface, the state orientated philosophy does not, in principle, allow for a speculative command. The problem being that the level of support for such speculation found in the typical commandline interface is not appropriate for its effective use. Users are met with negative messages such as 'x not found' or 'permission denied' instead of information that they really require such as, for example, that the item has been renamed, deleted or moved, plus its new name or location.

Secondly, the user can usually only really explore WIMPS applications within the confines of correct usage. It is certainly true that it is possible to learn as much (if not more) from mistakes than from correct usage but the philosophy of preclusion of errors (inherent in the WIMPS style) effectively negates this. Users are (typically) shown the limits on their potential action but not why those limits are there. Ghosted items in a menu, *at best*, inform the user that that selection is not available but do not explain why. The issue being pursued here is not that of choosing between interfaces which preclude contextual error and those which do not but concerns the responsiveness of the dialogue itself with respect to the short and long term conceptual models which motivate the the user's behaviour.

What these points are intended to convey is not the superiority of one interface style over another; each has its merits and disadvantages, but that the problem of context error cuts across issues of interface styles and that the adoption of a particular style can not be a complete answer. It is suggested that the approach advocated here has implications for any form of human-computer interaction which is interpretable as communication and that this includes most computer systems.

Context errors are also committed because users are not aware of some permanent feature of the system. For example, it might not be known that write permission must be granted on a file for it to be possible to delete it. In contrast to transient system states, such system 'rules' do not need so much to be continually updated in the user's mental model of the system, since they are, by nature, constant.

One of the complications here is that, as far as the system is concerned, the consequences of unawareness of persistent and of transient system features can be exactly the same. An attempt to delete a protected file may arise either from a lack of awareness of the protection on the file or a lack of awareness that protection is a necessary consideration. If this ambiguity is to be addressed, there is need to consider all possible means for resolving it so that all error messages are relevant to the users' current situation. The presentation of 'catch all' information intended to cover all eventualities

will tend to verbosity and risk irritating the user with unnecessary material.

It is feasible for a context error to occur because the user is not aware of all of the consequences of a previous action. An example of this in the study was when **S1** changed the protection on a directory to gain entry to it by assigning it execute permission, only to find that it could not be listed because she had simultaneously removed read permission (18.12.19). However, this kind of 'diachronic' effect was not much in evidence. It is contended that this is partly because of the relative functional simplicity of the system being used. It is to be expected that a more functionally complex application would increase the possibility for such a problem. Again, automatic diagnosis of this problem is not a completely straightforward matter, owing to the sharing of symptoms with other causes.

It is clear from the study that significant problems arise for users from the failure of the host system to signal errors in such a way as to enable swift and effective recovery and correction of misconceptions. The messages generated by the standard UNIX® system do not address the user's error with any precision or in terms that relate to the user's current concerns. Nor do they address the question of likely causes. In short, the host system exhibits a distinct lack of cooperative behaviour, which accounts, on the whole, for the ease with which it is possible to misinterpret an error.

This is certainly an area where improvements can be made by considering the question of what a cooperative response would consist of in the light of the implications of the user's command in the context of use. The approach to this problem proposed here takes the Gricean paradigm as its starting point and looks at the general requirements stipulated for cooperative communication. If we imagine that it is a goal and responsibility of the system to help encourage effective dialogue, then it is in the system's interest to enable the user to gain optimum benefit from its own responses. Therefore its responses should be of a nature that the user might expect in the same situation from a cooperative human interlocutor.

If a shopkeeper persisted in refusing your money for some item you were purchasing, indicating that the amount was wrong but without indicating the correct amount, you would think him churlish. But this is exactly what the UNIX® system does with its 'permission denied' and 'x not found' style of message.

It is interesting that subjects were, on the whole, quite patient with the system, suggesting, perhaps, that they have come to expect this level of uncooperative behaviour from computer systems. Error reports in WIMP style interfaces are often just as uninformative, since, on the whole, they only depart from the command language dialogue style by presenting the message in its own window.

It is common for users' commands to embody more than one error. The host system's error handling algorithm reacts only to the first error 'parsed' and ignores the rest of the command. This means that, on some occasions, users correct for a signalled error, only to be told that the command is still faulty in some way. This situation is rather like redirecting someone to a shop one knows to be closed! Again, it seems wholly inappropriate in these circumstances not to attempt to provide the level of cooperativeness that one would expect from another person. The comments made above concerning misinterpreted errors apply equally here.

Although it is difficult to identify examples of a direct link between the lack of feedback and the commission of context error in the study, the potential for such a connection is obvious and the conclusion to be drawn from the incidence of subjects' confusion and misinterpretation when the system responds to a perfectly acceptable command with nil output (e.g. when an empty directory is listed) is self evident. Given the system's practice of merely redisplaying the prompt after the successful execution of a command, this is roughly equivalent to a 'yes' answer to a 'wh' question. Again, this is confusing (it is suggested) because this does not follow conventions of cooperative communication.

It is interesting to note that the appearance of relevant information on

the computer monitor is often overlooked. Users appear to assimilate displayed information that they are specifically looking for, while failing to notice or comprehend other relevant information presented simultaneously. For example, directories are often listed in 'long' format in order to discover their contents. This command also shows the access modes, the number of child directories and files for each item and whether the item is a file or subdirectory.

It is not uncommon for users then to attempt some unnecessary or incorrect action, such as trying to list a file, which would be avoided had this information been assimilated (S2 - 17.38.19 "I need to change access privileges to get into that one... I should have known that!"). This suggests that the fact that the system presents information is not sufficient for it to be of use to the user and that a necessary (although not sufficient) condition on the assimilation of information is that the time, circumstances and focus of its presentation be appropriate for its intended use. The fact that it is displayed is not always enough. The user's attention may have to be drawn to it at a relevant time for it to be effective.

In the context of the host system, users can be led to expect all appropriate commands to default to *cwd* if no directory is specified in the command. This is a generalisation of the *ls* command syntax and is not supported. This inconsistency in the software design can be seen as the direct cause of these errors. This is an interesting example of how the functionality of an application influences interface design and the practical difficulty of separating application and interface. On the one hand, it might seem to be a positive feature to allow commands with directories as arguments to default to the *cwd* if no argument is supplied, since this will reduce the amount of typing required by the user.

However, the possibility of a directory being execute-protected and thus not capable of 'entry' means that the strategy of making a directory the *cwd* in order to apply certain commands to it may be thwarted in some circumstances. It would also introduce complications for the *rmdir* command in that the *cwd* would have to be automatically altered. Failing

some other solution, the interface designer is left with the choice of leaving out what may be a useful default feature or introducing inconsistencies into the interface because of the functional relations within the application.

Knowledge transfer between systems can be stifled by superficial differences in command syntax. In UNIX® and other systems, it is possible to instruct the interface to translate some specified input as a recognised command. In this way, one can provide oneself (and possibly others) with shortened or personalised commands or 'aliases'. The error of using commands from a functionally similar system with differing command language can easily be given an interpretation by the system if it is configured to accept certain recognised synonyms such as *ls* (UNIX®) and *dir* (MS DOS).

It is also possible to widen the context of acceptability to natural language terms, to some extent, so that instead of the negative 'no such command' type of response, suggestions as to the desired command may be given, if not a direct interpretation. This provides the user with at least some avenue of action when held up by the lack of command knowledge or provides a more cooperative response in circumstances where it might reasonably be expected.

Again, the static nature of menu systems in WIMPS interfaces leaves no room for this kind of cooperative response. If users do not recognise the significance of a particular menu entry, there is no avenue open to them for applying prior knowledge or speculation. Outside help, in the form of manuals, on line help or personal consultation, must be sought.The same can be said of typing mistakes and errors in naming, which accounted for a number of errors in the study. A cooperative partner would not reject out of hand a misspelled name.

It is clear that context errors are caused by topological disorientation and that the command line interface used in the study is poor in supporting the maintenance of these features of the user's mental model. We have argued above that the state orientated display of the WIMPS style interface is only a partial solution and that it introduces problems and limitations of

its own.

Context errors are also caused by users' misconceptions about the functionality of the system and how it is properly exploited. The common misconception concerning path names always beginning from the 'home' directory is an example. We characterise this as a mismatch between the system and the users *functional* model of it (or functional features of the user's model). Again it is suggested that the closure of the set of available user actions found in the WIMPS philosophy is not an entirely satisfactory solution.

The misinterpretation of errors, the failure to deal sympathetically and cooperatively with compound errors and typographical errors can be seen as a general failure in the system to communicate effectively with the user, both in terms of attention to the precise nature of the error (let alone its psychological cause) and consequently in the form and content of the information presented to the user. Again this cuts across issues of interface style and demands attention to the quality of the dialogue. It has been shown that attention to the natural language pragmatics paradigm can greatly aid in this regard.

It is clear that information is best assimilated if provided at the time when it is pertinent to the user's current concerns and that its mere presence is not always sufficient. The problem of determining what information is pertinent at any particular time is not a trivial one in the normal course of an interaction. However, the approach advocated maintains that error situations represent excellent opportunities for achieving this and stresses the positive benefits that errors can bring.

Of course, it will come as no surprise to anyone with some experience of the standard UNIX® interface that the problems discussed above exist. However, the aim of this study is to evaluate a particular analysis of these problems and their relations to other forms of interaction and styles of interface and not the interface itself. The UNIX® interface was chosen because the incidence of context error is strikingly obvious in its use but this does not mean that the phenomenon is solely a feature of, and therefore

only relevant to, that kind of interaction. On the other hand, the study has provided a systematic analysis of a class of problems which UNIX® users encounter, in terms of a principled account of cooperative communication, which furnishes us with a deeper understanding of these problems than that promoted by the taxonomies of user error or dialogue failure discussed in Chapter 3 or the statistical studies cited.

8.2 The WIMPS Interface

Although there has not been time to study in any detail the application of this approach to alternative interaction styles, it is possible to make some remarks about the WIMPS interface and its relation to the issues we have been addressing. The WIMPS interface differs from the command line interface in the following ways:

- Access to commands is via the menu system and thus do not need to be remembered.
- Arguments are supplied, when necessary, by selecting them with the mouse pointer prior to selecting a command.
- Parameters are set via dialogue boxes
- The system state is graphically displayed (partially) and so requires less memorisation.
- Some actions may be carried out by directly manipulating graphical symbols or icons.
- Commands whose preconditions are unfulfilled are made unavailable by dimming or ghosting their menu entry.

Apart from direct manipulation, issuing a command in the WIMPS interface is essentially the same as its command line equivalent. A command is applied to a certain object, perhaps with certain parameters, while certain preconditions must be fulfilled. For example, the 'New Folder', 'Close' and 'Print Directory' commands in the Macintosh Finder all require the argument's window to be open and selected, while the 'Get

Info', 'Duplicate' and 'Put Away' commands require a drive, folder or file icon to be preselected.

The first thing that might be said about this is that it is not at all obvious why arguments should have to be specified prior to the selection of command rather than afterwards. This is merely an aspect of the error preclusion policy and could be a source of confusion, especially to users familiar with command line interfaces, which often allow a command to be specified before prompting the user for arguments and parameters.

Secondly, the obviousness of the requirement of specific preconditions for a command will vary according to the command and according to the user. For example, it may seem obvious that the floppy disk drive must be occupied by a disk for the 'Eject' command to be appropriate but it is certainly less obvious that a server volume must be selected as a precondition of the 'Get Privileges' option or that a window must be open before a new folder can be created.

Thirdly, the ghosting out of menu items is totally insensitive to anything but the presence or absence of specific preconditions. If a user preselects the wrong kind of object for a particular command, rather than failing to select one at all, the effect is exactly the same. The system always responds in its own terms, with regard to its own interests. Users are expected to infer the aspects in which their conception of the system is erroneous.

Because of its state-based display and the inability to issue commands whose arguments are not directly accessible, the prospect of state-based misconception is drastically reduced in the WIMPS environment and the need for error handling focuses mainly on function or rule-based misconception. From what has already been said, it will probably be obvious that the application of the approach proposed in this thesis would require that the WIMPS system responds more cooperatively to the selection of ghosted commands both by providing an explanation of the reasons for the unavailability of that command, in terms of its preconditions or presuppositions, and by indicating and explaining any incorrect selection of arguments or parameters in order to directly address

users' misconceptions. It also seems clear that this requirement would be much easier to implement for the same reasons. However, the shortcomings of this solution, stemming from characteristics inherent in this interface style, such as the inability to issue a speculative command and thus generate help information about a deleted item, for example, remain.

8.3 Error Messages

The question of the form and wording of help or error messages has not been discussed, while it is acknowledged that this is at least as important an issue as misconception diagnosis. There is little point in being able to automate the detection of user misconception if the user is unable to assimilate the resulting messages. It has not been the aim of the present research to investigate or present recommendations for this aspect of automated help systems. However, it has been stressed that information should be presented in terms related to the user's current concerns. That is to say that reference should be made by name to the objects involved in the error and to the action associated with the command involved. For example: "You cannot delete the file 'myfile' because its parent directory 'mydir' was write protected earlier in this session. (command was: chmod mydir 000)".

No definitive substantiation of the related claims that, a) errors can be beneficial and b) the preclusion of errors has negative effects on learning, has been made. Instead, an attempt has been made to persuade the reader of these points by argument. It is considered that the possibility of providing conclusive evidence for these claims is fairly remote and well beyond the scope of this research. However, the thesis presented here does not rely on their veracity, in as much the approach it advocates is perceived to be an effective one and it is claimed that this is the case, in that it supplies a generalisable method for rendering any human-computer dialogue more cooperative with regard to dialogue responses in a class of situations of user error.

8.4 Possible Further Developments

Apart from the application of the current approach to context error to interfaces other than command line format, there are two other areas which could reward further investment of research effort. The first is the strengthening of criteria for misconception diagnosis and the second is the design of a generic implementation of the assistant.

Thus far we have evinced two criteria for the diagnosis of user misconception. Logical proximity is the closeness of a particular possible world to the actual world, in terms of the number of transforming actions separating them. This criterion represents a general principle of conservation of the system state, which has the significance of crediting the user with the ability to track the system state in a mostly accurate manner, as it evolves throughout the interaction.

A possible world is said to be psychologically proximate to the actual world if it differs to it in some respect which, had it not, would rehabilitate the erroneous command. This relates to the user characteristic of distorting the actual world (in terms of the user's mental model of the system) only in ways which make sense, given some error in STM or lack of awareness of the effects of some action.

Thus, these two criteria relate to two of the possible causes of context error, the third cause being unawareness of the preconditions of some action. As yet, we have determined no positive criteria for diagnosing context errors with this cause and have used the lack of evidence under other criteria to indicate this possibility. This is not an entirely unsatisfactory state of affairs. While a context error caused by a failure in STM or lack of awareness of the effects of some action will invariably be reflected by the presence of a particular action in the interaction history, if the cause is a lack of awareness of the preconditions of some action, a causal antecedent action may not be present. However, this cannot be generally relied upon for accurate diagnostic results and some other, positive criterion for assessing the hypothesis of this cause would be welcome.

The only available source would appear to be some measure of user expertise, gained by monitoring user behaviour in the course of the interaction. One such method would keep counts of successful and unsuccessful uses of commands and precondition violations and use these to construct a representation of user expertise across the functionality of the application. This would, of course, need to be a dynamic user model, capable of responding to the user's gain in and loss of expertise over time.

If the Prolog clauses constituting the UNIX® simulation described in Chapter 6 are read declaratively, they can be seen as a functional description of (part of) the UNIX® file system. Given this functional description, a command input, a system state and an interaction history, the analyser will produce diagnoses of user misconception and generate appropriate help messages.

As it stands, the analyser is tied to the particular application but this is not a necessary condition. The analyser could be made to generate its hypotheses of user misconception via the functional rule-base defining the system, since all causal relations between commands and their effects and preconditions are represented within it. Thus, there is a distinct possibility of producing a generic assistant which would operate on a rule-base, system state, interaction log and command description to produce misconception diagnoses for a range of applications and, although it has proven to be a non-trivial task (Jackson & Lefrere, 1984), it is also possible that error or help messages could be generated from the rule-base and system state description. It is recognised that this would constitute a fairly ambitious project.

8.5 Concluding Remarks

This research owes much to that of others cited herein. The recognition of contextual error, the general call for conversationally cooperative systems and research on NL interfaces in general are only some of the factors which have been influential. The work represented in this thesis derives its own achievement and originality from four main areas:

1) The detailed exploitation and adaptation of the Gricean Cooperative Principle, its maxims and inferential processes to non-NL HCI and the theoretical and practical demonstration of the validity of this endeavour.

2) The analysis of part of the UNIX® interface and its vagaries in terms of a principled and consistent set of criteria, derived from NL communication, rather than a simple taxonomy of faults.

3) The identification of context error as a significant class of dialogue breakdown, the circumstances and incidence of which cuts across issues of interface style.

4) The application of these ideas, via the notions of possible worlds and logical and psychological/epistemic proximity as rationalising principles, to the design and implementation of a prototype system.

References

Aaronson, A. & Carroll, J.M., 1986, 'The Answer is in the Question: A Protocol Study of Intelligent Help', Research Report RC 12034, IBM Thomas Watson Centre, Yorktown Heights, N.Y. 10598.

Aaronson, A. & Carroll, J.M., 1987, 'Intelligent Help in a One-Shot Dialogue System', in Carroll, J.M., Tanner, P.P. (eds), *Human Factors in Computing Systems IV,* Amsterdam, North Holland, pp. 163-74.

Allen, J., 1983, 'Recognising Intentions from Natural Language Utterances', in Brady, M. & Berwick, R.C. (eds), *Computational Models of Discourse,* Cambridge, Mass., MIT Press, pp. 107-66.

Austin, J.L., 1962, *How to Do Things with Words,* Oxford, Clarendon Press.

Black, J.B. & Carroll, J.M., 1987, 'What Kind of Instruction Manual is the Most Effective?', in Carroll, J.M., Tanner, P.P (eds), *Human Factors in Computing Systems IV,* Amsterdam, North Holland, pp. 159-62.

Booth, P.A., 1989, *An Introduction to Human-Computer Interaction,* Hove, Lawrence Erlbaum.

Booth, P.A., 1990a, 'ECM: A Scheme for Analysing User-System Errors', *International Journal of Human-Computer Interaction* v. 2, n. 4, pp. 307-32.

Booth, P.A., 1990b, 'Identifying and Interpreting Design Errors', *Human-Computer Interaction: Proceedings of Interact '90,* Elsevier Science Publishers, North Holland, pp. 47-60.

Booth, P.A., 1990c, 'Approaches to Error Analysis in Human-Computer Interaction', *Proceedings of 5th European Conference on Cognitive Ergonomics,* pp. 199-213.

Bradford, J. H., Murray, W.D. & Carey, T.T., 1990, 'What Kind of Errors do UNIX users make?', *Human-Computer Interaction : Proceedings of Interact '90*, pp. 43-6.

Breuker, J., 1988, 'Coaching in Help Systems', in Self, J. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction,* London, Chapman & Hall.

Buxton, W., 1987, 'The "Natural" Language of Interaction: A Perspective on Non-Verbal Dialogues', *Proceedings of CIPS,* Edmonton, pp. 16-9.

Carberry, S., 1988, 'Modelling the User's Plans and Goals', *Computational Linguistics* vol. 14, n. 3, Sept.

Card, S.K., Moran, T.P. & Newell, A., 1983, 'The Psychology of Human-Computer Interaction', Hillsdale, New Jersey, Lawrence Erlbaum.

Carroll, J.M. & Aaronson, A.P., 1988, 'Learning by Doing with Simulated Intelligent Help', *Communications of the ACM,* v.31, n.9, pp.1064-79.

Carroll, J.M. & McKendree, J., 1987, 'Interface Design Issues for Advice-Giving Expert Systems', *Communications of the ACM,* v. 30, n. 1, pp. 14-31.

Carroll, J.M. & Mack, R.L., 1984, 'Learning to Use a Word Processor: By Doing, By Thinking and by Knowing', in Thomas, J.C. & Schneider, M.L. (eds), *Human Factors in Computer Systems,* Norwood, NJ, Ablex, pp. 13-51.

Carroll, J.M., Kellog, W.A. & Rosson, M.B., 1991, 'The Task-Artifact Cycle', in Carroll, J.M. (ed), *Designing Interaction: Psychology at the Human-Computer Interface,* Cambridge University Press.

Catrambone, R., Carroll, J.M., 1987, 'Learning a Word Processing System with Training Wheels and Guided Exploration', in Carroll, J.M., Tanner, P.P (eds), *Human Factors in Computing Systems IV,* Amsterdam, North Holland, pp. 169-74.

Charniak, E. & McDermott, D., 1985, *Introduction to Artificial Intelligence,* Addison Wesley.

Chomsky, A.N., 1965, *Aspects of the Theory of Syntax,* Cambridge Mass., MIT Press.

Cowan, W. B. & Wein, M., 1990, 'State Versus History in User Interfaces', *Human-Computer Interaction : Proceedings of Interact '90*, pp. 555-60.

Davenport, C. & Weir, G., 1986, 'Plan Recognition for Intelligent Advice and Monitoring', in Harrison, M.D. & Monk, A.F. (eds), *People and Computers: Designing for Usability,* Cambridge, Cambridge University Press, pp. 278-315.

Davidson, D., 1984, *Inquiries into Truth and Interpretation,* Oxford, Clarendon Press.

Davis, R., 1983a, 'Task Analysis and User Errors: a Methodology for Assessing Interactions', *International Journal of Man-Machine Studies,* n.19, pp. 561-574.

Davis, R., 1983b, 'User Error or Computer Error? Observations on a Statistics Package', *International Journal of Man-Machine Studies,* n. 19, pp. 359-376.

Diaper, D., 1989, 'Task Observation for Human-Computer Interaction', in Diaper, D. (ed), *Task Analysis for Human-Computer Interaction,* Chichester, Ellis Horwood.

Downton, A., 1991, 'Evaluation Techniques for Human-Computer Systems Design', in *Engineering the Human-Computer Interface,* McGraw-Hill.

Edmondson, W.H., 1989, 'Asynchronous Parallelism in Human Behaviour: A Cognitive Science Perspective on Human-Computer Interaction', *Behaviour and Information Technology,* v. 8, n. 1, pp. 3-12.

Ehrich, R.W., Johnson, D.H., Roach, J.W., Sanford, D.L. & Narang, P., 1986, 'Role of Language in Human-Computer Interfaces', in Ehrich, R.W. & Williges, R.C. (eds), *Advances in Human Factors Ergonomics: Human-Computer Dialogue Design,* Amsterdam, Elsevier Science Publishers.

Erlandsen, J. & Holm, J., 1987, 'Intelligent Help Systems', *Information and Software Technology,* v.29, n.3, pp.115-21.

Finin, T., 1983, 'Providing Help and Advice in Task Orientated Systems', *Proceedings of the International Joint Conference on Artificial Intelligence,* pp. 176-78.

Flores, F, & Winograd, T., 1986, *Understanding Computers and Cognition: A New Foundation for Design,* Ablex Publishing, N.Y.

Fountain, A.J. & Norman, M.A., 1985, 'Modelling User Behaviour with Formal Grammar', in Johnson, P. & Cook, S. (eds), *People and Computers: Designing the Interface.,* Cambridge University Press.

Frese, M., Brodbeck, F., Heinbokel, T., Mooser, C., Schleiffenbaum, E. & Thieman, P., 1991, 'Errors in Training Computer Skills: On the Positive Function of Errors', *Human-Computer Interaction,* v. 6, pp. 77-93.

Goldstein, I.P., 1979, 'The Genetic Graph: A Representation for the Evolution of Procedural Knowledge', *International Journal of Man-Machine Studies,* n. 11, n. 1, pp. 51-77.

Grace, J.E., 1987, 'The Man-Machine Interface: The Natural Language Barrier', *Human-Computer Interaction: Proceedings of Interact '87,* Elsevier Science Publishers, North Holland, pp. 549-54.

Green, M., 1986, 'A Survey of Three Dialogue Models', *A C M Transactions on Graphics,* v. 5, n. 3, pp. 244-75.

Green, R.G., Schiele, F. & Payne, S.J., 1988, 'Formalisable Models of User Knowledge in Human-Computer Interaction', in Van der Veer, C.G., Green, T.R.G., Hoc, J., Murray, D.M. (eds), *Working with Computers: Theory versus Outcome,* San Diego, Academic Press.

Grice, H.P., 1975, 'Logic and Conversation', in Cole, P., Morgan, J.L. (eds), *Studies in Syntax,* vol. 3, New York Academic Press.

Hanson, S.J., Kraut, R.E., Farber, J.M., 1984, 'Interface Design and Multivariate Analysis of UNIX Commands', *ACM Transactions on Office Information Systems,* v. 2, n. 1, pp. 42-57.

Harris, L.R., 1983, 'The Advantages of Natural Language Programming', in Sime, M.E. & Coombs, M.J. (eds), *Designing for Human-Computer Communication,* London, Academic Press, pp. 73-85.

Hecking, M., 1987, 'How to Use Plan Recognition to Improve the Abilities of the Intelligent Help System Sinix Consultant', *Proceedings of Interact '87,* Elsevier Science Publishers, North Holland, pp. 657-662.

Hill, I.D., 1983, 'Natural Language versus Computer Language', Sime, M.E. & Coombs, M.J., *Designing for Human-Computer Communication,* London, Academic Press, pp. 55-72.

Houghton, R.C., 1984, 'Online Help Systems: A Conspectus', *Communications of the ACM*, v. 27 n. 2, pp. 126-33.

Jackson, P. & Lefrere, P., 1984, 'On the Application of Rule-Based Techniques to the Design of Advice Giving Systems', *International Journal of Man-Machine Studies,* n. 20, pp. 63-86.

Jerrams-Smith, J., 1985, 'Susi - A Smart User-System Interface', in Johnson, P. & Cook, S. (eds), *People and Computers: Designing the Interface*, Cambridge University Press, pp. 211-20.

Johnson, P., 1985, 'Towards a Task Model of Messaging: An Example of the Application of TAKD to User Interface Design', in Johnson, P. & Cook, S. (eds), *People and Computers: Designing the Interface*, Cambridge University Press.

Jones, J. & Virvou, M., 1991, 'User modelling and Advice Giving in Intelligent Help Systems for UNIX', *Information and Software Technology*, v. 33, n. 2, pp. 121-33.

Kammersgaard, J., 1990, 'Four Different Perspectives on Human-Computer Interaction', in Preece, P. & Keller, L. (eds), *Human-Computer Interaction: Selected Readings*, Hemel Hempstead, Prentice Hall, pp. 42-64.

Kaplan, J., 1983, 'Cooperative Responses from a Portable Natural Language Database Query System', in Brady, M. & Berwick, R.C. (eds), Cambridge, Mass, MIT Press, pp.107-66.

Kass, R. & Finin, T., 1988, 'Modelling the User in Natural Language Systems', *Computational Linguistics,* v. 14, n. 3, pp. 5-22.

Kieras, D.E., Bovair, S., 1990, 'The Role of a Mental Model in Learning to Operate a Device', in Preece, P. & Keller, L. (eds), *Human-Computer Interaction: Selected Readings,* Hemel Hempstead, Prentice Hall, pp. 205-21.

Kieras, D.E. & Polson, P., 1985, 'An Approach to the Formal Analysis of User Complexity', *International Journal of Man-Machine Studies,* v. 22, n. 4, pp. 365-94.

Kobsa, A. & Wahlster, W., 1986, 'Dialogue-Based User Models', *IEEE Proceedings,* v. 74, n. 7, pp. 948-61.

Konolige, K., 1981, 'A First Order Formalisation of Knowledge and Action for a Multi-Agent Planning System', in Hayes, J.E., Mitchie, D., & Pao, Y. (eds), *Machine Intelligence* (10) Chichester, U.K., Ellis Horwood.

Landauer, T.K., 1990, 'Relations Between Cognitive Psychology and Computer Systems Design', in Preece, J. & Keller, L. (eds), *Human Computer Interaction: Selected Readings,* Hemel Hempstead, Prentice Hall, pp. 141-60.

Ledgard, H., Whiteside, J.A., Singer, A. & Seymore, W., 1980, 'The Natural Language of Interactive Systems', *Communications of the ACM*, v. 23, n. 110, pp. 556-63.

Levinson, S. C., 1983, *Pragmatics*, Cambridge University Press.

Lewis, C. & Norman, D.A., 1987, 'Designing for Error', in Baeker, R.M. & Buxton, W.A.S. (eds), *Readings in Human-Computer Interaction: A Multidisciplinary Approach,* Los Altos, Morgan Kaufman, pp. 627-38.

Lutze, R., 1987, 'Customising Help Systems to Task Structures and User Needs', *Proceedings of Interact '87,* Elsevier Science Publishers, North Holland.

McCoy, K., 1988, 'Reasoning on a Highlighted User Model to Respond to Misconceptions', *Computational Linguistics*, v. 14, n. 3.

Mack, R.L., Lewis, C.H. & Carroll, J.M., 1987, 'Learning to Use Office Systems: Problems and Prospects', *ACM Transactions on Office Automation Systems*, 1, 254-71.

Mayes, J.T., Draper, S.W., McGregor, A.M. & Oatley, K., 1990, 'Information Flow in a User Interface: The Effect of Experience and Context on the Recall of Macwrite Screens', in Preece, P. & Keller, L., (eds), *Human-Computer Interaction: Selected Readings,* Hemel Hempstead, Prentice Hall, pp. 222-34.

Monk, A., 1986, 'Mode Errors: A User-Centered Analysis and Some Preventative Measures Using Keying-Contingent Sound', *International Journal of Man-Machine Studies,* n. 24, pp. 313-27.

Monk, A.F. & Wright, P.C., 1991, 'Observations and Inventions: New Approaches to the Study of Human-Computer Interaction', *Interacting with Computers,* v. 3, n. 2, pp. 204-16.

Moran, T.P., 1981, 'The Command Language Grammar: A Representation for the User Interface of Interactive Systems', *International Journal of Man-Machine Studies,* n. 15, pp. 3-50.

Moran, T.P., 1983, 'Getting into a System: External-Internal Task Mapping Analysis', *CHI '83 Conference of Human Factors and Computing Systems,* New York, ACM, pp. 45-9.

Murray, D.M., 1987, 'Embedded User Models', *Proceedings of Interact '87,* Elsevier Science Publishers, North Holland, pp. 229-35.

Nielsen, J., 1990, 'The Art of Navigating Through Hypertext', *Communications of the ACM,* v. 33, n. 3, pp. 296-311.

Norman, D.A., 1983, 'Design Rules Based on Analyses of Human Error', *Communications of the ACM,* v. 26, n. 4, pp. 254-58.

Norman, D.A., 1983a, 'Some Observations on Mental Models', in Gentner, D. & Stevens, A.L. (eds), *Mental Models,* Hillsdale, N.J., Lawrence Erlbaum.

Norman, D.A., 1986, 'Cognitive Engineering', in Norman, D.A. & Draper, S. (eds), *User-Centred System Design: New Perspectives on Human-Computer Interaction,* Lawrence Erlbaum, pp. 31-61.

Nozick, R., 1981, *Philosophical Explanations,* Oxford, Open University Press.

Payne, S.J. & Green, T.R.G., 1986, 'Task Action Grammars: A model of the Mental representation of Task Languages', *Human Computer Interaction,* n. 2, pp. 93-133.

Pinsky, L., 1983, 'What Kind of Dialogue is it when Working with a Computer?', Green, T.G., Payne, S.J. & Van der Veer, G.C. (eds), *The Psychology of Computer Use,* London, Academic Press, pp. 29-40.

Prager, J.M., Lamberti, D.M., Gardener, D.L. & Balzac, S.R., 1990, 'Reason: An Intelligent User Assistant for Interactive Environments', *IBM Systems Journal,* v. 29, pp. 141-66.

Quilici, A., Dyer, M.G. & Flowers, M., 1988, 'Recognising and Responding to Plan-Oriented Misconceptions', *Computational Linguistics,* v. 14, n. 3, pp. 38-51.

Quine, W.V.O., 1960, *Word and Object,* Cambridge, Mass., MIT Press.

Reichman, R., 1985, *Getting Computers to Talk Like You and Me,* Cambridge, Mass., MIT Press.

Reisner, P., 1984, 'Formal Grammar as a Tool for Analysing Ease of Use: Some Fundamental Concepts', in Thomas, J.C., & Schneider, M.L. (eds), *Human Factors in Computing Systems,* New Jersey, Ablex.

Rich, E., 1979, 'User Modelling via Stereotypes', *Cognitive Psychology,* 3, 329-45

Sanford, D.L. & Roach, J.W., 1988, 'A Theory of Dialogue Structures to Help Manage Human-Computer Interaction', *IEEE Transactions on Systems, Man and Cybernetics,* v. 18, n. 4, pp. 567-74.

Searle, J., 1969, *Speech Acts*, Cambridge University Press.

Shakel, B., 1990, 'Human Factors and Usability', in Preece, J. & Keller, L. (eds), *Human-Computer Interaction: Selected Readings,* Hemel Hempstead, Prentice Hall, pp. 27-41.

Sutcliffe, A., 1988, *Human-Computer Interface Design,* London, Macmillan.

Thimbleby, H., 1990, *User Interface Design,* New York, ACM.

Weiskamp, K. & Shammus, N., 1988, *Mastering Smalltalk,* Wiley.

Whitefield, A., 1987, 'Models in Human-Computer Interaction: A Classification with Special Reference to their Uses in Design', *Human-Computer Interaction: Proceedings of Interact '87,* Elsevier Science Publishers, North-Holland, pp. 57-63.

Whiteside, J., Bennett, J. & Holtzblatt, K., 1987, 'Usability Engineering: Our Experience and Evolution', in Helander, M. (ed), *Handbook of Human-Computer Interaction,* North Holland, 791-817.

Wilensky, R., 1983, *Planning and Understanding: A Computational Approach to Human Reasoning*, Reading, Mass., Addison-Wesley.

Wilensky, R., Chin, D.N., Luria, M., Martin, J., Mayfield, J. & Wu, D., 1988, 'The Berkeley UNIX Consultant Project', *Computational Linguistics,* v. 14, n. 4, pp. 35-84.

Wilson, M.D., Barnard, P.J., Green, T.R.G. & Maclean, A., 1988, 'Knowledge-Based Task Analysis for Human-Computer Systems', in Van der Veer, C.G., Green, T.R.G., Hoc, J., Murray, D.M. (eds), *Working with Computers: Theory versus Outcome,* San Diego, Academic Press.

Winograd, T., 1972, *Understanding Natural Language,* New York, Academic Press.

Woodroffe, M.R., 1988, 'Plan Recognition and Intelligent Tutoring Systems', in Self, J. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction,* London, Chapman & Hall, pp. 213-25.

Woods, W.A., 1972, 'An Experimental Parsing System for Transition Network Grammars', in Rustin, R. (ed), *Natural Language Processing,* New York, Algorithmics Press, pp. 113-54.

Wright, P.C. & Monk, A.F., 1991, 'A Cost-Effective Evaluation Method for Use by Designers', *International Journal of Man-Machine Studies,* (in Press).

Young, R. L., 1991, 'A Dialogue User Interface Architecture', in Sullivan, W.S. & Sherman, W.T. (eds), *Intelligent User Interfaces,* New York, ACM Press, 157-76.

Young, R.M. & Harris, J.E., 1986, 'A Viewdata-Structure Editor Designed Around a Task/Action Mapping', in Harrison, M.D. & Monk, A.F. (eds), *People and Computers: Designing for Usability,* Cambridge University Press, pp. 435-446.

Bibliography

Abrams, K.H., 1987, 'Who's the Boss: Talking to Your Computer in the A.I. Age', *Human-Computer Interaction: Proceedings of Interact '87,* Elsevier Science Publishers, North Holland, pp. 931-6.

Alexander, H., 1987, 'Executable Specifications as an Aid to Dialogue Design', *Human-Computer Interaction: Proceedings of Interact '87,* Elsevier Science Publishers, North Holland, pp. 739-44.

Alty, J.L. & McKell, P., 1986, 'Application Modelling in a User Interface Management System', in Harrison, M.D. & Monk, A.F. (eds), *People and Computers: Designing for Usability,* Cambridge, Cambridge University Press, pp. 319-35.

Alty, J.L. & Mullin, J., 1987, 'The Role of the Dialogue System in a User Interface Management System', *Human-Computer Interaction: Proceedings of Interact '87,* Elsevier Science Publishers, North Holland, pp.1007-12.

Arens, Y., 1986, 'Modelling an Interlocutor's Perception of Context', *IEEE International Conference on Systems, Man and Cybernetics,* pp. 771-5.

Bailey, W.A., Kay, E.J, 1986, 'Toward the Standard Analysis of Verbal Data', *IEEE International Conference on Systems, Man and Cybernetics,* pp. 582-7.

Balzert, H., 1987, 'Objectives for the Humanisation of Software: A New and Extensive Approach', *Human-Computer Interaction: Proceedings of Interact '87* Elsevier Science Publishers, North Holland, pp.5-10.

Barnard, P., Wilson, M., Maclean, A. 1988, 'Approximate Modelling of Cognitive Activity with an Expert System : A Theory-Based Strategy for Developing an Interactive Design Tool', *The Computer Journal,* v. 31, No. 5, 1988, pp. 445 - 56.

Bennett, J.L., Lorch, D.J., Kieras, D.E., Polson, P.G., 1987, 'Developing a User Interface Technology for Use in Industry', *Human-Computer Interaction: Proceedings of Interact '87* Elsevier Science Publishers, North Holland.

Benyon, D., Innocent, P., Murray, D., 1987, 'System Adaptivity and the Modelling of Stereotypes', *Human-Computer Interaction: Proceedings of Interact '87* Elsevier Science Publishers, North Holland.

Benyon, D., Murray, D., 1988, 'Experiences with Adaptive Interfaces', *The Computer Journal,* v.31, n.5, pp. 465-73.

Berwick, R.C., 1983, 'Computational Aspects of Discourse', in Brady, M., Berwick, R.C. (eds), *Computational Models of Discourse,* Cambridge, Mass., MIT Press.

Bjorn-Anderson, N., 1988, 'Are "Human Factors" Human?', *The Computer Journal,* v.31, n. 5.

Booth, P.A., 1990, 'Using Errors to Direct Design', *Knowledge-Based Systems*, v.3, n. 2, pp. 67-76.

Brooke, J.B. 1986, 'Usability in Office Product Development', British Computer Society Special Interest Group for Computer-Human Interaction Bulletin Special Issue, April 1986.

Cambridge, E., Browne, D.P. & Mitra, S.K., 1987, 'Application Modelling for the Provision of an Adaptive User Interface', *Human-Computer Interaction: Proceedings of Interact '87,* Elsevier Science Publishers, North Holland, pp. 981-7.

Canter, D., Rivers, R., Storrs, G., 1985, 'Characterising User Navigation through Complex Data Structures', *Behaviour and Information Technology,* v. 4, n. 2, pp. 93-102.

Carter, J.A., Schweighardt, M., 1987, 'The Basis for User-Oriented, Context-Sensitive Functions', *Human-Computer Interaction: Proceedings of Interact '87*, Elsevier Science Publishers, North Holland, pp. 1027-32.

Clanton, C., 1983, 'The Future of Metaphor in Man-Computer Systems', *Byte,* December, 1983.

Clarke, I.A., 1987, 'Designing a Human Interface by Minimising Cognitive Complexity', *Human-Computer Interaction: Proceedings of Interact '87* Elsevier Science Publishers, North Holland.

Clowes, I., Cole, I., Arshad, F., 'User Modelling Techniques for Interactive Systems', in Johnson, P. & Cook, S. (eds), *People and Computers: Designing the Interface*, C.U.P., Cambridge.

Cockton, G., 1986, 'Where do we Draw the Line? Derivation and Evaluation of User Interface Separation Rules', in Harrison, M.D., Monk, A.F. (eds), *Designing for Usability,* Cambridge, Cambridge University Press.

Cockton, G., 1987, 'A New Model for Separable Interactive Systems', *Human-Computer Interaction: Proceedings of Interact '87* Elsevier Science Publishers, North Holland.

Cooper, M., 1988, 'Interfaces that Adapt to the User', in Self, J. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction,* London, Chapman & Hall.

Costa, H., Duchenoy, S., Kodratoff, Y., 1988, 'A Resolution-Based Method for Discovering Students' Misconceptions', in Self, J. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction,* London, Chapman & Hall.

Coutaz, J., 1987, 'PAC: An Object-Oriented Model for Dialogue Design', *Human-Computer Interaction: Proceedings of Interact '87* Elsevier Science Publishers, North Holland.

Doyle, J.R., 1990, 'Naive Users and the Lotus Interface: A Field Study', *Behaviour and Information Technology*, v. 9, n. 1, pp. 81-89.

Drummond, M., 1989, 'AI Planning: Tutorial and review', AIAI-Technical Report 30, Edinburgh University.

Dzida, W., Hoffman, C., Valder, W., 1987, 'Mastering the Complexity of Dialogue Systems with the Aid of Work Contexts', *Human-Computer Interaction: Proceedings of Interact '87* Elsevier Science Publishers, North Holland, pp. 29-33.

Elliot, C. 1987, *A Critical Survey of Theories of Indirect Speech,* M.Phil. Thesis, University of Nottingham.

Elliot, C. 1990, 'Linguistic Models in the Design of Co-operative Help Systems', *Proceedings of Interact '90*, North Holland, Elsevier.

Elsom-Cook, M., 1988, 'Guided Discovery Tutoring and Bounded User Modelling', in Self, J. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction,* London, Chapman & Hall.

Fikes, R.E., Nilsson, N.J., 1971, 'STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving', *Artificial Intelligence,* n. 2, pp. 189-208.

Foley, J.D. & Van Dam, A., 1982, *Fundamentals of Computer Graphics,* Philippines, Addison Wesley.

Furnas, G.W., Landauer, T.K., Gomez, L.M., Dumais, S.T., 1987, 'The Vocabulary Problem in Human-System Communication', *Communications of the ACM,* v. 30, n. 11.

Greenberg, S., Witten, I.H., 1985, 'Adaptive, Personalised Interfaces - A Question of Viability', *Behaviour and Information Technology*, v. 4, n. 1, pp. 31-45.

Hart, A., 1985, 'Knowledge Elicitation: Issues and Methods', *Computer Aided Design,* v. 17, n. 9.

Hartley, J.R., Smith, M.J., 'Question Answering and Explanation Giving in On-line Help Systems', in Self, J. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction,* London, Chapman & Hall.

Jeremaes, P., 1987, 'Specifying a Logic of Dialogue', Discussion Paper for January Meeting of British Computer Society Human-Computer Interaction Special Interest Group.

Jones, J., Millington, M. and Ross, P., 1988, 'Understanding User Behaviour in Command-Driven Systems', in Self, J. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction,* London, Chapman & Hall.

Kautz, H.A., Allen, J.F., 1986, 'Generalised Plan Recognition', A.A.A.I., pp. 32-57.

Kohl, A., Rupietta, W., 1987, 'The Natural Language Metaphor: An Approach to Avoid Misleading Expectations', *Proceedings of Interact '87,* Elsevier Science Publishers, North Holland, pp. 657-662.

Maddix, F., 1990, *Human-Computer Interaction: Theory and Practice,* Ellis Horwood, Chichester.

Morris, D., Theaker, C.J., Phillips, R., Love, W., 1988, 'Human-Computer Interface Recording', *The Computer Journal,* v. 31, n. 5, pp. 437-44.

Nievergelt, J. & Weydert, J., 1980, 'Sites, Modes and Trails: Telling the User of an Interactive System Where He Is, What He Can Do and How to Get Places', *Methodology of Interaction,* Guedj, R.A. (ed), North Holland, pp. 327-338.

Norman, D.A., 1981, 'A Psychologist Views Human Processing: Human Errors and Other Phenomena Suggest Processing Mechanisms', *Proceedings of 7th International Joint Conference on Artificial Intelligence,* pp. 1097-1101, William Kaufmann, Los Altos.

Ogden, W.C., Sorknes, A., 1987, 'What do Users Say to Their Natural Language Interface?', *Proceedings of Interact '87,* Elsevier Science Publishers, North Holland.

Payne, S.J., 1991, 'A Descriptive Study of Mental Models', *Behaviour and Information Technology,* v. 10, n. 1, pp. 3-21.

Richards, M.A., Underwood, K., 1984, 'Talking to Machines: How are People Inclined to Speak?', *Proceedings of the Ergonomics Society Annual Conference,* pp. 62-7.

Robson, C., 1983, *Experimental Design and Statistics in Psychology,* Harmondsworth, Penguin.

Robson, C., 1990, 'Designing and Interpreting Psychological Experiments' in Preece, J., Keller, L., (eds), *Human-Computer Interaction: Selected Readings,* Hemel Hempstead, Prentice Hall, pp. 357-67.

Rubin, K.S., Jones, P.M., Mitchell, C.M., 1988, 'OFMspert: Inference of Operator Intentions in Supervisory Control Using a Blackboard Architecture', *IEEE Transactions on Systems, Man and Cybernetics,* v. 18, n. 4.

Sandberg, J., Breuker, J. & Winkels, R. 1988, 'Research on Help Systems: Empirical Study and Model Construction' *Proceedings of the European Conference on A.I.,* Munich, pp. 106-11.

Schroder, M., 1988, 'Evaluating User Utterances in Natural Language Interfaces to Databases', *Computers and Artificial Intelligence,* v. 7, n. 4, pp. 317-37.

Shneiderman, B. 1987, *Designing the User Interface: Strategies for Effective Human-Computer Interaction,* Menlo Park, Addison-Wesley.

Shrager, J. & Finin, T., 1982, 'An Expert System that Volunteers Help', *Proceedings of the National Conference on Artificial Intelligence AAAI-82,* Pittsburgh, Pennsylvania, Carnegie-Mellon University, pp.339-340.

Snowberry, K., Parkinson, S., Norwood, S., 1985, 'Effects of Help Fields on Navigating through Hierarchical Menu Structures', *International Journal of Man-Machine Studies,* n. 22, pp. 479-91.

Tauber, M.J., 1988, 'On Mental Models and the User Interface', in Van der Veer, C.G., Green, T.R.G., Hoc, J., Murray, D.M. (eds), *Working with Computers: Theory versus Outcome,* San Diego, Academic Press.

Taylor, C., & Self, J., 1990, 'Monitoring Hypertext Users', *Interacting with Computers,* v.2, n.3, pp. 297-312.

Ten Hagen, P.J.W., 1980, 'Syntactic Structures', Guedj, R.A., (ed), *Methodology of Interaction,* North Holland, pp.18-24.

Totterdell, P., Cooper, P., 1986, 'Design and Evaluation of the AID Adaptive Front-End to Telecom Gold', in Harrison, M.D. & Monk, A.F. (eds), Cambridge, Cambridge University Press, pp.435-446.

Totterdell, P.A., Norman, M.A and Browne, D.P., 1987, 'Levels of Adaptivity in Interface Design', *Human-Computer Interaction: Proceedings of Interact '87* Elsevier Science Publishers B.V. North Holland, pp .

Waterworth, J. 1982, 'Man-Machine Dialogue Acts', *Applied Ergonomics*, vol. 13, pp. 203-7.

Winfield, I., 1986, *Human Resources and Computing,* Heinemann.

```
/*
analyse/0 sorts the error types, calls the appropriate analysing
routines and gives advice on the preconditions of commands involved in
transient errors.
*/

analyse :-                %don't analyse good commands
 error(noerror, $, $).

analyse :-                %or eternal errors
 error(Errortype, $, $),
 isin([syntax, logical, range], Errortype).
analyse :-
 error(Errortype, $, $),      %what error?
 kind(Errortype).          %inspect it

kind(reference):-          %item was (re)moved
 error($, Source, Erroneous),
 lastrmvcheck,
 refprecond.
kind(reference):-          %item in last cwd
 error($, Source, Erroneous),
 lastcdcheck,
 refprecond.
kind(reference):-        %item never existed
 refprecond.
kind(read):-
 error($, Source, Erroneous),
 lastreadcheck,
 readprecond.
kind(write):-
 error($, Source, Erroneous),
 lastwritecheck,
 writeprecond.
kind(execute):-
 error($, Source, Erroneous),
 lastexecheck,
 exeprecond.
kind(duplication):-
 error($, Source, Erroneous),
 %writelist(['"', Erroneous, '"', ': is a duplicate.']), nl,
 lastmkmvcpcheck,
 dupprecond.
kind(nonempty):-
 nonemptyprecond.

/*
appearance/0 disguises the Mac interface by hiding the menubar and the
mouse and setting the window title to "QDOS"
*/

appearance :-
 frontwindow(Query),     %get ptr to front window
 setwtitle(Query, '\004QDOS'),   %set the title of it to <text>
 hidecursor,      %hide mouse cursor
 clearmenubar,      %clear menu list
 drawmenubar.     %redraw empty menubar
```

```
/*
cd/1 performs directory changes after checking the command for errors.
*/

cd([]):-
 change(home).    %no argument means home
cd([' '|[]]):-        %a space but no argument is rejected
 write('Usage : cd [directory]'), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
cd([' '|Path]):-  %syntax ok
 cwd(Cwd),    get cwd
 tracedir(Path, Cwd). trace the path
tracedir([$, ' '|$], $):-
 writelist(['Path separator is: ', '"', /, '"', '.']), nl,
 retract(error(_, _, _)),     %wrong separator
 assert(error(syntax, nul, nul)).
tracedir([File], Cwd):-      % item is a file
  file(File, $, $),
 writelist(['"', File, '"', ' is a file.']), nl,
 retract(error(_, _, _)),
 assert(error(logical, nul, nul)).
tracedir([Dir], Cwd):-    %target is child of cwd
 dir(Dir, $, Cwd, $),
 change(Dir).
tracedir([Dir], Cwd):-    %target is parent of cwd
 dir(Cwd, $, Dir, $),
 change(Dir).
tracedir([Head, '/'|Tail], Cwd):-   %trace upward
 dir(Cwd, $, Head, $),
 tracedir(Tail, Head).
tracedir([Head, '/'|Tail], Cwd):-   %trace downward
 dir(Head, $, Cwd, $),
 tracedir(Tail, Head).
tracedir([Head|$], Cwd):-
 writelist(['"', Head, '"', ' : not found']), nl,
 retract(error(_, _, _)),
 assert(error(reference, Cwd, Head)).

change(Dir):-
 xperm(Dir),    %protected?
 getdatetime(DoW, Day, Month, Year, Hour, Min, Sec),
 retract(cwd(Cwd)),
 retract(prewd(_, _, _, _, _, _, _, _)),  %record last dir change
 assert(prewd(Cwd, DoW, Day, Month, Year, Hour, Min, Sec)),
 assert(cwd(Dir)).
change($):-
 modefail(yes),   %protected target
 retract(modefail(yes)).

/*
chmod/1 changes the protection code on files and directories.
*/

chmod([' ', Mode, ' '|Path]):-
 checkmode(Mode, Path).
chmod($):-    %syntax error
 write('Usage : chmod <mode> <path>.'),
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).

checkmode(Mode, Path):-     %correct mode?
 isin([000,100,200,300,400,500,600,700], Mode),
```

```
 cwd(Cwd),
 gettarget(Mode, Path, Target, Cwd).
checkmode(Mode, $):-
 writelist(['"', Mode, '"', ': Invalid mode.']),
 retract(error(_, _, _)),
 assert(error(range, nul, nul)).

gettarget($, [$, ' '|$], $, $):-
 writelist(['Path separator is: ', '"', /, '"', '.']), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).      %wrong separator
gettarget(Mode, [Target], Target, Cwd):-     %target is file in cwd
 file(Target, Cwd, $),
 changeit(Mode, Target, Cwd).
gettarget(Mode, [Target], Target, Cwd):-     %target is a dir in cwd
 dir(Target, $, Cwd, $),
 changeit(Mode, Target, Cwd).
gettarget(Mode, [Target], Target, Cwd):-     %target is parent of cwd
 dir(Cwd, $, Target, $),
 changeit(Mode, Target, Cwd).
gettarget(Mode, [Head, '/'|Rest], Target, Source):-   %trace down
 dir(Head, $, Source, $),
 gettarget(Mode, Rest, Target, Head).
gettarget(Mode, [Head, '/'|Rest], Target, Source):-   %trace up
 dir(Source, $, Head, $),
 gettarget(Mode, Rest, Target, Head).
gettarget($, [Head|$], $, Cwd):-
 writelist(['"', Head, '"', ' : not found.']),
 retract(error(_, _, _)),
 assert(error(reference, Cwd, Head)).

changeit($, root, $):-      %can't alter root
 write('Permission denied. Not owner'),
 retract(error(_, _, _)),
 assert(error(execute, nul, nul)).
changeit($, home, $):-
 write('Permission denied. Not owner'),    %can't alter home
 retract(error(_, _, _)),
 assert(error(execute, nul, nul)).
changeit(Mode, Target, Cwd):-          %change file mode
 file(Target, Cwd, Old),
 retract(file(Target, Cwd, Old)),
 assert(file(Target, Cwd, Mode)).
changeit(Mode, Target, Cwd):-          %change dir mode
 dir(Target, Old, Cwd, $),
 retract(dir(Target, Old, Cwd, Contents)),
 assert(dir(Target, Mode, Cwd, Contents)).
changeit($, $, $).

/*
cp/1 deals with the arguments for the copy command. the important calls
are to findarg/7 which traces the path references to the target and
destination of the command and checks for valid reference at each level.
The changing Cwd is passed on for checking purposes and the original Cwd
is saved in Saver for the start of the second argument trace. Source is
the parent directory of Target and Dest is the parent of New which may
be a new name or an existing entity which is then overwritten.*/

cp([]):-
 write('Usage : cp [-i] <path> <path>'), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
cp([' ', Lonely|[]]):-
```

```
 write('Usage : cp [-i] <path> <path>'), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
cp([' '|Arguments]):-
 cwd(Cwd),
 findarg1(Arguments, Source, Target, Dest, New, Cwd, Cwd).

findarg1([Target, ' '|Second], Cwd, Target, Dest, New, Cwd, Saver):-
%end of 1st arg
 file(Target, Cwd, $),          %is target a file?
 findarg2(Second, Cwd, Target, Dest, New, Saver).   %get dest
findarg1([Target, ' '|$], $, $, $, $, Cwd, $):-      %target is child
dir
 dir(Target, $, Cwd, $),
 write('Cannot copy a directory.'), nl,
 retract(error(_, _, _)),
 assert(error(logical, _, _)).
findarg1([Target, ' '|$], $, $, $, $, Cwd, $):-      %target is parent
dir
 dir(Cwd, $, Target, $),
 write('Cannot copy a directory.'), nl,
 retract(error(_, _, _)),
 assert(error(logical, _, _)).
findarg1([Head, '/'|Rest], Source, Target, Dest, New, Cwd, Saver):-
%trace down
 dir(Head, $, Cwd, $),
 findarg1(Rest, Source, Target, Dest, New, Head, Saver).
findarg1([Head, '/'|Rest], Source, Target, Dest, New, Cwd, Saver):-
%trace up
 dir(Cwd, $, Head, $),
 findarg1(Rest, Source, Target, Dest, New, Head, Saver).
findarg1([Head|$], $, $, $, $, Cwd, $):-
 writelist(['"', Head, '"', ': not found.']), nl,
 retract(error(_, _, _)),
 assert(error(reference, Cwd, Head)).

findarg2([$, ' '|$], $, $, $, $, $):-
 writelist(['Path separator is: ', '"', /, '"', '.']), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
findarg2([New], Source, New, Source, New, $):- %attempted use same
 write('You cannot copy a file onto itself .'), nl,   %name
 retract(error(_, _, _)),
 assert(error(logical, Source, New)).
findarg2([New], Source, New, Dest, New, $):-  %attempt to use same
 write('Copy must have new name.'), nl,     %name diff dir for copy
 retract(error(_, _, _)),
 assert(error(logical, Dest, New)).
findarg2([New], Source, Old, Cwd, New, Cwd):-      %new is child dir
 dir(New, $, Cwd, $),          %don't cp
 write('Copy must have new name.'), nl,
 retract(error(_, _, _)),
 assert(error(logical, Cwd, New)).
findarg2([New], Source, Old, Cwd, New, Cwd):-    %new is parent dir
 dir(Cwd, $, New, $),          %don't cp
 write('Copy must have new name.'), nl,
 retract(error(_, _, _)),
 assert(error(logical, New, Cwd)).
findarg2([New], Source, Old, Dest, New, Cwd):-  %New is child file
 file(New, Cwd, $),          %don't cp
 writelist(['A file called ', '"', New, '"', ' already exists in ',
'"', Cwd, '"', '.']), nl,
 retract(error(_, _, _)),
```

```
 assert(error(duplication, Cwd, New)).
findarg2([New], Source, Old, Dest, New, Cwd):-
 file(New, Parent, $),
 writelist(['A file called ', '"', New, '"', ' already exists in ',
'"', Parent, '"', '.']), nl, %file exists globally
 retract(error(_, _, _)),
 assert(error(duplication, Parent, New)).
findarg2([New], Source, Old, Dest, New, Cwd):-
 dir(New, $, Parent, $),
 writelist(['There is a directory called ', '"', New, '"', '.']), nl,
  %dir exists globally
 retract(error(_, _, _)),
 assert(error(duplication, Parent, Head)).
findarg2([New], Source, Old, Cwd, New, Cwd):-      %no duplication
 copy(Source, Old, Cwd, New).
findarg2([Head, '/'|Rest], Source, Old, Dest, New, Cwd):-   %trace down
 dir(Head, $, Cwd, $),
 findarg2(Rest, Source, Old, Dest, New, Head).
findarg2([Head, '/'|Rest], Source, Old, Dest, New, Cwd):-   %trace up
 dir(Cwd, $, Head, $),
 findarg2(Rest, Source, Old, Dest, New, Head).
findarg2([Head|$], Source, Old, Dest, New, Cwd):-     %bad arg
 writelist(['"', Head, '"', ': not found.']), nl,
 retract(error(_, _, _)),
 assert(error(reference, Cwd, Head)).

copy(Source, Old, Dest, New):-
 dir(Old, Protection, Source, Contents),
 xperm(Source),      %check for protection
 xperm(Dest),
 rperm(Old),
 wperm(Dest),
 assert(dir(New, 700, Dest, Contents)),     %copy item
 dir(Dest, $, $, Conts),
 add(New, Conts, Newconts),
 retract(dir(Dest, Protect, Mum, Conts)),
 assert(dir(Dest, Protect, Mum, Newconts)).
copy(Source, Old, Dest, New):-
 xperm(Source),    %check protection
 xperm(Dest),
 wperm(Dest),
 rperm(Old),
 assert(file(New, Dest, 700)),
 dir(Dest, $, $, Conts),
 add(New, Conts, Newconts),
 retract(dir(Dest, Protect, Mum, Conts)),
 assert(dir(Dest, Protect, Mum, Newconts)).
copy($, $, $, $):-
 modefail(yes),       %permission denied
 retract(modefail(yes)).

/*
input/1 takes the keyboard input at the character level and converts
it into a list of words which form the command and its arguments.
*/

input(Commandline):-
 get0(Char),
 getrest(Char, Commandline).

getrest(13, []):- !.     /*carriage return*/
getrest(32, [Word|Commandline]):- !,
 getchars(32, Letters, Nextchar),
```

```
 name(Word, [32]),    /*space character*/
 input(Commandline).
getrest(47, [Word|Commandline]):- !,
 getchars(47, Letters, Nextchar),
 name(Word, [47]),    /* '/' character*/
 input(Commandline).
getrest(Letter, [Word|Commandline]):-
 getchars(Letter, Letters, Nextchar),
 name(Word, Letters),
 getrest(Nextchar, Commandline).

getchars(13, [], 13):-!.    /*end of word = return*/
getchars(32, [], 32):-!.    /*end of word = space */
getchars(47, [], 47):-!.    /*end of word = '/'  */
getchars(Let, [Let|Letters], Nextchar):-
 get0(Char),
 getchars(Char, Letters, Nextchar).

/*
lastcdcheck/1 gets the previous working directory. If Erroneous is in
the contents of it then last cd command is considered responsible for
the  error and the user is notified of this.
*/

lastcdcheck :-
 prewd(Pwd, DoW, Day, Month, Year, Hour, Min, Sec),
 error($, Source, Erroneous),
 dir(Pwd, $, $, Contents),    %look in contents of last working dir
 isin(Contents, Erroneous),   %is bad argument in there?
 cwd(Cwd),
 timelapse(Dnow, Hnow, Mnow, Snow, Day, Hour, Min, Sec, Dlapse, Hlapse,
Mlapse, Slapse),
 sysbeep(10), nl(2),
 write('*****************************************************'), nl,
 write('That Command may have failed'), nl,
 write('because you changed directory'), nl,
 writelist(['from ', '"', Pwd, '"', ' to ', '"', Cwd, '"']), nl,
 %writelist(['on : ', DoW, ' ', Day, ' ', Month, ' ']),
 %writelist(['at : ', Hour, ':', Min, ':', Sec]), nl,
 writelist(['(', Hlapse, ' hours ', Mlapse, ' minutes ',
  Slapse, ' seconds ago.)']), nl,
 writelist(['"', Pwd, '"', ' contains the item ', '"', Erroneous,
'"']), nl,
 write('*****************************************************'), nl(2).

/*
lastexecheck/1 searches the log file for the last chmod command
issued by the user which had Erroneous as its argument and removed
execute permission. If found, that command is responsible for the error
and the user is notified of this.
*/

lastexecheck :-
 execlear,                                    %remove old lastexes
 assert(lastexe(monday, 1, january, 1904,12,0,0)),    %insert a dummy
fact
 event([chmod, ' ', Mode, ' '|Argument], Pwd, Dow, Day, Month, Year,
Hour, Min, Sec),
 isin([000,200,400,600], Mode),
 error($, $, Erroneous),
 isexematch(Argument, Erroneous),
 retract(lastexe(_, _, _, _, _, _, _)),
 assert(lastexe(Dow, Day, Month, Year, Hour, Min, Sec)),
```

```
 lastexecheck1.
lastexecheck1 :-
 lastexe(monday, 1, january, 1904,12,0,0).          %no candidates found
lastexecheck1 :-                                  %succeeds when one found
 lastexe(DoW, Day, Month, Year, Hour, Min, Sec), nl,
 timelapse(Dnow, Hnow, Mnow, Snow, Day, Hour, Min, Sec, Dlapse, Hlapse,
Mlapse, Slapse),
 error($, $, Erroneous),
 sysbeep(10), nl(2),
 write('*****************************************'), nl,
 write('That Command failed because'), nl,
 write('the execute permission for'), nl,
 writelist(['"', Erroneous, '"', ' was removed']), nl,
 writelist(['on : ', DoW, ' ', Day, ' ', Month, ' ']),
 writelist(['at : ', Hour, ':', Min, ':', Sec]), nl,
 writelist(['(', Dlapse, ' days ', Hlapse, ' hours ', Mlapse, ' minutes
',
   Slapse, ' seconds ago.)']), nl,
 %write(['Use chmod to gain execute permission.']), nl,
 write('*****************************************'), nl(2).

execlear :-
 execlearit.
execlear.


execlearit :-
 retract(lastexe(_, _, _, _, _, _, _)),
 fail.

isexematch([Erroneous|[]], Erroneous).
isexematch([Head|Tail], Erroneous):-
isexematch(Tail, Erroneous).

/*
lastmkmvcpcheck/2 searches the log file for the last rm, rmdir or mv
command issued by the user whose target was Source/Erroneous. Any such
command is considered possibly responsible for the duplication error and
the user is notified of this.
*/

lastmkmvcpcheck :-
 mkmvcpclear,       %clear and insert a dummy candidate
 assert(lastmkmvcp(command, dest, target, monday, 1, january,
1904,12,0,0)),
 event([Command|Argument], Cwd, Dow, Day, Month, Year, Hour, Min, Sec),
 isin([cp, mv, mkfile, mkdir], Command),
 matchit1(Argument, Parent, Arg1, Dest, Arg2, Cwd, Cwd), %retrieve the
event args
 error($, Source, Erroneous),
 isitvalid(Source, Erroneous, Parent, Arg1, Dest, Arg2),  %does arg
match error?
 retract(lastmkmvcp(_, _, _, _, _, _, _, _, _, _)),
 assert(lastmkmvcp(Command, Dest, Arg1, Dow, Day, Month, Year, Hour,
Min, Sec)),
 lastmkmvcpcheck1.
lastmkmvcpcheck1 :-        %only dummy found
 lastmkmvcp(command, dest, target, monday, 1, january, 1904,12,0,0).
lastmkmvcpcheck1 :-
 lastmkmvcp(Command, Dest, Arg1, Dow, Day, Month, Year, Hour, Min,
Sec),
 isin([mv, cp], Command),
 timelapse(Dnow, Hnow, Mnow, Snow, Day, Hour, Min, Sec, Dlapse, Hlapse,
```

```
Mlapse, Slapse),
 error($, Source, Erroneous),
 sysbeep(10), nl(2),
write('*****************************************'), nl,
 write('That command may have failed'), nl,
 write('because '),
 writelist(['"', Source, '/', Arg1, '"']),
 writelist(['was ', Command, '-ed']), nl,
 writelist(['to ', '"', Dest, '/', Erroneous, '"']), nl,
 writelist(['on : ', Dow , ' ', Day, ' ', Month, ' ']), nl,
 writelist(['at : ', Hour, ':', Min, ':', Sec]), nl,
 writelist(['(', Dlapse, ' days ', Hlapse, ' hours ', Mlapse, ' minutes
',
   Slapse, ' seconds ago.)']), nl,
 write('*****************************************'), nl(2).

lastmkmvcpcheck1 :-
lastmkmvcp(Command, Dest, Target, Dow, Day, Month, Year, Hour, Min,
Sec),
 timelapse(Dnow, Hnow, Mnow, Snow, Day, Hour, Min, Sec, Dlapse, Hlapse,
Mlapse, Slapse),
 error($, Source, Erroneous),
 sysbeep(10), nl(2),
 write('*****************************************'), nl,
 write('That command may have failed because'), nl,
 writelist(['"', Dest, '/', Erroneous, '"']),
 write('was created '), nl,
 writelist(['on ', Dow, ' ', Day, ' ', Month, ' ']), nl,
 writelist(['at ', Hour, ':', Min, ':', Sec]), nl,
 writelist(['(', Dlapse, ' days ', Hlapse, ' hours ', Mlapse, ' minutes
',
   Slapse, ' seconds ago.)']), nl,
 write('*****************************************'), nl(2).
lastmkmvcpcheck1.        %error not diachronic

matchit1([' '|Rest], Parent, Arg1, Dest, Arg2, Pwd, Saver):-   %remove
space
 matchit1(Rest, Parent, Arg1, Dest, Arg2, Pwd, Saver).
matchit1([Arg1], Pwd, Arg1, Pwd, Arg1, Pwd, $).    %single path argument
matchit1([Arg1, ' '|Second ], Pwd, Arg1, Dest, Arg2, Pwd, Saver):- %two
path argument
 matchit2(Second, Pwd, Arg1, Dest, Arg2, Saver).
matchit1([Head, '/'|Rest], Parent, Arg1, Dest, Arg2, Pwd, Saver):-
%trace path
 matchit1(Rest, Parent, Arg1, Dest, Arg2, Head, Saver).

matchit2([Arg2], Parent, Arg1, Pwd, Arg2, Pwd).    %end of second path
matchit2([Head, '/'|Tail], Parent, Arg1, Dest, Arg2, Pwd):-   %trace
path
 matchit2(Tail, Parent, Arg1, Pwd, Arg2, Head).

isitvalid(Source, Erroneous, Dest, Erroneous, Dest, Erroneous).      %mk
command culprit
isitvalid($, Erroneous, $, $, $, Erroneous).     %mv/cp culprit
isitvalid($, Erroneous, $, Erroneous, $, Source).    %mv culprit

mkmvcpclear :-
 mkmvcpclearit.
 mkmvcpclear.

mkmvcpclearit :-
 retract(lastmkmvcp(_, _, _, _, _, _, _, _, _, _)),
 fail.
```

```
/*
lastreadcheck/1 searches the log file for the last chmod command
issued by the user which had Erroneous as its argument and removed read
permission. If found, that command is responsible for the error and the
user is notified of this.
*/

lastreadcheck :-
 readclear,        %remove old lastreads
 assert(lastread(monday, 1, january, 1904,12,0,0)),   %insert a dummy
fact
 event([chmod, ' ', Mode, ' '|Argument], Pwd, Dow, Day, Month, Year,
Hour, Min, Sec),
 isin([000,100,200,300], Mode),
 error($, $, Erroneous),
 isexematch(Argument, Erroneous),
 retract(lastread(_, _, _, _, _, _, _)),
 assert(lastread(Dow, Day, Month, Year, Hour, Min, Sec)),
 lastreadcheck1.
lastreadcheck1 :-
 lastread(monday, 1, january, 1904,12,0,0).   %no candidates found
lastreadcheck1 :-       %succeeds when no more found
 lastread(DoW, Day, Month, Year, Hour, Min, Sec), nl,
timelapse(Dnow, Hnow, Mnow, Snow, Day, Hour, Min, Sec, Dlapse, Hlapse,
Mlapse, Slapse),
 error($, $, Erroneous),
 sysbeep(10), nl(2),
write('*****************************************'), nl,
 write('That Command failed because'), nl,
 write('the read permission for '),
 writelist(['"', Erroneous, '"']), nl,
 write('was removed on: '),
 writelist([DoW, ' ', Day, ' ', Month, ' ']),
 writelist(['at : ', Hour, ':', Min, ':', Sec]), nl,
 writelist(['(', Dlapse, ' days ', Hlapse, ' hours ', Mlapse, ' minutes
',
   Slapse, ' seconds ago.)']), nl,
 %write('Use chmod to gain read permission.'), nl,
write('*****************************************'), nl(2).

readclear :-
 readclearit.
 readclear.

readclearit :-
 retract(lastread(_, _, _, _, _, _, _)),
 fail.

/*
lastrmvcheck/2 searches the log for the last rm, rmdir or mv command
issued by the user whose target was Source/Erroneous. Any such command
is considered possibly responsible for the reference error and the user
is notified of this.
*/

lastrmvcheck :-
 rmclear,       %clear and insert a dummy fact
 assert(lastrmv(command, dest, target, monday, 1, january,
1904,12,0,0)),
 event([Command|Argument], Cwd, Dow, Day, Month, Year, Hour, Min, Sec),
 isin([mv, rm, rmdir], Command),
 error($, Source, Erroneous),
```

```
 matchit1(Argument, Parent, Arg1, Dest, Arg2, Cwd, Cwd),   %get the
event args
 wasitthis(Source, Erroneous, Parent, Arg1),
 retract(lastrmv(_, _, _, _, _, _, _, _, _, _)),
 assert(lastrmv(Command, Dest, Arg2, Dow, Day, Month, Year, Hour, Min,
Sec)),
 lastrmvcheck1.        %search for others
lastrmvcheck1 :-
 lastrmv(command, dest, target, monday, 1, january, 1904,12,0,0), !,
 fail.                  %none found force cd check
lastrmvcheck1 :-
 error($, Source, Erroneous),
 lastrmv(mv, Dest, Target, Dow, Day, Month, Year, Hour, Min, Sec),
 timelapse(Dnow, Hnow, Mnow, Snow, Day, Hour, Min, Sec, Dlapse, Hlapse,
Mlapse, Slapse),
 sysbeep(10), nl(2),
 write('*****************************************'), nl,
 write('That command may have failed because'), nl,
 writelist(['"', Source, '/', Erroneous, '"', ' was moved']), nl,
 writelist(['to ', '"', Dest, '/', Target, '"']), nl,
 writelist(['on : ', Dow , ' ', Day, ' ', Month, ' ']),
 writelist(['at : ', Hour, ':', Min, ':', Sec]), nl,
 writelist(['(', Dlapse, ' days ', Hlapse, ' hours ', Mlapse, ' minutes
',
   Slapse, ' seconds ago.)']), nl,
 %write('The item may be accessed there.'), nl,
 write('*****************************************'), nl(2).
lastrmvcheck1 :-
 error($, Source, Erroneous),
 lastrmv(Command, Dest, Target, Dow, Day, Month, Year, Hour, Min, Sec),
 timelapse(Dnow, Hnow, Mnow, Snow, Day, Hour, Min, Sec, Dlapse, Hlapse,
Mlapse, Slapse),
 sysbeep(10), nl(2),
 write('*****************************************'), nl,
 write('That command may have failed because'), nl,
 writelist(['"', Source, '/', Erroneous, '"', 'was removed']), nl,
 writelist([' on ', Dow, ' ', Day, ' ', Month, ' ']),
 writelist(['at ', Hour, ':', Min, ':', Sec]), nl,
 writelist(['(', Dlapse, ' days ', Hlapse, ' hours ', Mlapse, ' minutes
',
   Slapse, ' seconds ago.)']), nl(2),
 write('*****************************************'), nl(2).

wasitthis(Source, Erroneous, Source, Erroneous).  %arg1 was target of rm
wasitthis(Source, Erroneous, Source, Erroneous).  %arg1 was target of mv

rmclear :-
 rmclearit.
rmclear.

rmclearit :-
 retract(lastrmv(_, _, _, _, _, _, _, _, _, _)),
 fail.

/*
lastwritecheck/1 searches the log file for the last chmod command
issued by the user which had Erroneous as its argument and removed write
permission. If found, that command is responsible for the error and the
user is notified of this.
*/

lastwritecheck :-
 writeclear,        %remove old lastwrites
```

```
 assert(lastwrite(monday, 1, january, 1904,12,0,0)),   %insert a dummy
fact
 event([chmod, ' ', Mode, ' '|Argument], Pwd, Dow, Day, Month, Year,
Hour, Min, Sec),
 isin([000,100,400,500], Mode),
 error($, $, Erroneous),
 isexematch(Argument, Erroneous),
 retract(lastwrite(_, _, _, _, _, _, _)),
 assert(lastwrite(Dow, Day, Month, Year, Hour, Min, Sec)),
 lastwritecheck1.
lastwritecheck1 :-
 lastwrite(monday, 1, january, 1904,12,0,0).   %no candidates found
lastwritecheck1 :-       %succeeds when no more found
 lastwrite(DoW, Day, Month, Year, Hour, Min, Sec), nl,
 timelapse(Dnow, Hnow, Mnow, Snow, Day, Hour, Min, Sec, Dlapse, Hlapse,
Mlapse, Slapse),
 sysbeep(10), nl(2),
 write('*****************************************'), nl,
 error($, $, Erroneous),
 write('That Command failed because'), nl,
 write('the write permission for '), nl,
 writelist(['"', Erroneous, '"']),
 write('was removed '), nl,
 writelist(['on : ', DoW, ' ', Day, ' ', Month, ' ']),
 writelist(['at : ', Hour, ':', Min, ':', Sec]), nl,
 writelist(['(', Dlapse , ' days ', Hlapse, ' hours ', Mlapse, '
minutes ',
   Slapse, ' seconds ago.)']), nl,
 %write('Use chmod to gain write permission.'), nl,
 write('*****************************************'), nl(2).

writeclear :-
 writeclearit.
writeclear.

writeclearit :-
 retract(lastwrite(_, _, _, _, _, _, _)),
 fail.

/*
ls-l/1 lists the specified directory contents in long form after
checking the correctness of the command.
*/

ltracepath([], $):-
 ls([]).
ltracepath([$, ' '|$], $):-
 writelist(['Path separator is: ', '"', /, '"', '.']), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
ltracepath([File], Source):-
 file(File, Source, $),
 writelist(['"', File, '"', ' is a file.']),
 retract(error(_, _, _)),
 assert(error(logical, $, $)).
ltracepath([Dir], Source):-       %target is in source
 dir(Dir, $, Source, Contents),
 rperm(Dir),
 xperm(Dir),
 longlistconts(Contents, Dir).
ltracepath([Dir], Source):-       %target parent of source
 dir(Source, $, Dir, $),
 dir(Dir, $, $, Contents),
```

```
 rperm(Dir),
 xperm(Dir),
 longlistconts(Contents, Dir).
ltracepath([Dir, '/'|Tail], Source):-     %trace down
 dir(Dir, $, Source, $),
 ltracepath(Tail, Dir).
ltracepath([Dir, '/'|Tail], Source):-     %trace up
 dir(Source, $, Dir, $),
 ltracepath(Tail, Dir).
ltracepath(Path, $):-
 modefail(yes),
 retract(modefail(yes)).
ltracepath([Head|$], Source):-          %bad arg
 writelist(['"', Head, '"', ': not found.']), nl,
 retract(error(_, _, _)),
 assert(error(reference, Source, Head)).

/*
ls/1 lists the specified directory contemts after checking that the
command is correct.
*/

ls([]):-
 cwd(Cwd),       %list cwd
 dir(Cwd, $, $, Contents),
 rperm(Cwd),
 xperm(Cwd),
 listconts(Contents).
ls([]):-
 modefail(yes),
 retract(modefail(yes)).
ls([' ', '-l']):-      %longlist cwd
 cwd(Cwd),
 dir(Cwd, $, $, Contents),
 rperm(Cwd),
 xperm(Cwd),
 longlistconts(Contents, Cwd).
ls([' ', '-l']):-
 modefail(yes),
 retract(modefail(yes)).
ls([' ', '-l', ' '|Path]):-    %loglist other dir
 cwd(Cwd),
 ltracepath(Path, Cwd).
ls([' '|Path]):-      %list other dir
 cwd(Cwd),
 tracepath(Path, Cwd).

tracepath([], $):-
 ls([]).
tracepath([$, ' '|$], $):-
 writelist(['Path separator is: ', '"', /, '"', '.']), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
tracepath([File], Source):-
 file(File, Source, $),
 writelist(['"', File, '"', ' is a file.']),
 retract(error(_, _, _)),
 assert(error(logical, nul, nul)).
tracepath([Dir], Source):-       %target is in source
 dir(Dir, $, Source, Contents),
 rperm(Dir),
 xperm(Dir),
 listconts(Contents).
```

```
tracepath([Dir], Source):-       %target parent of source
 dir(Source, $, Dir, $),
 dir(Dir, $, $, Contents),
 rperm(Dir),
 xperm(Dir),
 listconts(Contents).
tracepath([Dir, '/'|Tail], Source):-      %trace down
 dir(Dir, $, Source, $),
 tracepath(Tail, Dir).
tracepath([Dir, '/'|Tail], Source):-      %trace up
 dir(Source, $, Dir, $),
 tracepath(Tail, Dir).
tracepath(Path, $):-
 modefail(yes),
 retract(modefail(yes)).
tracepath([Head|$], Source):-          %bad argument
 writelist(['"', Head, '"', ': not found.']), nl,
 retract(error(_, _, _)),
 assert(error(reference, Source, Head)).

/*
mkdir/1 creates a new directory with the specified name at the
specified location. The command is checked for errors which are asserted
so that they can be anaysed by the analyser.
*/

mkdir([]):-
 write('Usage : mkdir <dirname>'), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
mkdir([' '|Arguments]):-
 cwd(Cwd),
 makedir(Arguments, Cwd).
makedir([$, ' '|$], $):-
 writelist(['Path separator is: ', '"', /, '"', '.']), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
makedir([Newdir], Dir):-
 dir(Newdir, $, $, $),
 write('Directory exists.'), nl,
 retract(error(_, _, _)),
 assert(error(duplication, Dir, Newdir)).
makedir([Newdir], Dir):-
 file(Newdir, Parent, $),
 write('File exists.'), nl,
 retract(error(_, _, _)),
 assert(error(duplication, Parent, Newdir)).
makedir([Newdir], Dir):-       %make newdir in dir
 xperm(Dir),
 wperm(Dir),
 assert(dir(Newdir, 700, Dir, [])),
 dir(Dir, $, $, Contents),
 add(Newdir, Contents, Newcontents),
 retract(dir(Dir, Protection, Parent, Contents)),
 assert(dir(Dir, Protection, Parent, Newcontents)).
makedir([Head, '/'|Rest], Cwd):-      %trace down
 dir(Head, $, Cwd, $),
 makedir(Rest, Head).
makedir([Head, '/'|Rest], Cwd):-      %trace up
 dir(Cwd, $, Head, $),
 makedir(Rest, Head).
makedir($, $):-
 modefail(yes),
```

```
 retract(modefail(yes)).

makedir([Head|$], Cwd):-
 writelist(['"', Head, '"', ': not found.']), nl,
 retract(error(_, _, _)),
 assert(error(reference, Cwd, Head)).

/*
mkfile/1 creates a file with the given name in the specified directory
or in Cwd if none is specified. The command is first checked for errors.
*/

mkfile([]):-
 write('Usage : mkfile file...'),
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
mkfile([' '|Argument]):-
 cwd(Cwd),
 makefile(Argument, Cwd).
makefile([$, ' '|$], $):-
 writelist(['Path separator is: ', '"', /, '"', '.']), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
makefile([File], Dir):-
 file(File, Parent, $),
 write('File exists.'), nl,
 retract(error(_, _, _)),
 assert(error(duplication, Parent, File)).
makefile([Dir], Cwd):-
 dir(Dir, $, Parent, $),
 write('Directory exists.'), nl,
 retract(error(_, _, _)),
 assert(error(duplication, Parent, Dir)).
makefile([File], Dir):-        %make file in dir
 xperm(Dir),
 wperm(Dir),
 dir(Dir, Mode, Parent, Contents),
 assert(file(File, Dir, 700)),
 add(File, Contents, Newcontents),
 retract(dir(Dir, Mode, Parent, Contents)),
 assert(dir(Dir, Mode, Parent, Newcontents)).
makefile([Head, '/'|Rest], Cwd):-       %trace down
 dir(Head, $, Cwd, $),
 makefile(Rest, Head).
makefile([Head, '/'|Rest], Cwd):-       %trace up
 dir(Cwd, $, Head, $),
 makefile(Rest, Head).
makefile($, $):-
 modefail(yes),
 retract(modefail(yes)).


makefile([Head|$], Cwd):-
 writelist(['"', Head, '"', ': not found.']), nl,
 retract(error(_, _, _)),
 assert(error(reference, Cwd, Head)).


/*
?perm/1 checks if there is protection on the target of the command. If
there is it fails the command and asserts this so that the servers
know not to backtrack
*/
```

```
xperm(Target):-
 file(Target, $, Mode),
 isin([100,300,500,700], Mode).
xperm(Target):-
 dir(Target, Mode, $, $),
 isin([100,300,500,700], Mode).
xperm(Target):-
 file(Target, Parent, $),
 xreject(Target, Parent).
xperm(Target):-
 dir(Target, $, Parent, $),
 xreject(Target, Parent).

xreject(Target, Parent):-
 not modefail(yes),
 write('Permission denied.'), nl,
 assert(modefail(yes)),
 retract(error(_, _, _)),
 assert(error(execute, Parent, Target)), !,
 fail.

rperm(Target):-
 file(Target, $, Mode),
 isin([400,500,600,700], Mode).
rperm(Target):-
 dir(Target, Mode, $, $),
 isin([400,500,600,700], Mode).
rperm(Target):-
 file(Target, Parent, $),
 rreject(Target, Parent).
rperm(Target):-
 dir(Target, $, Parent, $),
 rreject(Target, Parent).

rreject(Target, Parent):-
 not modefail(yes),
 write('Permission denied.'), nl,
 assert(modefail(yes)),
 retract(error(_, _, _)),
 assert(error(read, Parent, Target)), !,
 fail.

wperm(Target):-
 file(Target, $, Mode),
 isin([200,300,600,700], Mode).
wperm(Target):-
 dir(Target, Mode, $, $),
 isin([200,300,600,700], Mode).
wperm(Target):-
 file(Target, Parent, $),
 wreject(Target, Parent).
wperm(Target):-
 dir(Target, $, Parent, $),
 wreject(Target, Parent).

wreject(Target, Parent):-
 not modefail(yes),
 write('Permission denied.'), nl,
 assert(modefail(yes)),
 retract(error(_, _, _)),
 assert(error(write, Parent, Target)), !,
 fail.
```

```
/*
mv/1 deals with the arguments for the move command. the important calls
are to getarg(1/2)/7 which traces the path references to the target and
destination of the command and checks for valid reference at each level.
The changeing Cwd is passed on for checking purposes and the original
Cwd is saved in Saver for the start of the second argument trace. Source
is the parent directory of Target and Dest is the parent of New which
may be a new name or an existing entity which is then overwritten.
*/

mv([]):-
 write('Usage : mv [-i] file/dir dir'), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
mv([' ', Lonely|[]]):-
 write('Usage : mv [-i] file/dir dir'), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
mv([' '|Arguments]):-
 cwd(Cwd),
 getarg1(Arguments, Source, Target, Dest, New, Cwd, Cwd).

getarg1([Target, ' '|Rest], Source, Target, Dest, New, Source, Saver):-
 %targ is file
 file(Target, Source, $),
 getarg2(Rest, Source, Target, Dest, New, Saver).
getarg1([Target, ' '|Rest], Source, Target, Dest, New, Source, Saver):-
 %targ is child
 dir(Target, $, Source, $),
 write('Cannot move a directory.'), nl.
getarg1([Target, ' '|Rest], Source, Target, Dest, New, Source, Saver):-
 %targ is parent
 dir(Source, $, Target, $),
 write('Cannot move a directory.'), nl.
getarg1([Head, '/'|Rest], Source, Target, Dest, New, Cwd, Saver):-
%trace down
 dir(Head, $, Cwd, $),
 getarg1(Rest, Source, Target, Dest, New, Head, Saver).
getarg1([Head, '/'|Rest], Source, Target, Dest, New, Cwd, Saver):-
%trace up
 dir(Cwd, $, Head, $),
 getarg1(Rest, Source, Target, Dest, New, Head, Saver).
getarg1([Head|$], Source, Target, Dest, New, Cwd, Saver):-    %bad arg
 writelist(['"', Head, '"', ': not found.']), nl,
 retract(error(_, _, _)),
 assert(error(reference, Cwd, Head)).

getarg2([$, ' '|$], $, $, $, $, $):-
 writelist(['Path separator is: ', '"', /, '"', '.']), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
getarg2([New], Source, Old, Dest, New, Dest):-    %local file dup
 file(New, Dest, $),                              %prevent mv
 write('File exists.'), nl,
 retract(error(_, _, _)),
 assert(error(duplication, Dest, New)).
getarg2([New], Source, Old, Dest, New, Dest):-    %move File to extant
child
 dir(New, $, Dest, $),                            %dir
 move(Source, Old, New, Old).
getarg2([New], Source, Old, Dest, New, Dest):-    %move File to extant
parent
```

```
 dir(Dest, $, New, $),                          %dir
 move(Source, Old, New, Old).
getarg2([New], Source, Old, Dest, New, Dest):-    %global file exists
 file(New, $, $),
 sortdupfile(Source, Old, Dest, New).
getarg2([New], Source, Old, Dest, New, Dest):-    %global dir exists
 dir(New, $, $, $),
 sortdupdir(Source, Old, Dest, New).
getarg2([New], Source, Old, Dest, New, Dest):-    %no duplication
 move(Source, Old, Dest, New).
getarg2([Head, '/'|Rest], Source, Old, Dest, New, Cwd):- %trace down
 dir(Head, $, Cwd, $),
 getarg2(Rest, Source, Old, Dest, New, Head).
getarg2([Head, '/'|Rest], Source, Old, Dest, New, Cwd):- %trace up
 dir(Cwd, $, Head, $),
 getarg2(Rest, Source, Old, Dest, New, Head).
getarg2([Head|$], Source, Old, Dest, New, Cwd):-  %bad argument
 writelist(['"', Head, '"', ': not found.']), nl,
 retract(error(_, _, _)),
 assert(error(reference, Cwd, Head)).

sortdupfile(Source, Old, Dest, New):-     %dup is the moved file
 file(New, Source, $),
 move(Source, Old, Dest, New).
sortdupfile(Source, Old, Dest, New):-     %dup is not moved file
 file(New, Parent, $),
 writelist(['"', New, '"', ': exists.']), nl,
 retract(error(_, _, _)),
 assert(error(duplication, Parent, New)).

sortdupdir(Source, Old, Dest, New):-      %dup is the moved dir
 dir(New, $, Source, $),
 move(Source, Old, Dest, New).
sortdupdir(Source, Old, Dest, New):-      %dup is not the moved dir
 dir(New, $, Parent, $),
 writelist(['"', New, '"', ': exists.']), nl,
 retract(error(_, _, _)),
 assert(error(duplication, Parent, New)).

move(Source, Old, Dest, New):-
 isancestor(Old, Dest),
 write('Destination is a descendant.'), nl,
 retract(error(_, _, _)),
 assert(error(logical, nul, nul)).

move(Source, Old, Source, New):-          %rename dir
 dir(Old, $, Source, $),
 xperm(Source),
 wperm(Source),
 retract(dir(Old, Protection, Source, Contents)),
 assert(dir(New, 700, Source, Contents)),
 dir(Source, $, $, Contnts),
 delete(Old, Contnts, Newcontnts),
 add(New, Newcontnts, Newconts),
 retract(dir(Source, Protect, Mum, Contnts)),
 assert(dir(Source, Protect, Mum, Newconts)),
 tellkids(Contents, New).
move(Source, Old, Dest, New):-            %move dir
 dir(Old, $, Source, $),
 xperm(Source),
 xperm(Dest),
 wperm(Dest),
 retract(dir(Old, Protection, Source, Contents)),
```

```
  assert(dir(New, 700, Dest, Contents)),
  dir(Source, $, $, Contnts),
  dir(Dest, $, $, Conts),
  delete(Old, Contnts, Newcontnts),
  add(New, Conts, Newconts),
  retract(dir(Source, Prot, Dad, Contnts)),
  assert(dir(Source, Prot, Dad, Newcontnts)),
  retract(dir(Dest, Protect, Mum, Conts)),
  assert(dir(Dest, Protect, Mum, Newconts)),
  tellkids(Contents, New).
move(Source, Old, Source, New):-        %rename file
  xperm(Source),
  wperm(Source),
  retract(file(Old, Source, Protection)),
  assert(file(New, Source, 700)),
  dir(Source, $, $, Cntnts),
  delete(Old, Cntnts, Newcntnts),
  add(New, Newcntnts, Newconts),
  retract(dir(Source, Prot, Dad, Cntnts)),
  assert(dir(Source, Prot, Dad, Newconts)).
move(Source, Old, Dest, New):-          %move file
  xperm(Source),
  xperm(Dest),
  wperm(Dest),
  retract(file(Old, Source, Protection)),
  assert(file(New, Dest, 700)),
  dir(Source, $, $, Cntnts),
  dir(Dest, $, $, Conts),
  delete(Old, Cntnts, Newcntnts),
  add(New, Conts, Newconts),
  retract(dir(Source, Prot, Dad, Cntnts)),
  assert(dir(Source, Prot, Dad, Newcntnts)),
  retract(dir(Dest, Protect, Mum, Conts)),
  assert(dir(Dest, Protect, Mum, Newconts)).
move($, $, $, $):-
  modefail(yes),
  retract(modefail(yes)).

/*
These definitions have been abstracted from various Prolog includes
header files. They provide the interface with the operating system rom
routines needed by Qdos for timestamping and window/menu/cursor
management. This saves on time, disc space and memory since only a few
of the many rom routines are used from the includes folder.
*/

:- definerom(frontwindow, 16'a924, ptr, []).
:- definerom(setwtitle, 16'a91a, no, [ptr, string]).
:- definerom(readdatetime, 16'a039, integer+d0, [var+longint+a0]).
:- definerom(secs2date, 16'a9c6, no, [d0+longint, var+record+a0]).
:- definerom(sysbeep, 16'a9c8, no, [integer]).
:- definerom(clearmenubar, 16'a934, no, []).
:- definerom(drawmenubar, 16'a937, no, []).
:- definerom(hidecursor, 16'a852, no, []).
:- definerom(eventavail, 16'a971, boolean, [integer, var+record]).
:- definerom(getnextevent, 16'a970, boolean, [integer, var+record]).
:- definerom(bitand, 16'a858, longint, [longint, longint]).

get_field(FIELD, TYPE, STRUCT, RESULT) :-
  record_field(FIELD, TYPE, RETTYPE, OFFSET),
  access_function(RETTYPE, ACCESSFUNC),
  apply(ACCESSFUNC, [STRUCT, OFFSET, RESULT]).
```

```
access_function(word, mempeekword_ptr).
access_function(long, mempeek_ptr).

record_field(year, datetimerec, word, 0).
record_field(month, datetimerec, word, 2).
record_field(day, datetimerec, word, 4).
record_field(hour, datetimerec, word, 6).
record_field(minute, datetimerec, word, 8).
record_field(second, datetimerec, word, 10).
record_field(dayofweek, datetimerec, word, 12).
record_field(message, eventrecord, long, 2).
record_size(eventrecord, 16, ptr).
record_field(hilong, int64bit, long, 0).
record_field(lolong, int64bit, long, 4).

/*
These clauses provide all of the UNIX type output in the normal course
of the interaction. This is mainly directory listing in long and short
formats and the 'pwd' command
*/

writelist([]).
writelist([Head|Tail]):-
 write(Head),
 writelist(Tail).


listconts([]).
listconts([Head|Tail]) :-
  write(Head), nl,
  listconts(Tail).

longlistconts(Argument, Dir):-
 writelist([Dir, ' :']), nl,
 write('MODE'), tabto(10),
 write('OWNER'), tabto(20),
 write('SIZE'), tabto(30),
 write('NAME'), nl,
 write('~~~~'), tabto(10),
 write('~~~~~'), tabto(20),
 write('~~~~'), tabto(30),
 write('~~~~'), nl,
 longlist(Argument, Dir).

longlist([], $).
longlist([Head|Tail], Dir):-
 owner(Owner),
 dir(Head, Mode, Dir, Contents),
 argcount(Contents, Number),
 write('d'),
 transmode(Mode), tabto(10),
 write(Owner), tabto(20),
 write(Number), tabto(30),
 write(Head), nl,
 longlist(Tail, Dir).
longlist([Head|Tail], Dir):-
 file(Head, Dir, Mode),
 owner(Owner),
 transmode(Mode), tabto(10),
 write(Owner), tabto(20),
 write('0'), tabto(30),
 write(Head), nl,
 longlist(Tail, Dir).
```

```
/*
transmode/1 converts the output format of mode codings to 'rwx'
instead of numbers. This serves the ls -l command option.
*/

transmode(000):-
 write('---').
transmode(100):-
 write('--x').
transmode(200):-
 write('-w-').
transmode(300):-
 write('-wx').
transmode(400):-
 write('r--').
transmode(500):-
 write('r-x').
transmode(600):-
 write('rw-').
transmode(700):-
 write('rwx').



/*
?precond/0 contains all the help messages relating to the
preconditions that each failed command entails. In order that the
messages are not repeated more than three times, a count is kept for
each session.
*/

dupprecond :-
 enough(dup),
 nl,
 write('*****************************'), nl,
 write('* All names must be unique. *'), nl,
 write('*****************************'), nl,
 noteit(dup).
dupprecond.

nonemptyprecond :-
 enough(nonempt),
 nl,
 write('*****************************'), nl,
 write('* A directory must be empty *'), nl,
 write('* before it can be deleted. *'), nl,
 write('*****************************'), nl,
 noteit(nonempt).
nonemptyprecond.

refprecond :-
 enough(ref),
 nl,
 write('***********************************************'), nl,
 write('* The item you apply the command to must exist *'), nl,
 write('* in the specified directory or the current  *'), nl,
 write('* directory if you do not specify one.      *'), nl,
 write('***********************************************'), nl,
 noteit(ref).
refprecond.
```

```
readprecond :-
 enough(read),
 nl,
 write('*********************************************'), nl,
 write('* You must have read permission on the item *'), nl,
 write('* for that command to succeed. Use the chmod *'), nl,
 write('* command to gain read permission.      *'), nl,
 write('*********************************************'), nl,
 noteit(read).
readprecond.

writeprecond :-
 enough(write),
 nl,
 write('*********************************************'), nl,
 write('* You must have write permission on the item *'), nl,
 write('* for that command to succeed. Use the chmod *'), nl,
 write('* command to gain write permission.     *'), nl,
 write('*********************************************'), nl,
 noteit(write).
writeprecond.

exeprecond :-
 enough(exe),
 nl,
 write('*********************************************'), nl,
 write('* You must have execute permission on the  *'), nl,
 write('* item for that command to succeed. Use the *'), nl,
 write('* chmod command to gain execute permission. *'), nl,
 write('*********************************************'), nl,
 noteit(exe).
exeprecond.

noteit(Type):-
 iterate(Type, 3).
noteit(Type):-
 retract(iterate(Type, Count)),
 Newcount is Count + 1,
 assert(iterate(Type, Newcount)).

enough(Type):-
 iterate(Type, Count),
 Count < 3.


/*
pwd/1 prints out the path from the root to the cwd in response to the
pwd command.
*/

pwd($):-
 nl,
 cwd(Cwd),
 getparents(Cwd), nl.

getparents(Dir):-
 Dir = root,
 write(Dir).
getparents(Dir):-
 dir(Dir, $, Parent, $),
 getparents(Parent),
 write('/'),
 write(Dir).
```

```
/*
start/0 is the main calling clause of the program. It consults all the
other program files and controls the program flow via a repeat call to
the main calling clause 'prompt'. It also reads in the starting
directory structure for the evaluation exercises.
*/

 :- ['mylib.h'].        %reconsult my rom library

 :- [appearance].       %reconsult disguiser
 :- appearance.         %disguise prolog

 :- [service, input, output, precond,    %reconsult my system
 pwd_Server, cd_Server, ls_Server,
 rm_server, rmdir_server, mkdir_server,
 cp_server, mv_server, utils, chmod_server,
 mode_server, recorder, analyser, tellkids,
 time, lastcd_checker, 'ls-l_server',
 lastrmv_checker, lastread_checker,
 lastwrite_checker, lastexe_checker,
 lastmkmvcp_checker, mkfile_Server].

start :-
 [startup],          %create start situation
 getdatetime(Dow, D, Mo, Y, H, Mi, S),
 retract(starttime(_, _, _, _, _, _, _)),    %retract dummy starttime
 assert(starttime(Dow, D, Mo, Y, H, Mi, S)),    %save start time
 nl(40),
 go.

go :-
 repeat,          %loop cycle
 prompt.

prompt :-
 nl,
 write('ÆDOS: '),
 input(Commandline),
% writelist(Commandline), nl,     %for test harness work
 validcomm(Commandline),
 service(Commandline),
 record(Commandline),       %record events
% analyse,         %switch off help system here
 retract(error(_, _, _)),       %reset error flag
 assert(error(noerror, nul, nul)),
 !,
 fail.

 :- start.



/*
this version of record/1 asserts the log instead of recording it in a
file. It records commandline,  current directory and date/time record in
it (only implicative successful commands if these lines are not
commented out).
*/

record(Commandline):-
 %divide(Commandline, Command, Argument),
```

```
 %isin([cd, mv, rmdir, mkfile, mkdir, cp, rm, chmod], Command),
%implicative &
 error(noerror, $, $),
%successful
 cwd(Cwd),                                              %commands only
 getdatetime(DoW, Day, Month, Year, Hour, Min, Sec),
 asserta(event(Commandline, Cwd, DoW, Day, Month, Year, Hour, Min,
Sec)).
record(Commandline):-                                    %record error
 cwd(Cwd),                                              %details too
 error(Errortype, Dir, Arg),
 getdatetime(DoW, Day, Month, Year, Hour, Min, Sec),
 asserta(nonevent(Commandline, Errortype, Dir, Arg, Cwd, DoW, Day,
Month, Year, Hour, Min, Sec)).


%logerror(Errortype):-
 %getdatetime(DoW, Day, Month, Year, Hour, Min, Sec),
 %assertz(errorcount(Errortype, DoW, Day, Month, Year, Hour, Min,
Sec)).

save_dbase :-
 tell(finishup),
 listing(owner),
 listing(totaltime),
 listing(file),
 listing(dir),
 %listing(errorcount),
 listing(event),
 listing(nonevent),
 told.

/*
rmdir/1 removes directories from the structure after checking the
command for any errors or obstacles.
*/

rmdir([]):-
 write('Usage: rmdir directory ...'),
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
rmdir([' '|List]):-
 cwd(Cwd),
 followdir(List, Cwd).

followdir([$, ' '|$], $):-
 writelist(['Path separator is: ', '"', /, '"', '.']), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
followdir([Dir], Source):-                    %cannot rmdir parent
 dir(Dir, $, Source, []),
 dir(Source, $, $, Contents),
 xperm(Source),
 wperm(Dir),
 delete(Dir, Contents, Newlist),
 retract(dir(Source, Protection, Dad, Contents)),
 assert(dir(Source, Protection, Dad, Newlist)),
 retract(dir(Dir, $, Source, [])).
followdir([File], Source):-
 file(File, Source, $),
 writelist(['"', File, '"', ' is a file']), nl,
 retract(error(_, _, _)),
 assert(error(logical, nul, nul)).
```

```
followdir([Dir], Source):-
 dir(Dir, $, Source, Conts),
 not argcount(Conts, 0),
 writelist(['"', Dir, '"', ' : not empty.']),
 retract(error(_, _, _)),
 assert(error(nonempty, nul, nul)).
followdir([Head, '/'|Rest], Source):-    %trace down
 dir(Head, $, Source, $),
 followdir(Rest, Head).
followdir([Head, '/'|Rest], Source):-    %trace up
 dir(Source, $, Head, $),
 followdir(Rest, Head).
followdir([Dir], Source):-
 modefail(yes),
 retract(modefail(yes)).
followdir([Head|$], Source):-
 writelist(['"', Head, '"', ': not found.']), nl,
 retract(error(_, _, _)),
 assert(error(reference, Source, Head)).


/*
rm/1 removes files from the structure after checking the correctness of
the command.
*/

rm([]):-
 write('Usage: rm <file>.'),
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
rm([' '|List]):-
 cwd(Cwd),
 follow(List, Cwd).

follow([$, ' '|$], $):-
 writelist(['Path separator is: ', '"', /, '"', '.']), nl,
 retract(error(_, _, _)),
 assert(error(syntax, nul, nul)).
follow([Dir], Source):-
 dir(Dir, $, Source, $),
 writelist(['"', Dir, '"', ' is a directory.']), nl,
 retract(error(_, _, _)),
 assert(error(logical, nul, nul)).
follow([File], Source):-     %target must be file -> child
 file(File, Source, $),
 xperm(Source),
 wperm(Source),
 wperm(File),
 dir(Source, $, $, Contents),
 delete(File, Contents, Newlist),
 retract(dir(Source, Protection, Dad, Contents)),
 assert(dir(Source, Protection, Dad, Newlist)),
 retract(file(File, Source, $)).
follow([Head, '/'|List], Cwd):-     %trace down
 dir(Head, $, Cwd, $),
 follow(List, Head).
follow([Head, '/'|List], Cwd):-     %trace up
 dir(Cwd, $, Head, $),
 follow(List, Head).
follow($, $):-
 modefail(yes),
 retract(modefail(yes)).
follow([Head|$], Dir):-
```

```
 writelist(['"', Head, '"', ': not found.']), nl,
 retract(error(_, _, _)),
 assert(error(reference, Dir, Head)).

/*
service/1, validcomm/1 and command/1 are part of the parser. It
checks the command lexically performs the logout and calls the clauses
which record the required parts of the database log. It calls the
individual modules which perform the parsing of arguments for each of
the commands.
*/


validcomm([]):-
 fail.
validcomm([Command|$]):-
 command(Command).

command(logout).
command(chmod).
command(ls).
command(pwd).
command(cp).
command(mv).
command(rm).
command(rmdir).
command(mkdir).
command(mkfile).
command(cd).
command(Badcommand):-
 nl,
 retract(error(_, _, _)),
 assert(error(lexical, nul, nul)),
 record([Badcommand]),
 writelist(['"', Badcommand, '"', ' : not a valid command']), nl, !,
 fail.

service([logout|$]):-
 starttime(Dow, D, M, Y, H, Mi, S),
 timelapse(Dnow, Hnow, Mnow, Snow, D, H, Mi, S, Dlapse, Hlapse, Mlapse,
Slapse),
 retract(totaltime(_, _, _)),
 assert(totaltime(Hlapse, Mlapse, Slapse)),
 save_dbase,
 exit.                    %quit for return to prolog exit for desktop
service([ls|Arguments]):-
 ls(Arguments).
service([pwd|Arguments]):-
 pwd(Arguments).
service([cd|Arguments]):-
 cd(Arguments).
service([chmod|Arguments]):-
 chmod(Arguments).
service([mv|Arguments]):-
 mv(Arguments).   %call service routine for each command
service([rm|Arguments]):-
 rm(Arguments).
service([rmdir|Arguments]):-
 rmdir(Arguments).
service([mkdir|Arguments]):-
 mkdir(Arguments).
service([mkfile|Arguments]):-
 mkfile(Arguments).
```

```
service([cp|Arguments]):-
 cp(Arguments).


/*
Startup
These clauses set up all of the initial conditions required for the user
to interact with the system. All error counts, times and the directory
structure are set to startup values and the events which created the
initial structure are asserted so that the analyser has some evidence
for errors having theses events as causal antecedents.
*/

owner(boss).

modefail(no).    %original setting for permission violation

iterate(ref, 0).      %start values for error message repeats
iterate(dup, 0).     %no more than 3 repeats of each synchronic
iterate(read, 0).     %error message
iterate(write, 0).
iterate(exe, 0).
iterate(nonempt, 0).

error(noerror, nul, nul).   %no errors at startup

cwd(home).     %start in home dir

prewd(nul, monday, 1,1,1900,0,0,0). %dummy previous working dir

starttime(monday, 1,1,1900,0,0,0). %dummy starttime
totaltime(0,0,0).     %dummy elapsed time


/*the finishup file from proto-session contents go in here to provide a
realistic startup situation and a long term interaction history for the
analyser to work on. The makestartup file is seen by Qdos and its
finishup file is copied into here.*/


file(temp, home, 700).
file(fromjim, persin, 0).
file(frommary, persin, 0).
file(tojim, persout, 0).
file(complaintin, busin, 400).
file(enquirout, busout, 700).
file(suppaddr, busaddr, 700).
file(custaddr, busaddr, 700).
file(smiths, persaddr, 700).
file(joneses, persaddr, 700).
file(bloggs, persaddr, 700).
file(johnsons, persaddr, 700).
file(fromeric, mail, 700).
file(addout, busout, 700).


/* dir/4 */

dir(root, 0, nul, [home]).
dir(persin, 700, personal, [frommary, fromjim]).
dir(persout, 700, personal, [tojim]).
dir(personal, 0, mail, [persout, persin]).
dir(busin, 700, business, [complaintin]).
```

```
dir(addresses, 700, home, [persaddr, busaddr]).
dir(busaddr, 700, addresses, [custaddr, suppaddr]).
dir(mailin, 700, home, []).
dir(home, 700, root, [mailin, addresses, mail, temp]).
dir(mail, 700, home, [fromeric, business, personal]).
dir(busout, 700, business, [addout, enquirout]).
dir(business, 100, mail, [busout, busin]).
dir(persaddr, 200, addresses, [johnsons, bloggs, joneses, smiths]).


event([chmod, ' ', 200, ' ', home, (/), addresses, (/), persaddr], mail,
   wednesday, 8, jan, 1992,10,26,27).
event([chmod, ' ', 100, ' ', business], mail, wednesday, 8, jan,
1992,10,
   26,27).
event([mkfile, ' ', business, (/), busout, (/), addout], mail,
wednesday,
   8, jan, 1992,10,26,27).
event([mkfile, ' ', fromeric], mail, wednesday, 8, jan, 1992,10,26,27).
event([cd, ' ', mail], mail, wednesday, 8, jan, 1992,10,26,27).
event([mkdir, ' ', mailin], home, wednesday, 8, jan, 1992,10,26,27).
event([cd], home, wednesday, 8, jan, 1992,10,26,26).
event([mkfile, ' ', johnsons], persaddr, wednesday, 8, jan, 1992,10,26,
   26).
event([mkfile, ' ', bloggs], persaddr, wednesday, 8, jan, 1992,10,26,
   26).
event([mkfile, ' ', joneses], persaddr, wednesday, 8, jan, 1992,10,26,
   26).
event([mkfile, ' ', smiths], persaddr, wednesday, 8, jan, 1992,10,26,
   26).
event([cd, ' ', addresses, (/), persaddr], persaddr, wednesday, 2, may,
   1991,10,26,26).
event([mkfile, ' ', custaddr], busaddr, wednesday, 8, jan, 1992,10,26,
   25).
event([mkfile, ' ', suppaddr], busaddr, wednesday, 8, jan, 1992,10,26,
   25).
event([cd, ' ', busaddr], busaddr, wednesday, 8, jan, 1992,10,26,25).
event([mkdir, ' ', persaddr], addresses, wednesday, 8, jan, 1992,10,26,
   25).
event([mkdir, ' ', busaddr], addresses, wednesday, 8, jan, 1992,10,26,
   25).
event([cd, ' ', business, (/), mail, (/), home, (/), addresses],
addresses,
   wednesday, 8, jan, 1992,10,26,25).
event([mkfile, ' ', enquirout], busout, wednesday, 8, jan, 1992,10,26,
   25).
event([cd, ' ', business, (/), busout], busout, wednesday, 8, jan, 1992,
   10,26,24).
event([chmod, ' ', 400, ' ', complaintin], busin, wednesday, 2, may,
   1991,10,26,24).
event([mkfile, ' ', complaintin], busin, wednesday, 8, jan, 1992,10,26,
   24).
event([cd, ' ', busin], busin, wednesday, 8, jan, 1992,10,26,24).
event([mkdir, ' ', busout], business, wednesday, 8, jan, 1992,10,26,24).
event([mkdir, ' ', busin], business, wednesday, 8, jan, 1992,10,26,24).
event([cd, ' ', business], business, wednesday, 8, jan, 1992,10,26,23).
event([chmod, ' ', 0, ' ', personal], mail, wednesday, 8, jan, 1992,10,
   26,23).
event([cd, ' ', personal, (/), mail], mail, wednesday, 8, jan, 1992,10,
   26,23).
event([chmod, ' ', 0, ' ', tojim], persout, wednesday, 8, jan, 1992,10,
   26,23).
event([mkfile, ' ', tojim], persout, wednesday, 8, jan, 1992,10,26,23).
```

```
event([cd, ' ', personal, (/), persout], persout, wednesday, 2, may,
   1991,10,26,23).
event([chmod, ' ', 0, ' ', frommary], persin, wednesday, 8, jan,
1992,10,
   26,23).
event([mkfile, ' ', frommary], persin, wednesday, 8, jan, 1992,10,26,
   22).
event([chmod, ' ', 0, ' ', fromjim], persin, wednesday, 8, jan, 1992,10,
   26,22).
event([mkfile, ' ', fromjim], persin, wednesday, 8, jan, 1992,10,26,22).
event([cd, ' ', persin], persin, wednesday, 8, jan, 1992,10,26,22).
event([mkdir, ' ', persout], personal, wednesday, 8, jan, 1992,10,26,
   22).
event([mkdir, ' ', persin], personal, wednesday, 8, jan, 1992,10,26,21).
event([cd, ' ', personal], personal, wednesday, 8, jan, 1992,10,26,21).
event([mkdir, ' ', business], mail, wednesday, 8, jan, 1992,10,26,21).
event([mkdir, ' ', personal], mail, wednesday, 8, jan, 1992,10,26,21).
event([cd, ' ', mail], mail, wednesday, 8, jan, 1992,10,26,21).
event([mkdir, ' ', addresses], home, wednesday, 8, jan, 1992,10,26,21).
event([mkdir, ' ', mail], home, wednesday, 8, jan, 1992,10,26,21).


/*
tellkids/2 informs all of the child records of a directory record if
the directory is renamed as each object record has within it a field for
its parent directory.
*/

tellkids([], $).
tellkids([Head|Tail], Name):-
 retract(dir(Head, Mode, Parent, Conts)),
 assert(dir(Head, Mode, Name, Conts)),
 tellkids(Tail, Name).
 tellkids([Head|Tail], Name):-
 retract(file(Head, Parent, Mode)),
 assert(file(Head, Name, Mode)),
 tellkids(Tail, Name).

/*
getdatetime/7 gets the system time and converts it to twelve hour
clock. The day of the week is calculated and a timelapse can be
calculated also. This is for timestamping user action events as they
occur.
* /


getdatetime(Dayofweek, Day, Month, Year, Hour, Minute, Second) :-
 readdatetime(Secs, Errorcode),
 secs2date(Secs, 16'FFFE),
 get_field(year, datetimerec, 16'FFFE, Year),
 get_field(month, datetimerec, 16'FFFE, Monthnumber),
 get_field(day, datetimerec, 16'FFFE, Day),
 get_field(hour, datetimerec, 16'FFFE, Hour),
 get_field(minute, datetimerec, 16'FFFE, Minute),
 get_field(second, datetimerec, 16'FFFE, Second),
 get_field(dayofweek, datetimerec, 16'FFFE, Dayofweeknumber),
 %twelvehourclock(Hour, Hour12),
 numbertoday(Dayofweek, Dayofweeknumber),
 numbertomonth(Month, Monthnumber).

twelvehourclock(H24, H12):-
 H24 > 12,
```

```
 H12 is H24 - 12.
twelvehourclock(H24, H24).

numbertoday(sunday, 1).
numbertoday(monday, 2).
numbertoday(tuesday, 3).
numbertoday(wednesday, 4).
numbertoday(thursday, 5).
numbertoday(friday, 6).
numbertoday(saturday, 7).

numbertomonth(january, 1).
numbertomonth(february, 2).
numbertomonth(march, 3).
numbertomonth(april, 4).
numbertomonth(may, 5).
numbertomonth(june, 6).
numbertomonth(july, 7).
numbertomonth(august, 8).
numbertomonth(september, 9).
numbertomonth(october, 10).
numbertomonth(november, 11).
numbertomonth(december, 12).

/*
timelapse works out the difference between a past time event and the
current time. If the two times straddle a month boundary, then the lapse
in days will be incorrect but I couldn't be bothered  with different
length months!
*/

timelapse(Dnow, Hnow, Mnow, Snow, Dthen, Hthen, Mthen, Sthen, Dlapse,
Hlapse, Mlapse, Slapse):-
 getdatetime($, Dnow, $, $, Hnow, Mnow, Snow),
 secslapse(Snow, Sthen, Slapse, Scarry),
 minslapse(Mnow, Mthen, Mlapse, Scarry, Mcarry),
 hourslapse(Hnow, Hthen, Hlapse, Mcarry, Hcarry),
 dayslapse(Dnow, Dthen, Dlapse, Hcarry).

secslapse(Snow, Sthen, Slapse, -1):-
 Sthen > Snow,
 Slapse is Snow - Sthen + 60.
secslapse(Snow, Sthen, Slapse, 0):-
 Slapse is Snow - Sthen.

minslapse(Mnow, Mthen, Mlapse, Scarry, -1):-
 Mthen > Mnow,
 Mlapse is Mnow - Mthen + 60 + Scarry.
minslapse(Mnow, Mthen, Mlapse, Scarry, 0):-
 Mlapse is Mnow - Mthen + Scarry.

hourslapse(Hnow, Hthen, Hlapse, Mcarry, -1):-
 Hthen > Hnow,
 Hlapse is Hnow - Hthen + 12 + Mcarry.
hourslapse(Hnow, Hthen, Hlapse, Mcarry, 0):-
 Hlapse is Hnow - Hthen + Mcarry.

dayslapse(Dnow, Dthen, Dlapse, Hcarry):-
 Dthen > Dnow,
 Dlapse is Dnow - Dthen + 24 + Hcarry.
dayslapse(Dnow, Dthen, Dlapse, Hcarry):-
 Dlapse is Dnow - Dthen + Hcarry.
```

```
/*
A loose collection of sundry utilities for finding and deleting list
items etc.
*/


delete(Item, [Item|Tail], Tail).
delete(Item, [Y|Tail], [Y|Tail1]):-
delete(Item, Tail, Tail1).

add(Item, List, [Item|List]).

argcount([], 0).
argcount([_|Rest], Number):-
  argcount(Rest, Temp),
  Number is 1 + Temp.

isin([Target|_], Target):-!.
isin([_|Tail], Target):-
   isin(Tail, Target).

isancestor(Target, Dest):-
  dir(Dest, _, Target, _).
isancestor(Target, Dest):-
  dir(Dest, _, Parent, _),
  isancestor(Target, Parent).

divide([Command|Argument], Command, Argument).
```

Appendix C
Task Specification

Appendix C contains the task specification as presented to subjects in all evaluative exercises. Subjects were advised to create maps for the startup and target structures to make the task more manageable.

The Boss's Memo

Please amend my computerised filing system QDOS as at the moment it is in a bit of a mess. If the files I mention here already exist, please preserve them and if they don't, create them. Otherwise, do it any way you like as long as it ends up according to the following description: The home directory should contain two subdirectories called 'business' and 'personal', the latter should have no privileges granted. The business directory should have in it two subdirectories called 'busmail' and 'busaddr'. busmail should be read-protected. busaddr should contain the files 'custaddr', 'suppaddr'. busmail should contain subdirectories called 'busmailin' and 'busmailout'. busmailin should contain files called 'complaintin' and 'thanksin' and busmailout should contain files called 'inquireout' and 'compreply'. The personal directory should have two subdirectories called 'persmail' and 'persaddr' and persmail should have no privileges granted. persmail should contain files called 'tojim', 'fromjim', 'frommary' and 'tomary'. All the files in persmail should have no privileges granted. persaddr should contain files called 'smiths', 'joneses' and 'bloggs'.

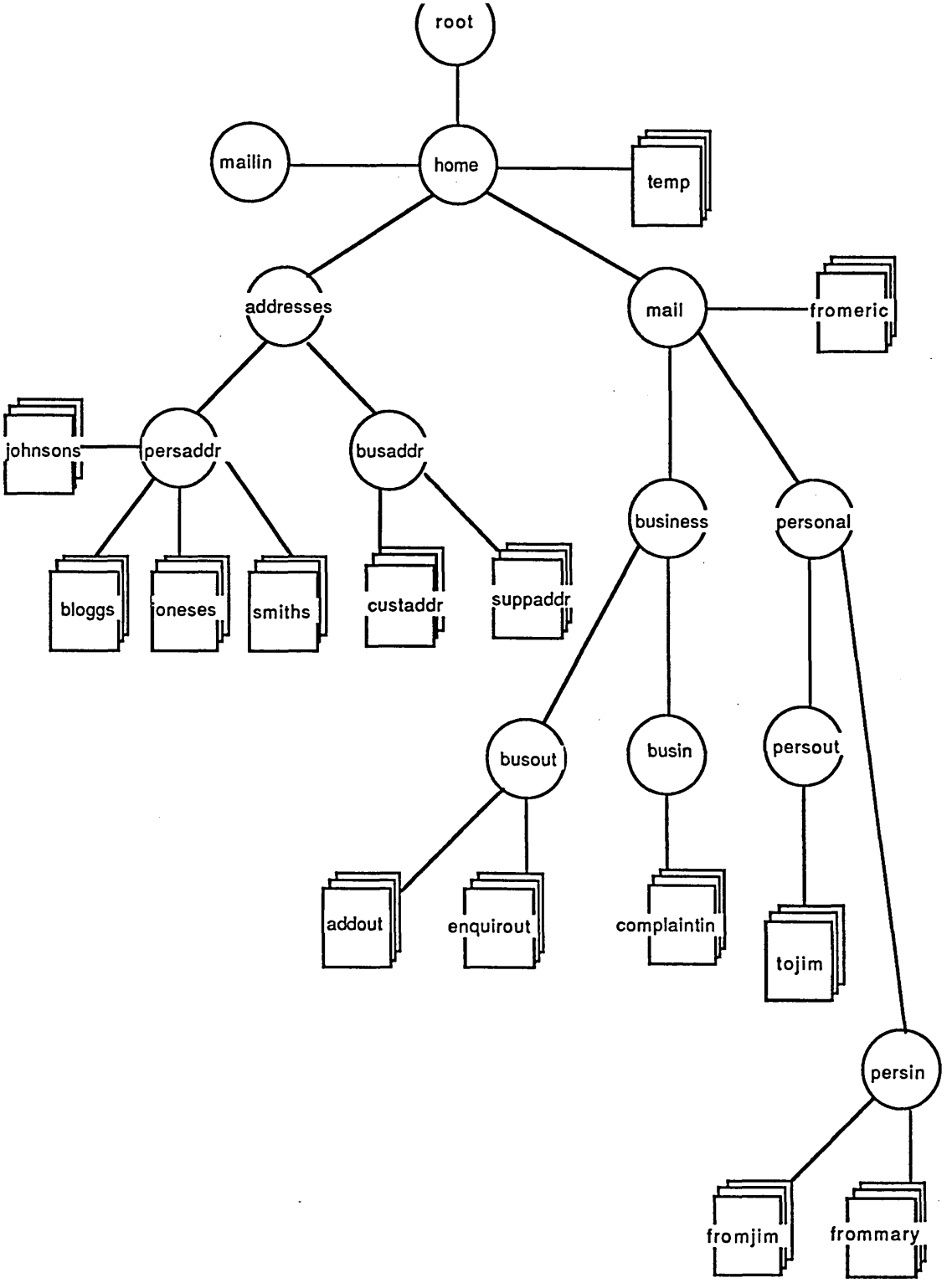
P.S. On second thoughts, get rid of the bloggs file and make sure to get rid of anything I haven't mentioned here. Oh, and add a file called 'accountsaddr' to the busaddr directory and delete the file 'tomary'.

Thanks,

The Boss.

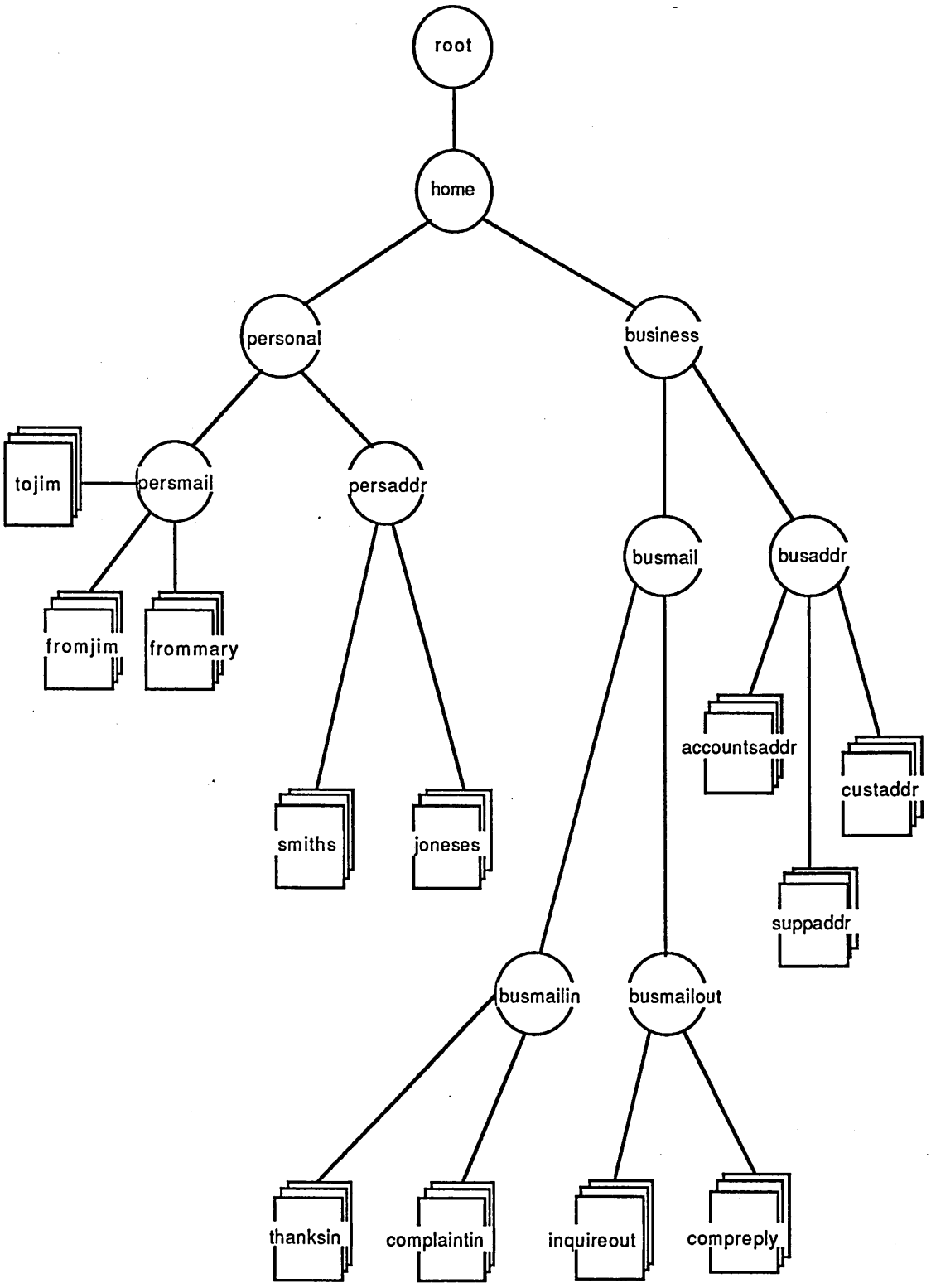
Appendix D
Directory Startup Structure

Appendix D shows pictorially the initial directory structure at the start of the task for all evaluative exercises.


root
mailin
home
temp
addresses
mail
fromeric
johnsons
persaddr
busaddr
business
personal
bloggs
joneses
smiths
custaddr
suppaddr
busout
busin
persout
addout
enquirout
complaintin
tojim
persin
fromjim
frommary

Appendix E
Directory Target Structure

Appendix E shows pictorially the target directory structure as described in the task specification. Protection modes are not shown.


root
home
personal
business
tojim
persmail
persaddr
busmail
busaddr
fromjim
frommary
smiths
joneses
accountsaddr
custaddr
suppaddr
busmailin
busmailout
thanksin
complaintin
inquireout
compreply

Appendix F
Tutorial Material

Appendix F contains material used in explanation of the system to all subjects involved in evaluative exercises. This amounts to a 'script' which was followed in instructing subjects in the various attributes of the system. The intention was to ensure that all subjects received the same information in the same form, thus eliminating any bias that might occur through variations in tutorial procedure.

Introduction

• Before I explain what is going to happen, a few comments.

• Firstly, I very much appreciate your agreeing to take part.

• Secondly, although I would like you to do your best in the task you're given, it is the system that is under test, not you. I am only interested in how easy you found the system to use, not in how well you did.

• And lastly don't be afraid to try things out. You can't break anything!

The System

• The system is a file system called QDOS which is similar in many ways to MS DOS and others which you may have come across.

What's There?

• Files. The system is used to store files in an organised way. Files can be created, destroyed, moved, renamed, copied and protected. You do not have to concern yourself with what is inside a file.

• All files are kept in directories.

• The directories are organised hierarchically, a bit like a family tree.

• Each directory may contain other directories or files.

• All directories are themselves items in a parent directory except for the root directory which has no parent.

• Files cannot be parents.

• Directories can be created, destroyed, listed and protected.

Current Working Directory

• At any time there is a current working directory. This is the directory that you are, so to speak, IN and the one to which all of your commands apply unless you specify otherwise.

• For example, the command 'rm letter1' assumes that the file letter1 is in the current directory.

• You can make some other directory the current directory using the 'cd' command.

Paths

• If you want a command to apply to a directory other than the current directory you must specify this by providing a **path** to the target directory from the current directory.

• You do this by giving a list of the directories 'on the way to' and including the target, separating each from the next by a stroke '/'.

• EG: rm Reps/Sales/Rep1

• Details of commands and their use are in the handout.

Getting Started

• To start, all you have to do is put your disc into the disc drive and switch the computer on.

• When a little window opens up in the top left hand corner of the screen, move the arrow over the QDOS icon, using the mouse, and click the mouse button twice in quick succession.

• After a short time, the QDOS prompt will appear and you can start typing in QDOS commands.

The Task

• Your boss has a QDOS system on which she has created a set of directories and files. You are to rearrange this as detailed in the handout.

• Please don't logout until you're sure that you have finished the task or I ask you to stop.

Appendix G
Command List

Appendix G contains a copy of the system documentation given as reference to each subect in all evaluative exercises. Subjects were allowed to refer to this during task performance.

Commands

pwd.....	Print current Working Directory
cd<path>.....	Change current working Directory
ls [-l][<path>].....	LiSt directory contents
mkfile <path>.....	MaKe a FILE
mkdir <path>.....	MaKe a DIRectory
rm <path>.....	ReMove a file
rmdir <path>.....	ReMove a DIRectory
chmod <mode> <path>.....	CHange protection MODe on file or dir.
cp <path1> <path2>.....	create CoPy of file1 as newname 2
mv <path1> <path2>.....	MoVe file 1 to newname or directory 2
logout.....	leave QDOS

• Anything in square brackets above is optional.

• A path can trace the directory structure upwards, downwards or both.

• The -l option for ls produces a directory listing which gives full details about each directory item.

• ls on its own lists the contents of the current directory.

• cd on its own makes the home directory the current directory.

• mv can be used to rename a file without moving it.

• Permission modes are set by numbers in multiples of one hundred from 000 to 700.

000 = no permissions granted
100 = execute permission granted
200 = write permission granted
400 = read permission granted

These are added together to give the various combinations. For example, to grant only read and execute permission the mode would have to be 100 + 400 = 500:

ie: chmod 500 <path>

This would appear in an 'ls -l' listing as "r-x".

Permission modes determine what you are allowed to do with an item. For example, you cannot list a directory unless it has execute and read permissions granted. You cannot delete a file without having write permission on it etc.

• Directories cannot be removed if they are not empty.

• All file and directory names must be unique so a file copy must be given a new name in addition to any destination directory to which it is to be copied.

• Directories cannot be moved or copied.

Appendix H
Directory Structure Illustration

The diagram in appendix H was used to illustrate the hierarchical nature of the UNIX directory structure to subjects involved in evaluative exercises.

Directory/File Structure

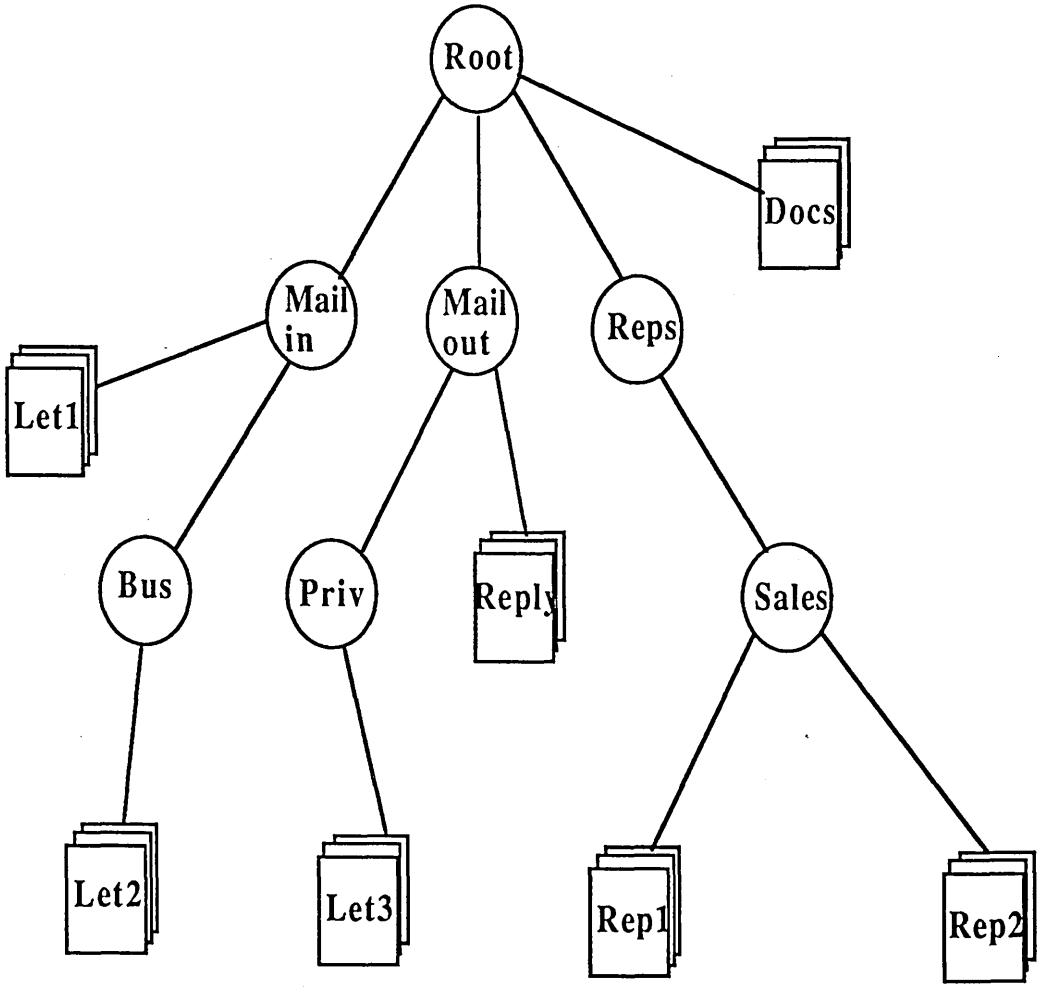


Diagram to illustrate to subjects the hierarchical directory structure.

Appendix I
Experimental Log Files

Appendix I contains the log files for all subjects involved in the experimental evaluation sessions. These files are generated by the system to provide an interaction history which can be used by the help system in reasoning about user errors and to provide the data for statistical analysis of performance for test and control conditions. Clauses showing the closing state of the system were used to evaluate the percentage of task completed. The total time taken is also recorded and taken into account. Records of error counts form the basis of evaluations of error performance. The event list of successful commands is used by the help system in determining the causes of errors.

```
/* owner/1 */

owner(s1).


/* file/3 */

file(custaddr, busaddr, 700).
file(suppaddr, busaddr, 700).
file(complaintin, busmailin, 700).
file(thanksin, busmailin, 700).
file(enquirout, busmailout, 700).
file(compreply, busmailout, 700).
file(tojim, persmail, 700).
file(fromjim, persmail, 700).
file(frommary, persmail, 700).
file(smiths, persaddr, 700).
file(joneses, persaddr, 700).
file(accountsaddr, busaddr, 700).


/* dir/4 */

dir(root, 0, nul, [home]).
dir(business, 700, home, [busaddr, busmail]).
dir(busmailin, 700, busmail, [thanksin, complaintin]).
dir(busmailout, 700, busmail, [compreply, enquirout]).
dir(persmail, 700, personal, [frommary, fromjim, tojim]).
dir(busaddr, 700, business, [accountsaddr, suppaddr, custaddr]).
dir(busmail, 200, business, [busmailout, busmailin]).
dir(persaddr, 0, personal, [joneses, smiths]).
dir(personal, 0, home, [persaddr, persmail]).
dir(home, 700, root, [personal, business]).


/* errorcount/2 */

errorcount(range, 0).
errorcount(uniqueness, 0).
errorcount(read, 0).
errorcount(write, 0).
errorcount(execute, 0).
errorcount(logical, 7).
errorcount(nonempty, 2).
errorcount(syntax, 6).
errorcount(duplication, 3).
errorcount(lexical, 3).
errorcount(reference, 26).


/* totaltime/3 */

totaltime(1, 21, 11).


/* event/9 */

event([rmdir, ' ', helen], home, thursday, 9, may, 1991, 12, 47, 16).
event([cd], home, thursday, 9, may, 1991, 12, 47, 7).
event([rm, ' ', johnsons], helen, thursday, 9, may, 1991, 12, 47, 3).
event([rm, ' ', bloggs], helen, thursday, 9, may, 1991, 12, 46, 56).
event([rm, ' ', fromeric], helen, thursday, 9, may, 1991, 12, 46, 49).
event([rm, ' ', addout], helen, thursday, 9, may, 1991, 12, 46, 41).
event([cd, ' ', helen], helen, thursday, 9, may, 1991, 12, 46, 12).
```

```
event([chmod, ' ', 0, ' ', personal], home, thursday, 9, may, 1991, 12,
   45, 56).
event([chmod, ' ', 0, ' ', personal, (/), persaddr], home, thursday, 9,
   may, 1991, 12, 45, 43).
event([chmod, ' ', 200, ' ', business, (/), busmail], home, thursday, 9,
   may, 1991, 12, 44, 59).
event([cd], home, thursday, 9, may, 1991, 12, 44, 12).
event([mkfile, ' ', accountsaddr], busaddr, thursday, 9, may, 1991, 12,
   43, 59).
event([cd, ' ', business, (/), busaddr], busaddr, thursday, 9, may,
   1991, 12, 43, 37).
event([mv, ' ', helen, (/), joneses, ' ', personal, (/), persaddr],
   home, thursday, 9, may, 1991, 12, 43, 2).
event([mv, ' ', helen, (/), smiths, ' ', personal, (/), persaddr], home,
   thursday, 9, may, 1991, 12, 42, 46).
event([cd], home, thursday, 9, may, 1991, 12, 42, 31).
event([rm, ' ', tomary], persmail, thursday, 9, may, 1991, 12, 41, 23).
event([mkfile, ' ', tomary], persmail, thursday, 9, may, 1991, 12, 41,
   3).
event([cd, ' ', personal, (/), persmail], persmail, thursday, 9, may,
   1991, 12, 40, 51).
event([mv, ' ', helen, (/), frommary, ' ', personal, (/), persmail],
   home, thursday, 9, may, 1991, 12, 39, 53).
event([cd], home, thursday, 9, may, 1991, 12, 38, 21).
event([cd, ' ', personal, (/), persmail], persmail, thursday, 9, may,
   1991,12, 37, 29).
event([mv, ' ', helen, (/), fromjim, ' ', personal, (/), persmail],
   home,thursday, 9, may, 1991, 12, 36, 46).
event([mv, ' ', helen, (/), tojim, ' ', personal, (/), persmail], home,
   thursday, 9, may, 1991, 12, 36, 16).
event([cd], home, thursday, 9, may, 1991, 12, 35, 26).
event([mkfile, ' ', compreply], busmailout, thursday, 9, may, 1991, 12,
   35, 23).
event([cd, ' ', business, (/), busmail, (/), busmailout], busmailout,
      thursday, 9, may, 1991, 12, 35, 4).
event([mv, ' ', helen, (/), enquirout, ' ', business, (/), busmail, (/),
   busmailout], home, thursday, 9, may, 1991, 12, 34, 33).
event([cd], home, thursday, 9, may, 1991, 12, 32, 48).
event([mkfile, ' ', thanksin], busmailin, thursday, 9, may, 1991, 12,
   32, 45).
event([cd, ' ', business, (/), busmail, (/), busmailin], busmailin,
      thursday, 9, may, 1991, 12, 32, 33).
event([mv, ' ', helen, (/), complaintin, ' ', business, (/), busmail,
   (/), busmailin], home, thursday, 9, may, 1991, 12, 30, 39).
event([mv, ' ', helen, (/), suppaddr, ' ', business, (/), busaddr],
   home,thursday, 9, may, 1991, 12, 29, 50).
event([mv, ' ', helen, (/), custaddr, ' ', business, (/), busaddr],
   home,
      thursday, 9, may, 1991, 12, 29, 17).
event([mkdir, ' ', personal, (/), persaddr], home, thursday, 9, may,
   1991,
      12, 27, 45).
event([mkdir, ' ', personal, (/), persmail], home, thursday, 9, may,
   1991,
      12, 27, 24).
event([mkdir, ' ', business, (/), busaddr], home, thursday, 9, may,
   1991, 12,  27, 2).
event([rmdir, ' ', busaddr], home, thursday, 9, may, 1991, 12, 26, 50).
event([mkdir, ' ', busaddr], home, thursday, 9, may, 1991, 12, 26, 34).
event([mkdir, ' ', business, (/), busmail, (/), busmailout], home,
      thursday, 9, may, 1991, 12, 25, 59).
event([mkdir, ' ', business, (/), busmail, (/), busmailin], home,
   thursday, 9, may, 1991, 12, 25, 8).
event([mkdir, '  24, 40).
event([mkdir, ' ', personal], home, thursday, 9, may, 1991, 12, 24, 19).
```

```
event([rmdir, ' ', mail], home, thursday, 9, may, 1991, 12, 23, 37).
event([rmdir, ' ', mail, (/), personal], home, thursday, 9, may, 1991,
   12, 23, 25).
event([rmdir, ' ', mail, (/), business], home, thursday, 9, may, 1991,
   12, 23,  16).
event([rmdir, ' ', mail, (/), personal, (/), persin], home, thursday, 9,
   may, 1991, 12, 22, 20).
event([rmdir, ' ', mail, (/), personal, (/), persout], home, thursday,
   9,  may, 1991, 12, 21, 46).
event([rmdir, ' ', mail, (/), business, (/), busin], home, thursday, 9,
   may,
      1991, 12, 21, 23).
event([rmdir, ' ', mail, (/), business, (/), busout], home, thursday, 9,
   may,  1991, 12, 21, 10).
event([rmdir, ' ', addresses], home, thursday, 9, may, 1991, 12, 20,
   47).
event([rmdir, ' ', addresse], home, thursday, 9, may, 1991, 12, 20, 36).
event([rmdir, ' ', addresses, (/), busaddr], home, thursday, 9, may,
   1991, 12, 20, 24).
event([rmdir, ' ', addresses, (/), persaddr], home, thursday, 9, may,
   1991,  12, 20, 9).
event([rm, ' ', temp], home, thursday, 9, may, 1991, 12, 18, 47).
event([rmdir, ' ', mailin], home, thursday, 9, may, 1991, 12, 18, 37).
event([mv, ' ', mail, (/), personal, (/), persin, (/), fromjim, ' ',
   helen],  home, thursday, 9, may, 1991, 12, 16, 33).
event([mv, ' ', mail, (/), personal, (/), persin, (/), frommary, ' ',
   helen], home, thursday, 9, may, 1991, 12, 16, 6).
event([mv, ' ', mail, (/), personal, (/), persout, (/), tojim, ' ',
   helen],  home, thursday, 9, may, 1991, 12, 15, 37).
event([mv, ' ', mail, (/), business, (/), busin, (/), complaintin, ' ',
   helen], home, thursday, 9, may, 1991, 12, 15, 16).
event([mv, ' ', mail, (/), business, (/), busout, (/), enquirout, ' ',
   helen],  home, thursday, 9, may, 1991, 12, 14, 43).
event([mv, ' ', mail, (/), business, (/), busout, (/), addout, ' ',
   helen],  home, thursday, 9, may, 1991, 12, 14, 6).
event([mv, ' ', mail, (/), fromeric, ' ', helen], home, thursday, 9,
   may,  1991, 12, 13, 35).
event([mv, ' ', addresses, (/), busaddr, (/), suppaddr, ' ', helen],
   home,  thursday, 9, may, 1991, 12, 13, 18).
event([mv, ' ', addresses, (/), busaddr, (/), custaddr, ' ', helen],
   home,  thursday, 9, may, 1991, 12, 12, 44).
event([mv, ' ', addresses, (/), persaddr, (/), bloggs, ' ', helen],
   home,  thursday, 9, may, 1991, 12, 11, 58).
event([mv, ' ', addresses, (/), persaddr, (/), smiths, ' ', helen],
   home,  thursday, 9, may, 1991, 12, 11, 35).
event([mv, ' ', addresses, (/), persaddr, (/), johnsons, ' ', helen],
   home,  thursday, 9, may, 1991, 12, 11, 18).
event([mv, ' ', addresses, (/), persaddr, (/), joneses, ' ', helen],
   home,  thursday, 9, may, 1991, 12, 10, 50).
event([mkdir, ' ', helen], home, thursday, 9, may, 1991, 12, 5, 0).
event([chmod, ' ', 700, ' ', mail, (/), personal], home, thursday, 9,
   may,  1991, 11, 52, 40).
event([chmod, ' ', 700, ' ', mail, (/), business], home, thursday, 9,
   may, 1991, 11, 50, 3).
event([chmod, ' ', 700, ' ', addresses, (/), persaddr], home, thursday,
   9, may, 1991, 11, 47, 46).
event([cd], home, thursday, 9, may, 1991, 11, 37, 4).
event([cd, ' ', mailin], mailin, thursday, 9, may, 1991, 11, 34, 20).


/* owner/1 */

owner(s2).
```

```
/* file/3 */

file(complaintin, busmailin, 700).
file(thanksin, busmailin, 700).
file(custaddr, busaddr, 700).
file(suppaddr, busaddr, 700).
file(accountsaddr, busaddr, 700).
file(enquirout, busmailout, 700).
file(compreply, busmailout, 700).
file(tojim, persmail, 700).
file(fromjim, persmail, 700).
file(smiths, persaddr, 700).
file(joneses, persaddr, 700).
file(frommary, persmail, 700).


/* dir/4 */

dir(root, 0, nul, [home]).
dir(business, 700, home, [busaddr, busmsil]).
dir(busmsil, 700, business, [busmailout, busmailin]).
dir(busmailin, 700, busmsil, [thanksin, complaintin]).
dir(busaddr, 700, business, [accountsaddr, suppaddr, custaddr]).
dir(busmailout, 700, busmsil, [compreply, enquirout]).
dir(personal, 700, home, [persaddr, persmail]).
dir(persaddr, 700, personal, [joneses, smiths]).
dir(persmail, 700, personal, [frommary, fromjim, tojim]).
dir(home, 700, root, [personal, business]).


/* errorcount/2 */

errorcount(range, 0).
errorcount(uniqueness, 0).
errorcount(read, 0).
errorcount(write, 0).
errorcount(execute, 7).
errorcount(syntax, 15).
errorcount(nonempty, 2).
errorcount(logical, 10).
errorcount(lexical, 6).
errorcount(duplication, 5).
errorcount(reference, 33).


/* totaltime/3 */

totaltime(2, 24, 15).


/* event/9 */

event([cd, ' ', personal], personal, thursday, 9, may, 1991, 13, 49,
   42).
event([rm, ' ', addout], home, thursday, 9, may, 1991, 13, 49, 27).
event([rmdir, ' ', addresses], home, thursday, 9, may, 1991, 13, 49,
   16).
event([mv, ' ', frommary, ' ', personal, (/), persmail], home, thursday,
   9,  may, 1991, 13, 48, 43).
event([mv, ' ', joneses, ' ', personal, (/), persaddr], home, thursday,
   9, may, 1991, 13, 48, 5).
event([mv, ' ', smiths, ' ', personal, (/), persaddr], home, thursday,
   9,  may, 1991, 13, 47, 49).
event([mv, ' ', fromjim, ' ', personal, (/), persmail], home, thursday,
```

```
event([mv, ' ', tojim, ' ', personal, (/), persmail], home, thursday, 9,
   may, 1991, 13, 46, 52).
event([cd], home, thursday, 9, may, 1991, 13, 46, 17).
event([mkdir, ' ', persaddr], personal, thursday, 9, may, 1991, 13, 46,
   11).
event([mkdir, ' ', persmail], personal, thursday, 9, may, 1991, 13, 46,
   0).
event([cd, ' ', personal], personal, thursday, 9, may, 1991, 13, 45,
   42).
event([mkdir, ' ', personal], home, thursday, 9, may, 1991, 13, 45, 31).
event([cd], home, thursday, 9, may, 1991, 13, 45, 12).
event([mkfile, ' ', compreply], busmailout, thursday, 9, may, 1991, 13,
   45, 5).
event([cd, ' ', business, (/), busmsil, (/), busmailout], busmailout,
      thursday, 9, may, 1991, 13, 44, 43).
event([mv, ' ', enquirout, ' ', business, (/), busmsil, (/),
   busmailout], home, thursday, 9, may, 1991, 13, 43, 41).
event([cd], home, thursday, 9, may, 1991, 13, 42, 53).
event([cd, ' ', business, (/), busmsil], busmsil, thursday, 9, may,
   1991, 13, 42, 23).
event([mkfile, ' ', accountsaddr], busaddr, thursday, 9, may, 1991, 13,
   41, 15).
event([cd, ' ', business, (/), busaddr], busaddr, thursday, 9, may,
   1991, 13, 40, 50).
event([mv, ' ', suppaddr, ' ', business, (/), busaddr], home, thursday,
   9, may, 1991, 13, 40, 35).
event([mv, ' ', custaddr, ' ', business, (/), busaddr], home, thursday,
   9, may, 1991, 13, 40, 11).
event([cd], home, thursday, 9, may, 1991, 13, 38, 55).
event([cd, ' ', busaddr], busaddr, thursday, 9, may, 1991, 13, 38, 49).
event([cd, ' ', business], business, thursday, 9, may, 1991, 13, 37,
   48).
event([cd], home, thursday, 9, may, 1991, 13, 37, 27).
event([mv, ' ', thanksin, ' ', business, (/), busmsil, (/), busmailin],
   home, thursday, 9, may, 1991, 13, 37, 21).
event([mv, ' ', complaintin, ' ', business, (/), busmsil, (/),
   busmailin], home, thursday, 9, may, 1991, 13, 36, 56).
event([cd], home, thursday, 9, may, 1991, 13, 36, 6).
event([mkdir, ' ', busmailout], busmsil, thursday, 9, may, 1991, 13, 35,
   42).
event([mkdir, ' ', busmailin], busmsil, thursday, 9, may, 1991, 13, 35,
   14).
event([cd, ' ', busmsil], busmsil, thursday, 9, may, 1991, 13, 34, 25).
event([mkdir, ' ', busaddr], business, thursday, 9, may, 1991, 13, 34,
   1).
event([mkdir, ' ', busmsil], business, thursday, 9, may, 1991, 13, 33,
   41).
event([cd, ' ', business], business, thursday, 9, may, 1991, 13, 33,
   16).
event([mkdir, ' ', business], home, thursday, 9, may, 1991, 13, 33, 0).
event([rmdir, ' ', letters], home, thursday, 9, may, 1991, 13, 32, 28).
event([cd], home, thursday, 9, may, 1991, 13, 32, 19).
event([mv, ' ', smiths, ' ', home], letters, thursday, 9, may, 1991, 13,
   32, 1).
event([mv, ' ', joneses, ' ', home], letters, thursday, 9, may, 1991,
   13, 31, 53).
event([cd, ' ', letters], letters, thursday, 9, may, 1991, 13, 31, 46).
event([mv, ' ', joneses, ' ', letters], home, thursday, 9, may, 1991,
   13, 31, 10).
event([cd], home, thursday, 9, may, 1991, 13, 30, 33).
event([cd, ' ', letters], letters, thursday, 9, may, 1991, 13, 30, 3).
event([cd], home, thursday, 9, may, 1991, 13, 29, 39).
event([rmdir, ' ', busaddr], addresses, thursday, 9, may, 1991, 13, 29,
   34).
```

19).
event([mv, ' ', suppaddr, ' ', addresses, (/), home], busaddr, thursday, 9, may, 1991, 13, 29, 4).
event([mv, ' ', custaddr, ' ', addresses, (/), home], busaddr, thursday, 9, may, 1991, 13, 28, 45).
event([cd, ' ', busaddr], busaddr, thursday, 9, may, 1991, 13, 28, 0).
event([rmdir, ' ', persaddr], addresses, thursday, 9, may, 1991, 13, 27, 52).
event([cd, ' ', addresses], addresses, thursday, 9, may, 1991, 13, 27, 29).
event([mv, ' ', joneses, ' ', addresses, (/), home], persaddr, thursday, 9, may, 1991, 13, 27, 14).
event([cd, ' ', persaddr], persaddr, thursday, 9, may, 1991, 13, 26, 49).
event([cd, ' ', addresses], addresses, thursday, 9, may, 1991, 13, 26, 35).
event([rmdir, ' ', mail], home, thursday, 9, may, 1991, 13, 25, 35).
event([cd], home, thursday, 9, may, 1991, 13, 25, 17).
event([rmdir, ' ', business], mail, thursday, 9, may, 1991, 13, 25, 11).
event([cd, ' ', mail], mail, thursday, 9, may, 1991, 13, 24, 59).
event([rmdir, ' ', busout], business, thursday, 9, may, 1991, 13, 24, 50).
event([cd, ' ', mail, (/), business], business, thursday, 9, may, 1991, 13, 24, 37).
event([cd], home, thursday, 9, may, 1991, 13, 24, 13).
event([cd, ' ', busout], busout, thursday, 9, may, 1991, 13, 23, 13).
event([cd, ' ', business], business, thursday, 9, may, 1991, 13, 23, 0).
event([mv, ' ', enquirout, ' ', business, (/), mail, (/), home], busout, thursday, 9, may, 1991, 13, 22, 43).
event([mv, ' ', addout, ' ', business, (/), mail, (/), home], busout, thursday, 9, may, 1991, 13, 22, 23).
event([cd, ' ', busout], busout, thursday, 9, may, 1991, 13, 21, 43).
event([rmdir, ' ', busin], business, thursday, 9, may, 1991, 13, 21, 35).
event([cd, ' ', business], business, thursday, 9, may, 1991, 13, 21, 24).
event([mv, ' ', thanksin, ' ', business, (/), mail, (/), home], busin, thursday, 9, may, 1991, 13, 21, 13).
event([mv, ' ', complaintin, ' ', business, (/), mail, (/), home], busin, thursday, 9, may, 1991, 13, 20, 55).
event([cd, ' ', mail, (/), business, (/), busin], busin, thursday, 9, may, 1991, 13, 20, 21).
event([cd], home, thursday, 9, may, 1991, 13, 19, 13).
event([rmdir, ' ', personal], mail, thursday, 9, may, 1991, 13, 18, 56).
event([cd, ' ', mail], mail, thursday, 9, may, 1991, 13, 18, 31).
event([rmdir, ' ', persout], personal, thursday, 9, may, 1991, 13, 18, 17).
event([cd, ' ', personal], personal, thursday, 9, may, 1991, 13, 18, 2).
event([mv, ' ', tojim, ' ', personal, (/), mail, (/), home], persout, thursday, 9, may, 1991, 13, 17, 46).
event([chmod, ' ', 700, ' ', tojim], persout, thursday, 9, may, 1991, 13, 17, 18).
event([cd, ' ', persout], persout, thursday, 9, may, 1991, 13, 16, 51).
event([rmdir, ' ', persin], personal, thursday, 9, may, 1991, 13, 16, 34).
event([cd, ' ', personal], personal, thursday, 9, may, 1991, 13, 16, 12).
event([mv, ' ', fromjim, ' ', personal, (/), mail, (/), home], persin, thursday, 9, may, 1991, 13, 15, 38).
event([mv, ' ', frommary, ' ', personal, (/), mail, (/), home], persin, thursday, 9, may, 1991, 13, 15, 13).
event([cd, ' ', mail, (/), personal, (/), persin], persin, thursday, 9, may, 1991, 13, 14, 2).
event([mv, ' ', smiths, ' ', letters], home, thursday, 9, may, 1991, 13,

12, 50).
event([cd], home, thursday, 9, may, 1991, 13, 12, 28).
event([mv, ' ', smiths, ' ', addresses, (/), home], persaddr, thursday, 9, may, 1991, 13, 12, 23).
event([cd, ' ', addresses, (/), persaddr], persaddr, thursday, 9, may, 1991, 13, 11, 54).
event([cd], home, thursday, 9, may, 1991, 13, 10, 27).
event([cd, ' ', addresses, (/), persaddr], persaddr, thursday, 9, may, 1991, 13, 9, 35).
event([cd], home, thursday, 9, may, 1991, 13, 9, 18).
event([cd, ' ', personal], personal, thursday, 9, may, 1991, 13, 8, 45).
event([cd, ' ', mail], mail, thursday, 9, may, 1991, 13, 3, 55).
event([cd, ' ', personal], personal, thursday, 9, may, 1991, 13, 3, 0).
event([cd, ' ', mail], mail, thursday, 9, may, 1991, 13, 2, 7).
event([cd], home, thursday, 9, may, 1991, 13, 1, 6).
event([rm, ' ', fromeric], mail, thursday, 9, may, 1991, 12, 59, 48).
event([cd, ' ', mail], mail, thursday, 9, may, 1991, 12, 59, 23).
event([cd], home, thursday, 9, may, 1991, 12, 58, 27).
event([rm, ' ', bloggs], persaddr, thursday, 9, may, 1991, 12, 57, 12).
event([rm, ' ', johnsons], persaddr, thursday, 9, may, 1991, 12, 57, 4).
event([cd, ' ', addresses, (/), persaddr], persaddr, thursday, 9, may, 1991, 12, 56, 36).
event([cd], home, thursday, 9, may, 1991, 12, 55, 5).
event([cd, ' ', persaddr], persaddr, thursday, 9, may, 1991, 12, 44, 54).
event([cd, ' ', addresses], addresses, thursday, 9, may, 1991, 12, 44, 43).
event([cd], home, thursday, 9, may, 1991, 12, 44, 37).
event([cd, ' ', letters], letters, thursday, 9, may, 1991, 12, 44, 2).
event([mkdir, ' ', letters], home, thursday, 9, may, 1991, 12, 43, 29).
event([rm, ' ', temp], home, thursday, 9, may, 1991, 12, 41, 30).
event([rmdir, ' ', mailin], home, thursday, 9, may, 1991, 12, 37, 46).
event([cd], home, thursday, 9, may, 1991, 12, 37, 36).
event([cd, ' ', mailin], mailin, thursday, 9, may, 1991, 12, 37, 8).
event([cd], home, thursday, 9, may, 1991, 12, 36, 52).
event([cd, ' ', mailin], mailin, thursday, 9, may, 1991, 12, 36, 1).
event([cd, ' ', personal, (/), mail, (/), home], home, thursday, 9, may, 1991, 12, 35, 47).
event([cd, ' ', persin], persin, thursday, 9, may, 1991, 12, 34, 47).
event([cd, ' ', personal], personal, thursday, 9, may, 1991, 12, 34, 41).
event([cd, ' ', persout], persout, thursday, 9, may, 1991, 12, 34, 0).
event([cd, ' ', personal], personal, thursday, 9, may, 1991, 12, 33, 33).
event([cd, ' ', mail], mail, thursday, 9, may, 1991, 12, 32, 24).
event([cd], home, thursday, 9, may, 1991, 12, 32, 1).
event([cd, ' ', business], business, thursday, 9, may, 1991, 12, 31, 58).
event([chmod, ' ', 700, ' ', complaintin], busin, thursday, 9, may, 1991, 12, 31, 40).
event([mkfile, ' ', thanksin], busin, thursday, 9, may, 1991, 12, 31, 7).
event([cd, ' ', busin], busin, thursday, 9, may, 1991, 12, 30, 0).
event([cd, ' ', business], business, thursday, 9, may, 1991, 12, 25, 59).
event([cd, ' ', mail], mail, thursday, 9, may, 1991, 12, 25, 43).
event([cd, ' ', mail, (/), home], home, thursday, 9, may, 1991, 12, 22, 44).
event([cd, ' ', business], business, thursday, 9, may, 1991, 12, 22, 25).
event([cd, ' ', busin], busin, thursday, 9, may, 1991, 12, 21, 5).
event([cd, ' ', business], business, thursday, 9, may, 1991, 12, 20, 54).
event([cd, ' ', busout], busout, thursday, 9, may, 1991, 12, 20, 15).
event([cd, ' ', business], business, thursday, 9, may, 1991, 12, 19,

```
   42).
event([chmod, ' ', 700, ' ', personal], mail, thursday, 9, may, 1991,
   12, 19, 27).
event([chmod, ' ', 700, ' ', business], mail, thursday, 9, may, 1991,
   12, 19, 3).
event([chmod, ' ', 400, ' ', business], mail, thursday, 9, may, 1991,
   12, 18, 32).
event([chmod, ' ', 300, ' ', business], mail, thursday, 9, may, 1991,
   12, 18, 3).
event([cd, ' ', mail], mail, thursday, 9, may, 1991, 12, 17, 17).
event([cd], home, thursday, 9, may, 1991, 12, 16, 34).
event([cd, ' ', busaddr], busaddr, thursday, 9, may, 1991, 12, 15, 43).
event([cd, ' ', addresses], addresses, thursday, 9, may, 1991, 12, 15,
   29).
event([cd, ' ', persaddr], persaddr, thursday, 9, may, 1991, 12, 13,
   20).
event([chmod, ' ', 700, ' ', persaddr], addresses, thursday, 9, may,
   1991, 12, 12, 57).
event([cd, ' ', addresses], addresses, thursday, 9, may, 1991, 12, 9,
   20).
event([cd], home, thursday, 9, may, 1991, 12, 9, 7).
event([cd, ' ', mailin], mailin, thursday, 9, may, 1991, 12, 7, 54).
event([cd], home, thursday, 9, may, 1991, 12, 7, 23).
event([cd, ' ', personal, (/), mail], mail, thursday, 9, may, 1991, 12,
   7, 0).
event([cd, ' ', personal], mail, thursday, 9, may, 1991, 12, 3, 54).
event([cd, ' ', mail], mail, thursday, 9, may, 1991, 12, 3, 42).
event([cd, ' ', business], business, thursday, 9, may, 1991, 11, 59,
   19).
event([cd, ' ', mail], mail, thursday, 9, may, 1991, 11, 58, 4).
event([cd, ' ', addresses, (/), home], home, thursday, 9, may, 1991, 11,
   57, 34).
event([cd, ' ', busaddr], busaddr, thursday, 9, may, 1991, 11, 55, 47).
event([cd, ' ', persaddr, (/), addresses], addresses, thursday, 9, may
   1991, 11, 55, 26).
event([cd, ' ', home, (/), addresses], addresses, thursday, 9, may,
   1991, 11, 52, 20).
event([cd, ' ', mailin], mailin, thursday, 9, may, 1991, 11, 51, 48).
event([cd], home, thursday, 9, may, 1991, 11, 38, 46).
event([cd], home, thursday, 9, may, 1991, 11, 34, 37).
event([cd], home, thursday, 9, may, 1991, 11, 34, 22).
event([cd], home, thursday, 9, may, 1991, 11, 34, 8).

/* owner/1 */

owner(s3).


/* file/3 */

file(fromjim, persin, 0).
file(frommary, persin, 0).
file(tojim, persout, 0).
file(complaintin, busin, 400).
file(enquirout, busout, 700).
file(suppaddr, busaddr, 700).
file(custaddr, busaddr, 700).
file(smiths, persaddr, 700).
file(joneses, persaddr, 700).
file(bloggs, persaddr, 700).
file(johnsons, persaddr, 700).
file(addout, busout, 700).


/* dir/4 */
```

```
dir(root, 0, nul, [home]).
dir(persin, 700, personal, [frommary, fromjim]).
dir(persout, 700, personal, [tojim]).
dir(busin, 700, business, [complaintin]).
dir(addresses, 700, home, [persaddr, busaddr]).
dir(busaddr, 700, addresses, [custaddr, suppaddr]).
dir(busout, 700, business, [addout, enquirout]).
dir(persaddr, 200, addresses, [johnsons, bloggs, joneses, smiths]).
dir(home, 700, root, [addresses, mail]).
dir(mail, 700, home, [business, personal]).
dir(personal, 700, mail, [persout, persin]).
dir(business, 700, mail, [busout, busin]).


/* errorcount/2 */

errorcount(range, 0).
errorcount(uniqueness, 0).
errorcount(write, 0).
errorcount(execute, 0).
errorcount(syntax, 5).
errorcount(read, 3).
errorcount(logical, 2).
errorcount(nonempty, 5).
errorcount(duplication, 5).
errorcount(lexical, 8).
errorcount(reference, 61).


/* totaltime/3 */

totaltime(1, 14, 10).


/* event/9 */

event([mv, ' ', business, (/), busout, ' ', addresses], mail, thursday,
   16,  may, 1991, 15, 34, 25).
event([cd, ' ', mail], mail, thursday, 16, may, 1991, 15, 30, 32).
event([cd, ' ', home], home, thursday, 16, may, 1991, 15, 28, 38).
event([chmod, ' ', 700, ' ', business], mail, thursday, 16, may, 1991,
   15, 26, 3).
event([mv, ' ', business, ' ', busmail], mail, thursday, 16, may, 1991,
   15,  24, 48).
event([chmod, ' ', 500, ' ', business], mail, thursday, 16, may, 1991,
   15, 22,  37).
event([chmod, ' ', 100, ' ', business], mail, thursday, 16, may, 1991,
   15, 22, 0).
event([cd, ' ', mail], mail, thursday, 16, may, 1991, 15, 21, 9).
event([cd], home, thursday, 16, may, 1991, 15, 20, 44).
event([cd, ' ', home], home, thursday, 16, may, 1991, 15, 19, 12).
event([rm, ' ', persin], mail, thursday, 16, may, 1991, 15, 18, 16).
event([chmod, ' ', 700, ' ', personal], mail, thursday, 16, may, 1991,
   15, 18, 6).
event([chmod, ' ', 400, ' ', personal], mail, thursday, 16, may, 1991,
   15, 16, 51).
event([rm, ' ', persout], mail, thursday, 16, may, 1991, 15, 15, 36).
event([chmod, ' ', 700, ' ', personal], mail, thursday, 16, may, 1991,
   15, 14, 56).
event([rm, ' ', fromeric], mail, thursday, 16, may, 1991, 15, 14, 8).
event([cd, ' ', mail], mail, thursday, 16, may, 1991, 15, 13, 41).
event([rm, ' ', temp], home, thursday, 16, may, 1991, 15, 12, 49).
event([rmdir, ' ', mailin], home, thursday, 16, may, 1991, 15, 11, 1).
event([mv, ' ', mailin, ' ', personal], home, thursday, 16, may, 1991,
```

```
   15, 10, 1).
event([cd], home, thursday, 16, may, 1991, 15, 9, 36).
event([cd, ' ', home], home, thursday, 16, may, 1991, 15, 4, 54).
event([chmod, ' ', 700, ' ', business], mail, thursday, 16, may, 1991,
   15, 3, 40).
event([chmod, ' ', 700, ' ', business], mail, thursday, 16, may, 1991,
   15, 1, 57).
event([cd, ' ', mail], mail, thursday, 16, may, 1991, 15, 0, 30).
event([rm, ' ', business], home, thursday, 16, may, 1991, 14, 57, 32).
event([cd, ' ', home], home, thursday, 16, may, 1991, 14, 51, 36).
event([cd, ' ', mailin], mailin, thursday, 16, may, 1991, 14, 51, 16).
event([cd, ' ', home], home, thursday, 16, may, 1991, 14, 51, 11).
event([cd, ' ', mail], mail, thursday, 16, may, 1991, 14, 50, 26).
event([cd, ' ', home], home, thursday, 16, may, 1991, 14, 50, 22).
event([cd, ' ', addresses], addresses, thursday, 16, may, 1991, 14, 48,
   26).
event([cd, ' ', home], home, thursday, 16, may, 1991, 14, 44, 36).
event([cd, ' ', mailin], mailin, thursday, 16, may, 1991, 14, 43, 18).

/* owner/1 */

owner(s3).


/* file/3 */

file(temp, home, 700).
file(fromjim, persin, 0).
file(frommary, persin, 0).
file(tojim, persout, 0).
file(complaintin, busin, 400).
file(enquirout, busout, 700).
file(suppaddr, busaddr, 700).
file(custaddr, busaddr, 700).
file(smiths, persaddr, 700).
file(joneses, persaddr, 700).
file(bloggs, persaddr, 700).
file(johnsons, persaddr, 700).
file(fromeric, mail, 700).
file(addout, busout, 700).


/* dir/4 */

dir(root, 0, nul, [home]).
dir(persin, 700, personal, [frommary, fromjim]).
dir(persout, 700, personal, [tojim]).
dir(personal, 0, mail, [persout, persin]).
dir(busin, 700, business, [complaintin]).
dir(addresses, 700, home, [persaddr, busaddr]).
dir(busaddr, 700, addresses, [custaddr, suppaddr]).
dir(mail, 700, home, [fromeric, business, personal]).
dir(busout, 700, business, [addout, enquirout]).
dir(persaddr, 200, addresses, [johnsons, bloggs, joneses, smiths]).
dir(home, 700, root, [addresses, mail, temp]).
dir(business, 400, mail, [busout, busin]).


/* errorcount/2 */

errorcount(range, 0).
errorcount(nonempty, 0).
errorcount(uniqueness, 0).
errorcount(read, 0).
errorcount(write, 0).
```

```
errorcount(execute, 0).
errorcount(duplication, 1).
errorcount(logical, 3).
errorcount(lexical, 8).
errorcount(syntax, 1).
errorcount(reference, 9).


/* totaltime/3 */

totaltime(0, 41, 46).


/* event/9 */

event([cd], home, thursday, 16, may, 1991, 14, 45, 54).
event([chmod, ' ', 400, ' ', mail, (/), business], home, thursday, 16,
   may, 1991, 14, 44, 5).
event([cd], home, thursday, 16, may, 1991, 14, 35, 32).
event([rmdir, ' ', mailin], home, thursday, 16, may, 1991, 14, 25, 54).
event([cd], home, thursday, 16, may, 1991, 14, 17, 11).
event([cd, ' ', mailin], mailin, thursday, 16, may, 1991, 14, 11, 50).

/* owner/1 */

owner(s10).


/* file/3 */

file(fromjim, persin, 0).
file(frommary, persin, 0).
file(tojim, persout, 0).
file(complaintin, busin, 400).
file(enquirout, busout, 700).
file(smiths, persaddr, 700).
file(joneses, persaddr, 700).
file(bloggs, persaddr, 700).
file(johnsons, persaddr, 700).
file(fromeric, mail, 700).
file(addout, busout, 700).
file(accountsaddr, busaddr, 700).
file(custaddr, busaddr, 700).
file(suppaddr, busaddr, 700).


/* dir/4 */

dir(root, 0, nul, [home]).
dir(persin, 700, personal, [frommary, fromjim]).
dir(persout, 700, personal, [tojim]).
dir(personal, 0, mail, [persout, persin]).
dir(busin, 700, business, [complaintin]).
dir(addresses, 700, home, [persaddr, busaddr]).
dir(mail, 700, home, [fromeric, business, personal]).
dir(busout, 700, business, [addout, enquirout]).
dir(business, 100, mail, [busout, busin]).
dir(persmail, 700, personnel, []).
dir(personnel, 700, home, [persmail]).
dir(home, 700, root, [personnel, addresses, mail]).
dir(busaddr, 700, addresses, [suppaddr, custaddr, accountsaddr]).
dir(persaddr, 700, addresses, [johnsons, bloggs, joneses, smiths]).


/* errorcount/2 */
```

```
errorcount(range, 0).
errorcount(uniqueness, 0).
errorcount(read, 0).
errorcount(write, 0).
errorcount(duplication, 13).
errorcount(lexical, 9).
errorcount(reference, 29).
errorcount(logical, 4).
errorcount(execute, 4).
errorcount(nonempty, 9).
errorcount(syntax, 12).


/* totaltime/3 */

totaltime(1, 21, 16).


/* event/9 */

event([cd, ' ', persaddr], persaddr, wednesday, 15, may, 1991, 18, 51,
   11).
event([chmod, ' ', 700, ' ', persaddr], addresses, wednesday, 15, may,
   1991, 18, 51, 4).
event([cd, ' ', addresses], addresses, wednesday, 15, may, 1991, 18, 49,
   21).
event([cd, ' ', home], home, wednesday, 15, may, 1991, 18, 44, 41).
event([rmdir, ' ', addresses], addresses, wednesday, 15, may, 1991, 18,
   43, 42).
event([cd, ' ', addresses], addresses, wednesday, 15, may, 1991, 18, 43,
   1).
event([mkfile, ' ', suppaddr], busaddr, wednesday, 15, may, 1991, 18,
   42, 17).
event([rm, ' ', suppaddr], busaddr, wednesday, 15, may, 1991, 18, 42,
   0).
event([mkfile, ' ', custaddr], busaddr, wednesday, 15, may, 1991, 18,
   41, 43).
event([rm, ' ', custaddr], busaddr, wednesday, 15, may, 1991, 18, 41,
   26).
event([mkfile, ' ', accountsaddr], busaddr, wednesday, 15, may, 1991,
   18, 41, 4).
event([cd, ' ', busaddr], busaddr, wednesday, 15, may, 1991, 18, 39,
   57).
event([cd, ' ', addresses], addresses, wednesday, 15, may, 1991, 18, 32,
   49).
event([rm, ' ', temp], home, wednesday, 15, may, 1991, 18, 32, 5).
event([rmdir, ' ', mailin], home, wednesday, 15, may, 1991, 18, 30, 44).
event([cd, ' ', home], home, wednesday, 15, may, 1991, 18, 29, 9).
event([cd, ' ', personnel], personnel, wednesday, 15, may, 1991, 18, 26,
   55).
event([cd, ' ', persmail], persmail, wednesday, 15, may, 1991, 18, 23,
   38).
event([cd, ' ', personnel], personnel, wednesday, 15, may, 1991, 18, 16,
   21).
event([cd, ' ', persmail], persmail, wednesday, 15, may, 1991, 18, 7,
   52).
event([mkdir, ' ', persmail], personnel, wednesday, 15, may, 1991, 18,
   2, 47).
event([cd, ' ', personnel], personnel, wednesday, 15, may, 1991, 18, 1,
   59).
event([mkdir, ' ', personnel], home, wednesday, 15, may, 1991, 18, 0,
   58).
event([cd, ' ', home], home, wednesday, 15, may, 1991, 17, 47, 41).
event([cd, ' ', mail], mail, wednesday, 15, may, 1991, 17, 44, 44).
```

Appendix J
Protocol Log Files

Appendix J contains log files for all subjects involved in the protocol analysis exercise. The *event* clauses represent successful commands issued and the *nonevent* clauses relate to errors made during task completion. All events are time stamped in order to enable correlation with simultaneous video tape recordings and also show the erroneous command, the error type, the erroneous item and its alleged parent and the current working directory (*cwd*).

```
/* owner/1 */

owner(S1).


/* totaltime/3 */

totaltime(2, 9, 44).


/* file/3 */

file(smiths, persaddr, 700).
file(joneses, persaddr, 700).
file(bloggs, persaddr, 700).
file(johnsons, persaddr, 700).
file(enquirout, temp, 700).
file(complaintin, temp, 700).
file(custaddr, busaddr, 700).
file(suppaddr, busaddr, 700).
file(accountsaddr, busaddr, 700).
file(tojim, temp, 700).
file(frommary, temp, 700).
file(fromjim, temp, 700).


/* dir/4 */

dir(root, 0, nul, [home]).
dir(persaddr, 700, addresses, [johnsons, bloggs, joneses, smiths]).
dir(addresses, 700, home, [persaddr]).
dir(business, 700, home, [busaddr]).
dir(busaddr, 700, business, [accountsaddr, suppaddr, custaddr]).
dir(temp, 700, home, [fromjim, frommary, tojim, complaintin,
    enquirout]).
dir(personal, 700, home, []).
dir(home, 700, root, [personal, business, temp, addresses]).


/* event/9 */

event([rmdir, ' ', mail], home, monday, 20, january, 1992, 20, 2, 21).
event([ls, ' ', '-l', ' ', temp], home, monday, 20, january, 1992, 20,
    2, 9).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 20, 1, 53).
event([pwd], home, monday, 20, january, 1992, 20, 1, 48).
event([ls, ' ', '-l', ' ', mail], home, monday, 20, january, 1992, 20,
    1, 29).
event([rm, ' ', mail, (/), fromeric], home, monday, 20, january, 1992,
    20, 1,23).
event([ls, ' ', '-l', ' ', mail], home, monday, 20, january, 1992, 20,
    0, 50).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 20, 0, 22).
event([mkdir, ' ', personal], home, monday, 20, january, 1992, 20, 0,
    16).
event([cd], home, monday, 20, january, 1992, 20, 0, 2).
event([rmdir, ' ', mail, (/), personal], home, monday, 20, january,
    1992, 19, 59, 53).
event([ls, ' ', '-l', ' ', mail, (/), personal], home, monday, 20,
    january,1992, 19, 59, 34).
event([rmdir, ' ', mail, (/), personal, (/), persin], home, monday, 20,
    january, 1992, 19, 59, 18).
event([ls, ' ', '-l', ' ', temp], home, monday, 20, january, 1992, 19,
    59, 1).
event([mv, ' ', mail, (/), personal, (/), persin, (/), fromjim, ' ',
```

temp], home, monday, 20, january, 1992, 19, 58, 56).
event([mv, ' ', mail, (/), personal, (/), persin, (/), frommary, ' ', temp], home, monday, 20, january, 1992, 19, 58, 45).
event([rmdir, ' ', mail, (/), personal, (/), persout], home, monday, 20, january, 1992, 19, 57, 47).
event([ls, ' ', '-l', ' ', temp], home, monday, 20, january, 1992, 19, 57, 18).
event([mv, ' ', mail, (/), personal, (/), persout, (/), tojim, ' ', temp], home, monday, 20, january, 1992, 19, 57, 12).
event([pwd], home, monday, 20, january, 1992, 19, 55, 59).
event([ls, ' ', '-l', ' ', temp], home, monday, 20, january, 1992, 19, 54, 41).
event([cd], home, monday, 20, january, 1992, 19, 53, 12).
event([pwd], temp, monday, 20, january, 1992, 19, 53, 9).
event([cd, ' ', temp], temp, monday, 20, january, 1992, 19, 50, 45).
event([cd], home, monday, 20, january, 1992, 19, 49, 19).
event([pwd], busaddr, monday, 20, january, 1992, 19, 49, 6).
event([ls, ' ', '-l', ' ', business, (/), busaddr], busaddr, monday, 20, january, 1992, 19, 46, 50).
event([mkfile, ' ', accountsaddr], busaddr, monday, 20, january, 1992, 19, 46, 39).
event([cd, ' ', business, (/), busaddr], busaddr, monday, 20, january, 1992, 19, 46, 26).
event([pwd], home, monday, 20, january, 1992, 19, 46, 19).
event([ls, ' ', '-l', ' ', business, (/), busaddr], home, monday, 20, january, 1992, 19, 45, 55).
event([cd], home, monday, 20, january, 1992, 19, 45, 46).
event([pwd], temp, monday, 20, january, 1992, 19, 45, 44).
event([mv, ' ', suppaddr, ' ', home, (/), business, (/), busaddr], temp, monday, 20, january, 1992, 19, 45, 37).
event([ls, ' ', '-l'], temp, monday, 20, january, 1992, 19, 45, 11).
event([cd, ' ', temp], temp, monday, 20, january, 1992, 19, 45, 7).
event([pwd], home, monday, 20, january, 1992, 19, 45, 4).
event([ls, ' ', '-l', ' ', temp], home, monday, 20, january, 1992, 19, 44, 31).
event([cd], home, monday, 20, january, 1992, 19, 43, 47).
event([pwd], business, monday, 20, january, 1992, 19, 43, 41).
event([ls, ' ', '-l', ' ', busaddr], business, monday, 20, january, 1992, 19, 43, 28).
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 19, 43, 19).
event([cd, ' ', business], business, monday, 20, january, 1992, 19, 43, 17).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 19, 43, 10).
event([mv, ' ', temp, (/), custaddr, ' ', business, (/), busaddr], home, monday, 20, january, 1992, 19, 43, 2).
event([cd], home, monday, 20, january, 1992, 19, 42, 14).
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 19, 41, 56).
event([cd, ' ', business], business, monday, 20, january, 1992, 19, 41, 53).
event([pwd], home, monday, 20, january, 1992, 19, 41, 49).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 19, 41, 1).
event([cd], home, monday, 20, january, 1992, 19, 40, 56).
event([ls, ' ', '-l', ' ', addresses], home, monday, 20, january, 1992, 19, 40, 43).
event([rmdir, ' ', addresses, (/), busaddr], home, monday, 20, january, 1992, 19, 40, 31).
event([pwd], home, monday, 20, january, 1992, 19, 40, 23).
event([cd], home, monday, 20, january, 1992, 19, 40, 20).
event([ls, ' ', '-l'], temp, monday, 20, january, 1992, 19, 39, 59).
event([cd, ' ', temp], temp, monday, 20, january, 1992, 19, 39, 54).
event([cd], home, monday, 20, january, 1992, 19, 39, 51).
event([mv, ' ', suppaddr, ' ', addresses, (/), home, (/), temp], busaddr, monday, 20, january, 1992, 19, 39, 48).
event([ls, ' ', '-l'], busaddr, monday, 20, january, 1992, 19, 39, 22).
event([cd, ' ', busaddr], busaddr, monday, 20, january, 1992, 19, 39,

```
17).
event([pwd], addresses, monday, 20, january, 1992, 19, 39, 8).
event([ls, ' ', '-l'], addresses, monday, 20, january, 1992, 19, 38,
    14).
event([cd, ' ', addresses], addresses, monday, 20, january, 1992, 19,
    38, 10).
event([cd, ' ', home], home, monday, 20, january, 1992, 19, 38, 4).
event([pwd], business, monday, 20, january, 1992, 19, 38, 1).
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 19, 37, 31).
event([cd, ' ', business], business, monday, 20, january, 1992, 19, 37,
    28).
event([cd], home, monday, 20, january, 1992, 19, 37, 0).
event([pwd], business, monday, 20, january, 1992, 19, 36, 57).
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 19, 36, 44).
event([cd, ' ', business], business, monday, 20, january, 1992, 19, 36,
    42).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 19, 36, 39).
event([pwd], home, monday, 20, january, 1992, 19, 36, 33).
event([ls, ' ', '-l', ' ', temp], home, monday, 20, january, 1992, 19,
    35, 52).
event([cd], home, monday, 20, january, 1992, 19, 35, 46).
event([mv, ' ', custaddr, ' ', addresses, (/), home, (/), temp],
    busaddr, monday, 20, january, 1992, 19, 35, 24).
event([pwd], busaddr, monday, 20, january, 1992, 19, 33, 20).
event([ls, ' ', '-l'], busaddr, monday, 20, january, 1992, 19, 33, 1).
event([pwd], busaddr, monday, 20, january, 1992, 19, 32, 20).
event([ls, ' ', '-l'], busaddr, monday, 20, january, 1992, 19, 31, 14).
event([cd, ' ', busaddr], busaddr, monday, 20, january, 1992, 19, 31,
    8).
event([ls, ' ', '-l'], addresses, monday, 20, january, 1992, 19, 31, 4).
event([cd, ' ', addresses], addresses, monday, 20, january, 1992, 19,
    30, 59).
event([mv, ' ', addresses, (/), busaddr, ' ', business], home, monday,
    20, january, 1992, 19, 29, 1).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 19, 27, 56).
event([pwd], home, monday, 20, january, 1992, 19, 27, 45).
event([cd], home, monday, 20, january, 1992, 19, 27, 42).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 19, 23, 49).
event([mkdir, ' ', business], home, monday, 20, january, 1992, 19, 23,
    45).
event([cd], home, monday, 20, january, 1992, 19, 23, 40).
event([pwd], mail, monday, 20, january, 1992, 19, 23, 36).
event([ls, ' ', '-l'], mail, monday, 20, january, 1992, 19, 23, 26).
event([rmdir, ' ', business], mail, monday, 20, january, 1992, 19, 23,
    22).
event([ls, ' ', '-l'], mail, monday, 20, january, 1992, 19, 23, 16).
event([cd, ' ', mail], mail, monday, 20, january, 1992, 19, 23, 8).
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 19, 23, 1).
event([rmdir, ' ', busin], business, monday, 20, january, 1992, 19, 22,
    58).
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 19, 22, 50).
event([cd, ' ', mail, (/), business], business, monday, 20, january,
    1992, 19, 22, 48).
event([ls, ' ', '-l', ' ', temp], home, monday, 20, january, 1992, 19,
    22, 19).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 19, 22, 13).
event([cd], home, monday, 20, january, 1992, 19, 22, 5).
event([mv, ' ', complaintin, ' ', business, (/), mail, (/), home, (/),
    temp], busin, monday, 20, january, 1992, 19, 21, 30).
event([ls, ' ', '-l'], busin, monday, 20, january, 1992, 19, 19, 26).
event([ls, ' ', '-l'], busin, monday, 20, january, 1992, 19, 18, 10).
event([cd, ' ', busin], busin, monday, 20, january, 1992, 19, 18, 7).
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 19, 17, 56).
event([rmdir, ' ', busout], business, monday, 20, january, 1992, 19, 17,
    53).
```

```
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 19, 17, 38).
event([pwd], business, monday, 20, january, 1992, 19, 17, 31).
event([cd, ' ', business], business, monday, 20, january, 1992, 19, 17,
    29).
event([ls, ' ', '-l'], busout, monday, 20, january, 1992, 19, 17, 20).
event([rm, ' ', addout], busout, monday, 20, january, 1992, 19, 17, 17).
event([ls, ' ', '-l'], busout, monday, 20, january, 1992, 19, 17, 8).
event([cd, ' ', mail, (/), business, (/), busout], busout, monday, 20,
    january, 1992, 19, 17, 5).
event([pwd], home, monday, 20, january, 1992, 19, 15, 45).
event([ls], home, monday, 20, january, 1992, 19, 15, 41).
event([cd], home, monday, 20, january, 1992, 19, 15, 37).
event([pwd], temp, monday, 20, january, 1992, 19, 15, 32).
event([pwd], temp, monday, 20, january, 1992, 19, 15, 24).
event([ls, ' ', '-l'], temp, monday, 20, january, 1992, 19, 14, 40).
event([cd, ' ', temp], temp, monday, 20, january, 1992, 19, 14, 37).
event([pwd], home, monday, 20, january, 1992, 19, 14, 34).
event([mv, ' ', mail, (/), business, (/), busout, (/), enquirout, ' ',
    temp], home, monday, 20, january, 1992, 19, 14, 29).
event([pwd], home, monday, 20, january, 1992, 19, 12, 58).
event([mkdir, ' ', temp], home, monday, 20, january, 1992, 19, 12, 16).
event([pwd], home, monday, 20, january, 1992, 19, 11, 57).
event([cd], home, monday, 20, january, 1992, 19, 11, 53).
event([rmdir, ' ', temp], busout, monday, 20, january, 1992, 19, 11,
    51).
event([pwd], busout, monday, 20, january, 1992, 19, 11, 37).
event([ls, ' ', '-l'], busout, monday, 20, january, 1992, 19, 11, 9).
event([mkdir, ' ', temp], busout, monday, 20, january, 1992, 19, 11, 5).
event([ls, ' ', '-l'], busout, monday, 20, january, 1992, 19, 9, 41).
event([cd, ' ', busout], busout, monday, 20, january, 1992, 19, 9, 37).
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 19, 9, 32).
event([cd, ' ', business], business, monday, 20, january, 1992, 19, 9,
    29).
event([ls, ' ', '-l'], mail, monday, 20, january, 1992, 19, 9, 1).
event([cd, ' ', mail], mail, monday, 20, january, 1992, 19, 8, 57).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 19, 8, 44).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 19, 3, 27).
event([pwd], home, monday, 20, january, 1992, 19, 3, 19).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 19, 2, 43).
event([rm, ' ', temp], home, monday, 20, january, 1992, 19, 2, 39).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 19, 2, 7).
event([rmdir, ' ', mailin], home, monday, 20, january, 1992, 19, 2, 3).
event([cd], home, monday, 20, january, 1992, 18, 56, 52).
event([pwd], persin, monday, 20, january, 1992, 18, 56, 49).
event([ls, ' ', '-l'], persin, monday, 20, january, 1992, 18, 56, 44).
event([ls, ' ', '-l'], persin, monday, 20, january, 1992, 18, 51, 24).
event([cd, ' ', persin], persin, monday, 20, january, 1992, 18, 51, 20).
event([ls, ' ', '-l'], personal, monday, 20, january, 1992, 18, 51, 15).
event([cd, ' ', personal], personal, monday, 20, january, 1992, 18, 51,
    11).
event([ls, ' ', '-l'], persout, monday, 20, january, 1992, 18, 50, 47).
event([cd, ' ', persout], persout, monday, 20, january, 1992, 18, 50,
    43).
event([ls, ' ', '-l'], personal, monday, 20, january, 1992, 18, 50, 33).
event([cd, ' ', personal], personal, monday, 20, january, 1992, 18, 50,
    18).
event([ls, ' ', '-l'], mail, monday, 20, january, 1992, 18, 49, 31).
event([cd, ' ', mail], mail, monday, 20, january, 1992, 18, 49, 23).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 18, 48, 47).
event([ls, ' ', '-l', ' ', mail], home, monday, 20, january, 1992, 18,
    47, 5).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 18, 46, 45).
event([pwd], home, monday, 20, january, 1992, 18, 46, 37).
event([ls, ' ', '-l', ' ', addresses, (/), busaddr], home, monday, 20,
    january, 1992, 18, 45, 35).
```

```
event([ls, ' ', '-l', ' ', addresses], home, monday, 20, january, 1992,
    18, 44, 55).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 18, 44, 36).
event([cd], home, monday, 20, january, 1992, 18, 44, 28).
event([ls, ' ', '-l'], mailin, monday, 20, january, 1992, 18, 44, 16).
event([cd, ' ', mailin], mailin, monday, 20, january, 1992, 18, 44, 12).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 18, 43, 34).
event([cd], home, monday, 20, january, 1992, 18, 43, 29).
event([ls, ' ', '-l'], persin, monday, 20, january, 1992, 18, 43, 19).
event([chmod, ' ', 700, ' ', fromjim], persin, monday, 20, january,
    1992, 18, 43, 15).
event([chmod, ' ', 700, ' ', frommary], persin, monday, 20, january,
    1992, 18, 43, 6).
event([ls, ' ', '-l'], persin, monday, 20, january, 1992, 18, 42, 41).
event([cd, ' ', persin], persin, monday, 20, january, 1992, 18, 42, 35).
event([ls, ' ', '-l'], personal, monday, 20, january, 1992, 18, 42, 30).
event([cd, ' ', personal], personal, monday, 20, january, 1992, 18, 42,
    26).
event([pwd], persout, monday, 20, january, 1992, 18, 42, 18).
event([ls, ' ', '-l'], persout, monday, 20, january, 1992, 18, 42, 13).
event([chmod, ' ', 700, ' ', tojim], persout, monday, 20, january, 1992,
    18, 42, 9).
event([ls, ' ', '-l'], persout, monday, 20, january, 1992, 18, 41, 15).
event([cd, ' ', persout], persout, monday, 20, january, 1992, 18, 41,
    11).
event([ls, ' ', '-l'], personal, monday, 20, january, 1992, 18, 40, 45).
event([cd, ' ', personal], personal, monday, 20, january, 1992, 18, 40,
    42).
event([ls, ' ', '-l'], mail, monday, 20, january, 1992, 18, 40, 37).
event([chmod, ' ', 700, ' ', personal], mail, monday, 20, january, 1992,
    18, 40, 33).
event([ls, ' ', '-l'], mail, monday, 20, january, 1992, 18, 40, 7).
event([cd, ' ', mail], mail, monday, 20, january, 1992, 18, 40, 3).
event([pwd], business, monday, 20, january, 1992, 18, 40, 0).
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 18, 39, 48).
event([cd, ' ', business], business, monday, 20, january, 1992, 18, 39,
    45).
event([ls, ' ', '-l'], busin, monday, 20, january, 1992, 18, 39, 13).
event([chmod, ' ', 700, ' ', complaintin], busin, monday, 20, january,
    1992, 18, 38, 56).
event([pwd], busin, monday, 20, january, 1992, 18, 38, 39).
event([ls, ' ', '-l'], busin, monday, 20, january, 1992, 18, 38, 31).
event([cd, ' ', busin], busin, monday, 20, january, 1992, 18, 38, 27).
event([cd, ' ', business], business, monday, 20, january, 1992, 18, 38,
    23).
event([ls, ' ', '-l'], busout, monday, 20, january, 1992, 18, 38, 2).
event([cd, ' ', busout], busout, monday, 20, january, 1992, 18, 37, 59).
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 18, 37, 48).
event([pwd], business, monday, 20, january, 1992, 18, 37, 31).
event([cd, ' ', business], business, monday, 20, january, 1992, 18, 36,
    34).
event([ls, ' ', '-l'], busin, monday, 20, january, 1992, 18, 36, 21).
event([cd, ' ', busin], busin, monday, 20, january, 1992, 18, 36, 18).
event([chmod, ' ', 700, ' ', busin], business, monday, 20, january,
    1992, 18, 36, 12).
event([cd, ' ', business], business, monday, 20, january, 1992, 18, 36,
    5).
event([pwd], busin, monday, 20, january, 1992, 18, 35, 57).
event([ls, ' ', '-l'], busin, monday, 20, january, 1992, 18, 34, 50).
event([cd, ' ', busin], busin, monday, 20, january, 1992, 18, 34, 45).
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 18, 34, 39).
event([pwd], business, monday, 20, january, 1992, 18, 34, 34).
event([cd, ' ', business], business, monday, 20, january, 1992, 18, 34,
    32).
event([ls, ' ', '-l'], busout, monday, 20, january, 1992, 18, 34, 7).
```

```
event([pwd], busout, monday, 20, january, 1992, 18, 34, 1).
event([cd, ' ', busout], busout, monday, 20, january, 1992, 18, 33, 55).
event([ls, ' ', '-l'], business, monday, 20, january, 1992, 18, 32, 59).
event([cd, ' ', business], business, monday, 20, january, 1992, 18, 32,
    55).
event([chmod, ' ', 700, ' ', business], mail, monday, 20, january, 1992,
    18, 32, 49).
event([cd, ' ', mail], mail, monday, 20, january, 1992, 18, 32, 43).
event([pwd], business, monday, 20, january, 1992, 18, 32, 30).
event([pwd], business, monday, 20, january, 1992, 18, 31, 46).
event([cd, ' ', business], business, monday, 20, january, 1992, 18, 31,
    42).
event([ls, ' ', '-l'], mail, monday, 20, january, 1992, 18, 29, 55).
event([cd, ' ', mail], mail, monday, 20, january, 1992, 18, 29, 50).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 18, 29, 32)..
event([cd], home, monday, 20, january, 1992, 18, 29, 26).
event([ls, ' ', '-l'], busaddr, monday, 20, january, 1992, 18, 28, 11).
event([cd, ' ', busaddr], busaddr, monday, 20, january, 1992, 18, 28,
    7).
event([ls, ' ', '-l'], addresses, monday, 20, january, 1992, 18, 27,
    59).
event([pwd], addresses, monday, 20, january, 1992, 18, 27, 26).
event([cd, ' ', addresses], addresses, monday, 20, january, 1992, 18,
    27, 15).
event([ls, ' ', '-l'], home, monday, 20, january, 1992, 18, 26, 13).
event([cd], home, monday, 20, january, 1992, 18, 26, 7).
event([pwd], persaddr, monday, 20, january, 1992, 18, 25, 59).
event([ls, ' ', '-l'], persaddr, monday, 20, january, 1992, 18, 24, 21).
event([ls, ' ', '-l'], addresses, monday, 20, january, 1992, 18, 23,
    16).
event([chmod, ' ', 700, ' ', persaddr], addresses, monday, 20, january,
    1992, 18, 23, 10).
event([pwd], addresses, monday, 20, january, 1992, 18, 22, 16).
event([ls, ' ', '-l'], addresses, monday, 20, january, 1992, 18, 21,
    49).
event([ls, ' ', '-l'], addresses, monday, 20, january, 1992, 18, 20,
    18).
event([pwd], addresses, monday, 20, january, 1992, 18, 20, 8).
event([chmod, ' ', 400, ' ', persaddr], addresses, monday, 20, january,
    1992, 18, 20, 4).
event([pwd], addresses, monday, 20, january, 1992, 18, 19, 43).
event([cd, ' ', addresses], addresses, monday, 20, january, 1992, 18,
    19, 40).
event([pwd], persaddr, monday, 20, january, 1992, 18, 17, 24).
event([cd, ' ', persaddr], persaddr, monday, 20, january, 1992, 18, 15,
    42).
event([ls], addresses, monday, 20, january, 1992, 18, 15, 31).
event([cd, ' ', addresses], addresses, monday, 20, january, 1992, 18,
    15, 29).
event([ls], home, monday, 20, january, 1992, 18, 15, 22).
event([pwd], home, monday, 20, january, 1992, 18, 14, 52).
event([cd], home, monday, 20, january, 1992, 18, 14, 48).
event([pwd], persaddr, monday, 20, january, 1992, 18, 12, 15).
event([cd, ' ', persaddr], persaddr, monday, 20, january, 1992, 18, 12,
    11).
event([ls, ' ', '-l'], addresses, monday, 20, january, 1992, 18, 11,
    45).
event([chmod, ' ', 100, ' ', persaddr], addresses, monday, 20, january,
    1992, 18, 11, 35).
event([ls, ' ', '-l'], addresses, monday, 20, january, 1992, 18, 8, 30).
event([pwd], addresses, monday, 20, january, 1992, 18, 8, 24).
event([ls], addresses, monday, 20, january, 1992, 18, 7, 6).
event([pwd], addresses, monday, 20, january, 1992, 18, 6, 8).
event([ls, ' ', '-l'], addresses, monday, 20, january, 1992, 18, 3, 12).
event([ls], addresses, monday, 20, january, 1992, 18, 2, 51).
```

```
event([cd, ' ', addresses], addresses, monday, 20, january, 1992, 18, 2,
    48).
event([ls], home, monday, 20, january, 1992, 18, 2, 40).
event([pwd], home, monday, 20, january, 1992, 18, 2, 34).
event([cd], home, monday, 20, january, 1992, 18, 2, 31).
event([pwd], mailin, monday, 20, january, 1992, 18, 2, 26).
event([ls, ' ', '-l'], mailin, monday, 20, january, 1992, 17, 59, 48).
event([ls], mailin, monday, 20, january, 1992, 17, 58, 41).
event([pwd], mailin, monday, 20, january, 1992, 17, 58, 7).
event([ls], mailin, monday, 20, january, 1992, 17, 58, 3).
event([pwd], mailin, monday, 20, january, 1992, 17, 57, 46).
event([ls], mailin, monday, 20, january, 1992, 17, 57, 20).
event([cd, ' ', mailin], mailin, monday, 20, january, 1992, 17, 57, 14).
event([pwd], home, monday, 20, january, 1992, 17, 55, 49).
event([ls], home, monday, 20, january, 1992, 17, 54, 5).
event([chmod, ' ', 200, ' ', home, (/), addresses, (/), persaddr], mail,
    wednesday, 8, jan, 1992, 10, 26, 27).
event([chmod, ' ', 100, ' ', business], mail, wednesday, 8, jan, 1992,
    10, 26, 27).
event([mkfile, ' ', business, (/), busout, (/), addout], mail,
    wednesday, 8, jan, 1992, 10, 26, 27).
event([mkfile, ' ', fromeric], mail, wednesday, 8, jan, 1992, 10, 26,
    27).
event([cd, ' ', mail], mail, wednesday, 8, jan, 1992, 10, 26, 27).
event([mkdir, ' ', mailin], home, wednesday, 8, jan, 1992, 10, 26, 27).
event([cd], home, wednesday, 8, jan, 1992, 10, 26, 26).
event([mkfile, ' ', johnsons], persaddr, wednesday, 8, jan, 1992, 10,
    26, 26).
event([mkfile, ' ', bloggs], persaddr, wednesday, 8, jan, 1992, 10, 26,
    26).
event([mkfile, ' ', joneses], persaddr, wednesday, 8, jan, 1992, 10, 26,
    26).
event([mkfile, ' ', smiths], persaddr, wednesday, 8, jan, 1992, 10, 26,
    26).
event([cd, ' ', addresses, (/), persaddr], persaddr, wednesday, 2, may,
    1991, 10, 26, 26).
event([mkfile, ' ', custaddr], busaddr, wednesday, 8, jan, 1992, 10, 26,
    25).
event([mkfile, ' ', suppaddr], busaddr, wednesday, 8, jan, 1992, 10, 26,
    25).
event([cd, ' ', busaddr], busaddr, wednesday, 8, jan, 1992, 10, 26, 25).
event([mkdir, ' ', persaddr], addresses, wednesday, 8, jan, 1992, 10,
    26, 25).
event([mkdir, ' ', busaddr], addresses, wednesday, 8, jan, 1992, 10, 26,
    25).
event([cd, ' ', business, (/), mail, (/), home, (/), addresses],
    addresses, wednesday, 8, jan, 1992, 10, 26, 25).
event([mkfile, ' ', enquirout], busout, wednesday, 8, jan, 1992, 10, 26,
    25).
event([cd, ' ', business, (/), busout], busout, wednesday, 8, jan, 1992,
    10, 26, 24).
event([chmod, ' ', 400, ' ', complaintin], busin, wednesday, 2, may,
    1991, 10, 26, 24).
event([mkfile, ' ', complaintin], busin, wednesday, 8, jan, 1992, 10,
    26, 24).
event([cd, ' ', busin], busin, wednesday, 8, jan, 1992, 10, 26, 24).
event([mkdir, ' ', busout], business, wednesday, 8, jan, 1992, 10, 26,
    24).
event([mkdir, ' ', busin], business, wednesday, 8, jan, 1992, 10, 26,
    24).
event([cd, ' ', business], business, wednesday, 8, jan, 1992, 10, 26,
    23).
event([chmod, ' ', 0, ' ', personal], mail, wednesday, 8, jan, 1992, 10,
    26, 23).
event([cd, ' ', personal, (/), mail], mail, wednesday, 8, jan, 1992, 10,
```

```
    26, 23).
event([chmod, ' ', 0, ' ', tojim], persout, wednesday, 8, jan, 1992, 10,
    26, 23).
event([mkfile, ' ', tojim], persout, wednesday, 8, jan, 1992, 10, 26,
    23).
event([cd, ' ', personal, (/), persout], persout, wednesday, 2, may,
    1991, 10, 26, 23).
event([chmod, ' ', 0, ' ', frommary], persin, wednesday, 8, jan, 1992,
    10, 26, 23).
event([mkfile, ' ', frommary], persin, wednesday, 8, jan, 1992, 10, 26,
    22).
event([chmod, ' ', 0, ' ', fromjim], persin, wednesday, 8, jan, 1992,
    10, 26, 22).
event([mkfile, ' ', fromjim], persin, wednesday, 8, jan, 1992, 10, 26,
    22).
event([cd, ' ', persin], persin, wednesday, 8, jan, 1992, 10, 26, 22).
event([mkdir, ' ', persout], personal, wednesday, 8, jan, 1992, 10, 26,
    22).
event([mkdir, ' ', persin], personal, wednesday, 8, jan, 1992, 10, 26,
    21).
event([cd, ' ', personal], personal, wednesday, 8, jan, 1992, 10, 26,
    21).
event([mkdir, ' ', business], mail, wednesday, 8, jan, 1992, 10, 26,
    21).
event([mkdir, ' ', personal], mail, wednesday, 8, jan, 1992, 10, 26,
    21).
event([cd, ' ', mail], mail, wednesday, 8, jan, 1992, 10, 26, 21).
event([mkdir, ' ', addresses], home, wednesday, 8, jan, 1992, 10, 26,
    21).
event([mkdir, ' ', mail], home, wednesday, 8, jan, 1992, 10, 26, 21).


/* nonevent/12 */

nonevent([mv, ' ', mail, (/), personal, (/), persout, (/), tojim, ' ',
    persout, (/), personal, (/), mail, (/), home, (/), temp], reference,
    home, persout, home, monday, 20, january, 1992, 19, 55, 18).
nonevent([mv, ' ', mail, (/), personal, (/), persout, (/), tojim, ' ', '
    ', persout, (/), personal, (/), mail, (/), home, (/), temp],
    reference, home, ' ', home, monday, 20, january, 1992, 19, 54, 25).
nonevent([ls, ' ', '-l'], reference, temp, '-l', temp, monday, 20,
    january, 1992, 19, 50, 51).
nonevent([ls, ' ', '-l'], execute, home, temp, temp, monday, 20,
    january, 1992, 19, 50, 51).
nonevent([ls, ' ', '-l'], lexical, nul, nul, temp, monday, 20, january,
    1992, 19, 50, 50).
nonevent([';s'], lexical, nul, nul, temp, monday, 20, january, 1992, 19,
    50, 48).
nonevent([mkdir, ' ', personal], duplication, home, personal, home,
    monday, 20, january, 1992, 19, 49, 24).
nonevent([mv, ' ', temp, (/), suppadr, ' ', business, (/), busaddr],
    reference, temp, suppadr, home, monday, 20, january, 1992, 19, 45,
    0).
nonevent([cd, ' ', tep], reference, home, tep, home, monday, 20,
    january, 1992, 19, 44, 24).
nonevent([mv, ' ', temp, (/), suppadr, ' ', business, (/), busaddr],
    reference, temp, suppadr, home, monday, 20, january, 1992, 19, 44,
    14).
nonevent([mv, ' ', home, (/), temp, (/), custaddr, ' ', business, (/),
    busaddr], reference, home, home, home, monday, 20, january, 1992,
    19, 42, 42).
nonevent([mkdir, ' ', business, (/), busaddr], write, home, business,
    home, monday, 20, january, 1992, 19, 41, 30).
nonevent([mkdir, ' ', business, (/), busaddr], lexical, nul, nul, home,
    monday, 20, january, 1992, 19, 41, 29).
```

nonevent([md], lexical, nul, nul, home, monday, 20, january, 1992, 19, 41, 19).
nonevent([rmdir, ' ', addresses, (/), busaddr], reference, temp, addresses, temp, monday, 20, january, 1992, 19, 40, 14).
nonevent([mv, ' ', suppaddr], syntax, nul, nul, addresses, monday, 20, january, 1992, 19, 39, 5).
nonevent([rmdir, ' ', busaddr], nonempty, nul, nul, addresses, monday, 20, january, 1992, 19, 38, 20).
nonevent([cd, ' ', addresses], reference, business, addresses, business, monday, 20, january, 1992, 19, 37, 55).
nonevent([cd, ' ', busaddr], reference, home, busaddr, home, monday, 20, january, 1992, 19, 37, 24).
nonevent([ls, ' ', '-l', ' ', temp], reference, busaddr, temp, busaddr, monday, 20, january, 1992, 19, 35, 41).
nonevent([ls, ' ', '-l', ' ', home, (/), temp], reference, busaddr, home, busaddr, monday, 20, january, 1992, 19, 35, 34).
nonevent([mv, ' ', custaddr, (/), addresses, ' ', temp], reference, busaddr, custaddr, busaddr, monday, 20, january, 1992, 19, 32, 55).
nonevent([mv, ' ', custaddr, ' ', addresses, ' ', temp], syntax, nul, nul, busaddr, monday, 20, january, 1992, 19, 32, 42).
nonevent([mv, ' ', custaddr, ' ', busaddr, (/), addresses, ' ', temp], reference, busaddr, busaddr, busaddr, monday, 20, january, 1992, 19, 32, 6).
nonevent([cd, ' ', ' ', addresses, (/), busaddr], reference, home, ' ', home, monday, 20, january, 1992, 19, 30, 49).
nonevent([ls, ' ', '-l', ' ', temp], reference, busin, temp, busin, monday, 20, january, 1992, 19, 21, 58).
nonevent([ls, ' ', '-l', ' ', home, (/), temp], reference, busin, home, busin, monday, 20, january, 1992, 19, 21, 53).
nonevent([ls, ' ', '-l'], reference, busin, '-l', busin, monday, 20, january, 1992, 19, 20, 3).
nonevent([ls, ' ', '-l'], execute, business, busin, busin, monday, 20, january, 1992, 19, 20, 3).
nonevent([ls, ' ', '-l'], lexical, nul, nul, busin, monday, 20, january, 1992, 19, 20, 3).
nonevent([pwd], lexical, nul, nul, busin, monday, 20, january, 1992, 19, 19, 51).
nonevent([pwd], lexical, nul, nul, busin, monday, 20, january, 1992, 19, 19, 50).
nonevent(['[wd'], lexical, nul, nul, busin, monday, 20, january, 1992, 19, 19, 48).
nonevent([mv, ' ', complaintin, ' ', home, (/), temp], reference, busin, home, busin, monday, 20, january, 1992, 19, 19, 9).
nonevent([mv, ' ', complaintin, ' ', temp], duplication, home, temp, busin, monday, 20, january, 1992, 19, 18, 32).
nonevent([cd, ' '], syntax, nul, nul, temp, monday, 20, january, 1992, 19, 15, 28).
nonevent([mv, ' ', mail, (/), business, (/), busout, (/), enquirout, ' ', home, (/), 'temp]'], reference, home, home, home, monday, 20, january, 1992, 19, 14, 6).
nonevent([cd, ' '], syntax, nul, nul, busout, monday, 20, january, 1992, 19, 11, 0).
nonevent([mkdir, ' ', business], duplication, home, business, home, monday, 20, january, 1992, 19, 4, 4).
nonevent([mkdir, ' ', business], duplication, home, business, home, monday, 20, january, 1992, 19, 3, 9).
nonevent([ls, ' ', '-l'], reference, home, '-l', home, monday, 20, january,1992, 18, 59, 32).
nonevent([ls, ' ', '-l'], execute, root, home, home, monday, 20, january,1992, 18, 59, 31).
nonevent([ls, ' ', '-l'], lexical, nul, nul, home, monday, 20, january, 1992, 18, 59, 31).
nonevent([pwd], lexical, nul, nul, home, monday, 20, january, 1992, 18, 56, 57).
nonevent([pwd], lexical, nul, nul, home, monday, 20, january, 1992, 18,

```
56, 57).
nonevent(['pwd]'], lexical, nul, nul, home, monday, 20, january, 1992,
    18, 56, 54).
nonevent([cd, ' ', persin], reference, persout, persin, persout, monday,
    20, january, 1992, 18, 51, 4).
nonevent([ls, ' ', l], reference, mail, l, mail, monday, 20, january,
    1992, 18, 49, 26).
nonevent([cd, ' ', busout], reference, home, busout, home, monday, 20,
    january, 1992, 18, 48, 40).
nonevent([cd, ' ', business], reference, home, business, home, monday,
    20, january, 1992, 18, 48, 37).
nonevent([cd, ' ', business], reference, business, business, business,
    monday, 20, january, 1992, 18, 37, 38).
nonevent([chmod, ' ', 700, ' ', complaintin], reference, business,
    complaintin, business, monday, 20, january, 1992, 18, 36, 42).
nonevent([ls, ' ', '-l'], read, mail, business, business, monday, 20,
    january, 1992, 18, 31, 59).
nonevent([cd, ' ', persaddr], execute, addresses, persaddr, persaddr,
    monday, 20, january, 1992, 18, 23, 46).
nonevent([cd, ' ', persaddr], lexical, nul, nul, persaddr, monday, 20,
    january, 1992, 18, 23, 45).
nonevent([ch], lexical, nul, nul, addresses, monday, 20, january, 1992,
    18, 23, 40).
nonevent([cd, ' ', addresses], reference, addresses, addresses,
    addresses, monday, 20, january, 1992, 18, 22, 4).
nonevent([cd, ' ', persaddr], execute, addresses, persaddr, addresses,
    monday, 20, january, 1992, 18, 20, 30).
nonevent([cd, ' ', (/), adresses], reference, persaddr, (/), persaddr,
    monday, 20, january, 1992, 18, 19, 33).
nonevent([cd, ' ', persaddr, (/), addresses], reference, persaddr,
    persaddr, persaddr, monday, 20, january, 1992, 18, 18, 10).
nonevent([cd, ' ', root, (/), home, (/), addresses], reference,
    persaddr, root, persaddr, monday, 20, january, 1992, 18, 17, 37).
nonevent([ls, ' ', '-l'], reference, persaddr, '-l', persaddr, monday,
    20, january, 1992, 18, 17, 8).
nonevent([ls, ' ', '-l'], lexical, nul, nul, persaddr, monday, 20,
    january, 1992, 18, 17, 7).
nonevent([ls], lexical, nul, nul, persaddr, monday, 20, january, 1992,
    18, 17, 1).
nonevent([cd, ' ', persaddr], reference, persaddr, persaddr, persaddr,
    monday, 20, january, 1992, 18, 16, 34).
nonevent([pwd], lexical, nul, nul, persaddr, monday, 20, january, 1992,
    18, 16, 17).
nonevent([pwd], lexical, nul, nul, persaddr, monday, 20, january, 1992,
    18, 16, 17).
nonevent([ls], lexical, nul, nul, persaddr, monday, 20, january, 1992,
    18, 15, 46).
nonevent([cd, ' ', addresses], reference, persaddr, addresses,
    addresses, monday, 20, january, 1992, 18, 14, 40).
nonevent([cd, ' ', addresses], execute, home, addresses, addresses,
    monday, 20, january, 1992, 18, 14, 40).
nonevent([cd, ' ', addresses], lexical, nul, nul, addresses, monday, 20,
    january, 1992, 18, 14, 40).
nonevent([ch], lexical, nul, nul, persaddr, monday, 20, january, 1992,
    18,14, 31).
nonevent([chmod, ' ', 400, ' ', persaddr], reference, persaddr,
    persaddr, persaddr, monday, 20, january, 1992, 18, 13, 17).
nonevent([ls, ' ', '-l'], read, addresses, persaddr, persaddr, monday,
    20, january, 1992, 18, 12, 19).
nonevent([chmod, ' ', persaddr], syntax, nul, nul, addresses, monday,
    20,  january, 1992, 18, 11, 14).
nonevent([chmod, ' ', 100], syntax, nul, nul, addresses, monday, 20,
    january, 1992, 18, 9, 10).
nonevent([chmod, (/), home, (/), addresses, (/), persaddr], syntax, nul,
    nul, addresses, monday, 20, january, 1992, 18, 7, 23).
```

```
nonevent([chmod, ' ', home, (/), addresses, (/)], syntax, nul, nul,
    addresses, monday, 20, january, 1992, 18, 7, 0).
nonevent([chmod, ' ', 100, ' ', root, (/), home, (/), addresses],
    reference, addresses, root, addresses, monday, 20, january, 1992,
    18, 6, 23).
nonevent([chmod, ' ', 100, ' '], range, nul, nul, addresses, monday, 20,
    january, 1992, 18, 6, 1).
nonevent([chmod, ' ', ' ', 100], syntax, nul, nul, addresses, monday,
    20, january, 1992, 18, 5, 38).
nonevent([cd, ' ', persaddr], execute, addresses, persaddr, addresses,
    monday, 20, january, 1992, 18, 4, 44).


/* owner/1 */

owner(S2).


/* totaltime/3 */

totaltime(1, 17, 4).


/* file/3 */

file(smiths, persaddr, 700).
file(joneses, persaddr, 700).
file(custaddr, busaddr, 700).
file(suppraddr, busaddr, 700).
file(thanksin, busmailin, 0).
file(complaintin, busmailin, 0).
file(compreply, busmailout, 0).
file(inquireout, busmailout, 0).
file(accountsaddr, busaddr, 700).
file(frommary, persmail, 0).
file(fromjim, persmail, 0).
file(tojim, persmail, 0).


/* dir/4 */

dir(root, 0, nul, [home]).
dir(persaddr, 700, personal, [joneses, smiths]).
dir(business, 700, home, [busmail, busaddr]).
dir(home, 700, root, [business, personal]).
dir(busmailin, 700, busmail, [thanksin, complaintin]).
dir(busmailout, 700, busmail, [compreply, inquireout]).
dir(busmail, 300, business, [busmailout, busmailin]).
dir(busaddr, 700, business, [accountsaddr, suppraddr, custaddr]).
dir(persmail, 0, personal, [frommary, fromjim, tojim]).
dir(personal, 0, home, [persaddr, persmail]).


/* event/9 */

event([ls, ' ', '-l'], home, tuesday, 21, january, 1992, 18, 50, 20).
event([chmod, ' ', 0, ' ', personal], home, tuesday, 21, january, 1992,
    18,  50, 16).
event([cd], home, tuesday, 21, january, 1992, 18, 50, 0).
event([cd, ' ', personal], personal, tuesday, 21, january, 1992, 18, 49,
    59).
event([ls, ' ', '-l'], persaddr, tuesday, 21, january, 1992, 18, 49,
    39).
event([cd, ' ', persaddr], persaddr, tuesday, 21, january, 1992, 18, 49,
    34).
```

```
event([chmod, ' ', 0, ' ', persmail], personal, tuesday, 21, january,
    1992, 18, 49, 27).
event([cd, ' ', personal], personal, tuesday, 21, january, 1992, 18, 48,
    57).
event([ls, ' ', '-l'], persmail, tuesday, 21, january, 1992, 18, 48,
    45).
event([chmod, ' ', 0, ' ', tojim], persmail, tuesday, 21, january, 1992,
    18, 48, 40).
event([chmod, ' ', 0, ' ', fromjim], persmail, tuesday, 21, january,
    1992, 18, 48, 32).
event([chmod, ' ', 0, ' ', frommary], persmail, tuesday, 21, january,
    1992, 18, 48, 21).
event([ls, ' ', '-l'], persmail, tuesday, 21, january, 1992, 18, 47,
    59).
event([cd, ' ', persmail], persmail, tuesday, 21, january, 1992, 18, 47,
    56).
event([cd, ' ', personal], personal, tuesday, 21, january, 1992, 18, 47,
    45).
event([cd], home, tuesday, 21, january, 1992, 18, 47, 38).
event([ls, ' ', '-l'], busaddr, tuesday, 21, january, 1992, 18, 47, 30).
event([mkfile, ' ', accountsaddr], busaddr, tuesday, 21, january, 1992,
    18, 47, 26).
event([ls, ' ', '-l'], busaddr, tuesday, 21, january, 1992, 18, 46, 33).
event([cd, ' ', busaddr], busaddr, tuesday, 21, january, 1992, 18, 46,
    30).
event([ls, ' ', '-l'], business, tuesday, 21, january, 1992, 18, 46,
    11).
event([chmod, ' ', 300, ' ', busmail], business, tuesday, 21, january,
    1992, 18, 46, 7).
event([cd, ' ', business], business, tuesday, 21, january, 1992, 18, 44,
    23).
event([cd, ' ', busmail], busmail, tuesday, 21, january, 1992, 18, 44,
    11).
event([chmod, ' ', 0, ' ', inquireout], busmailout, tuesday, 21,
    january, 1992, 18, 43, 58).
event([chmod, ' ', 0, ' ', compreply], busmailout, tuesday, 21, january,
    1992, 18, 43, 45).
event([mkfile, ' ', compreply], busmailout, tuesday, 21, january, 1992,
    18, 43, 29).
event([ls, ' ', '-l'], busmailout, tuesday, 21, january, 1992, 18, 43,
    5).
event([cd, ' ', busmailout], busmailout, tuesday, 21, january, 1992, 18,
    43, 2).
event([cd, ' ', busmail], busmail, tuesday, 21, january, 1992, 18, 42,
    46).
event([ls, ' ', '-l'], busmailin, tuesday, 21, january, 1992, 18, 42,
    39).
event([chmod, ' ', 0, ' ', complaintin], busmailin, tuesday, 21,
    january, 1992, 18, 42, 26).
event([chmod, ' ', 0, ' ', thanksin], busmailin, tuesday, 21, january,
    1992, 18, 42, 16).
event([ls, ' ', '-l'], busmailin, tuesday, 21, january, 1992, 18, 41,
    41).
event([cd, ' ', busmailin], busmailin, tuesday, 21, january, 1992, 18,
    41, 35).
event([cd, ' ', busmail], busmail, tuesday, 21, january, 1992, 18, 40,
    59).
event([mkfile, ' ', thanksin], busmailin, tuesday, 21, january, 1992,
    18, 40, 46).
event([ls], busmailin, tuesday, 21, january, 1992, 18, 40, 28).
event([cd, ' ', busmailin], busmailin, tuesday, 21, january, 1992, 18,
    40, 22).
event([ls], busmail, tuesday, 21, january, 1992, 18, 40, 15).
event([cd, ' ', busmail], busmail, tuesday, 21, january, 1992, 18, 40,
    11).
```

```
event([ls], business, tuesday, 21, january, 1992, 18, 40, 5).
event([cd, ' ', business], business, tuesday, 21, january, 1992, 18, 40,
    3).
event([ls], home, tuesday, 21, january, 1992, 18, 39, 33).
event([cd], home, tuesday, 21, january, 1992, 18, 38, 57).
event([ls], addresses, tuesday, 21, january, 1992, 18, 38, 52).
event([cd, ' ', addresses], addresses, tuesday, 21, january, 1992, 18,
    38, 38).
event([ls], home, tuesday, 21, january, 1992, 18, 38, 25).
event([rmdir, ' ', scrap], home, tuesday, 21, january, 1992, 18, 38,
    22).
event([cd], home, tuesday, 21, january, 1992, 18, 38, 17).
event([ls], scrap, tuesday, 21, january, 1992, 18, 38, 14).
event([rm, ' ', fromeric], scrap, tuesday, 21, january, 1992, 18, 38,
    12).
event([mv, ' ', suppraddr, ' ', home, (/), business, (/), busaddr],
    scrap, tuesday, 21, january, 1992, 18, 37, 54).
event([mv, ' ', custaddr, ' ', home, (/), business, (/), busaddr],
    scrap, tuesday, 21, january, 1992, 18, 36, 41).
event([ls], scrap, tuesday, 21, january, 1992, 18, 36, 6).
event([cd, ' ', scrap], scrap, tuesday, 21, january, 1992, 18, 36, 4).
event([rmdir, ' ', mailin], home, tuesday, 21, january, 1992, 18, 35,
    54).
event([ls], home, tuesday, 21, january, 1992, 18, 35, 31).
event([mv, ' ', inquireout, ' ', business, (/), busmail, (/)
    busmailout], home, tuesday, 21, january, 1992, 18, 35, 2).
event([mv, ' ', complaintin, ' ', business, (/), busmail, (/),
    busmailin], home, tuesday, 21, january, 1992, 18, 33, 37).
event([ls], home, tuesday, 21, january, 1992, 18, 33, 8).
event([cd], home, tuesday, 21, january, 1992, 18, 33, 3).
event([cd, ' ', busmailin], busmailin, tuesday, 21, january, 1992, 18,
    31, 49).
event([mkdir, ' ', busmailout], busmail, tuesday, 21, january, 1992, 18,
    31, 22).
event([mkdir, ' ', busmailin], busmail, tuesday, 21, january, 1992, 18,
    31, 3).
event([cd, ' ', busmail], busmail, tuesday, 21, january, 1992, 18, 30,
    51).
event([mkdir, ' ', busmail], business, tuesday, 21, january, 1992, 18,
    30, 46).
event([ls], business, tuesday, 21, january, 1992, 18, 30, 38).
event([mkdir, ' ', busaddr], business, tuesday, 21, january, 1992, 18,
    30, 33).
event([cd, ' ', business], business, tuesday, 21, january, 1992, 18, 30,
    14).
event([cd], home, tuesday, 21, january, 1992, 18, 30, 10).
event([rmdir, ' ', busaddr], addresses, tuesday, 21, january, 1992, 18,
    30, 8).
event([cd, ' ', addresses], addresses, tuesday, 21, january, 1992, 18,
    29, 57).
event([ls], busaddr, tuesday, 21, january, 1992, 18, 29, 39).
event([cd, ' ', busaddr], busaddr, tuesday, 21, january, 1992, 18, 29,
    35).
event([cd, ' ', addresses], addresses, tuesday, 21, january, 1992, 18,
    29, 27).
event([cd], home, tuesday, 21, january, 1992, 18, 29, 20).
event([cd, ' ', business], business, tuesday, 21, january, 1992, 18, 28,
    56).
event([mkdir, ' ', business], home, tuesday, 21, january, 1992, 18, 28,
    37).
event([rm, ' ', temp], home, tuesday, 21, january, 1992, 18, 28, 11).
event([ls, ' ', '-l'], home, tuesday, 21, january, 1992, 18, 27, 45).
event([rmdir, ' ', mail], home, tuesday, 21, january, 1992, 18, 27, 34).
event([cd], home, tuesday, 21, january, 1992, 18, 27, 29).
event([rmdir, ' ', business], mail, tuesday, 21, january, 1992, 18, 27,
```

24).
event([cd, ' ', mail], mail, tuesday, 21, january, 1992, 18, 27, 16).
event([rmdir, ' ', busin], business, tuesday, 21, january, 1992, 18, 27, 12).
event([rmdir, ' ', busout], business, tuesday, 21, january, 1992, 18, 27, 5).
event([cd, ' ', business], business, tuesday, 21, january, 1992, 18, 26, 54).
event([cd, ' ', mail], mail, tuesday, 21, january, 1992, 18, 26, 50).
event([mv, ' ', mail, (/), business, (/), busin, (/), complaintin, ' ', complaintin], home, tuesday, 21, january, 1992, 18, 25, 1).
event([mv, ' ', mail, (/), business, (/), busout, (/), inquireout, ' ', inquireout], home, tuesday, 21, january, 1992, 18, 24, 33).
event([cd], home, tuesday, 21, january, 1992, 18, 23, 54).
event([ls], busout, tuesday, 21, january, 1992, 18, 22, 13).
event([cd, ' ', busout], busout, tuesday, 21, january, 1992, 18, 22, 11).
event([cd, ' ', business], business, tuesday, 21, january, 1992, 18, 22, 2).
event([cd, ' ', mail], mail, tuesday, 21, january, 1992, 18, 21, 57).
event([cd], home, tuesday, 21, january, 1992, 18, 21, 46).
event([ls], scrap, tuesday, 21, january, 1992, 18, 21, 9).
event([mv, ' ', joneses, ' ', home, (/), personal, (/), persaddr, (/), joneses], scrap, tuesday, 21, january, 1992, 18, 21, 7).
event([mv, ' ', smiths, ' ', home, (/), personal, (/), persaddr, (/), smiths], scrap, tuesday, 21, january, 1992, 18, 20, 45).
event([mv, ' ', frommary, ' ', home, (/), personal, (/), persmail, (/), frommary], scrap, tuesday, 21, january, 1992, 18, 19, 56).
event([mv, ' ', fromjim, ' ', home, (/), personal, (/), persmail, (/), fromjim], scrap, tuesday, 21, january, 1992, 18, 19, 16).
event([ls], scrap, tuesday, 21, january, 1992, 18, 18, 23).
event([cd, ' ', scrap], scrap, tuesday, 21, january, 1992, 18, 18, 20).
event([mv, ' ', tojim, ' ', personal, (/), persmail, (/), tojim], home, tuesday, 21, january, 1992, 18, 17, 55).
event([ls], home, tuesday, 21, january, 1992, 18, 16, 1).
event([mv, ' ', personal, (/), home, (/), scrap, (/), tojim, ' ', tojim], home, tuesday, 21, january, 1992, 18, 15, 59).
event([cd], home, tuesday, 21, january, 1992, 18, 13, 33).
event([cd, ' ', persmail], persmail, tuesday, 21, january, 1992, 18, 12, 36).
event([ls, ' ', '-l'], personal, tuesday, 21, january, 1992, 18, 11, 59).
event([mkdir, ' ', persaddr], personal, tuesday, 21, january, 1992, 18, 11, 51).
event([cd, ' ', personal], personal, tuesday, 21, january, 1992, 18, 11, 34).
event([cd], home, tuesday, 21, january, 1992, 18, 11, 28).
event([cd, ' ', mail], mail, tuesday, 21, january, 1992, 18, 11, 24).
event([cd], home, tuesday, 21, january, 1992, 18, 11, 20).
event([rmdir, ' ', persaddr], addresses, tuesday, 21, january, 1992, 18, 11, 17).
event([cd, ' ', addresses], addresses, tuesday, 21, january, 1992, 18, 11, 3).
event([cd], home, tuesday, 21, january, 1992, 18, 10, 48).
event([mkdir, ' ', persmail], personal, tuesday, 21, january, 1992, 18, 10, 20).
event([rmdir, ' ', permail], personal, tuesday, 21, january, 1992, 18, 10, 13).
event([mv, ' ', permail, ' ', persmail], personal, tuesday, 21, january, 1992, 18, 9, 47).
event([mkdir, ' ', permail], personal, tuesday, 21, january, 1992, 18, 9, 24).
event([cd, ' ', personal], personal, tuesday, 21, january, 1992, 18, 9, 16).
event([mkdir, ' ', personal], home, tuesday, 21, january, 1992, 18, 9,

```
5).
event([cd], home, tuesday, 21, january, 1992, 18, 8, 36).
event([rmdir, ' ', personal], mail, tuesday, 21, january, 1992, 18, 8,
    30).
event([cd, ' ', mail], mail, tuesday, 21, january, 1992, 18, 8, 24).
event([rmdir, ' ', persin], personal, tuesday, 21, january, 1992, 18, 8,
    17).
event([rmdir, ' ', persout], personal, tuesday, 21, january, 1992, 18,
    8, 9).
event([cd, ' ', personal], personal, tuesday, 21, january, 1992, 18, 7,
    56).
event([ls], persout, tuesday, 21, january, 1992, 18, 7, 48).
event([cd, ' ', persout], persout, tuesday, 21, january, 1992, 18, 7,
    45).
event([cd, ' ', personal], personal, tuesday, 21, january, 1992, 18, 7,
    35).
event([cd, ' ', mail], mail, tuesday, 21, january, 1992, 18, 7, 29).
event([mv, ' ', mail, (/), personal, (/), persin, (/), fromjim, ' ',
    scrap, (/), fromjim], home, tuesday, 21, january, 1992, 18, 7, 20).
event([mv, ' ', mail, (/), personal, (/), persin, (/), frommary, ' ',
    scrap, (/), frommary], home, tuesday, 21, january, 1992, 18, 6, 57).
event([mv, ' ', mail, (/), personal, (/), persout, (/), tojim, ' ',
    scrap, (/), tojim], home, tuesday, 21, january, 1992, 18, 6, 9).
event([cd], home, tuesday, 21, january, 1992, 18, 5, 15).
event([ls, ' ', '-l'], scrap, tuesday, 21, january, 1992, 18, 5, 9).
event([cd, ' ', scrap], scrap, tuesday, 21, january, 1992, 18, 5, 4).
event([cd], home, tuesday, 21, january, 1992, 18, 4, 55).
event([mv, ' ', addresses, (/), busaddr, (/), suppaddr, ' ', scrap, (/),
    suppraddr], home, tuesday, 21, january, 1992, 18, 4, 52).
event([mv, ' ', addresses, (/), busaddr, (/), custaddr, ' ', scrap, (/),
    custaddr], home, tuesday, 21, january, 1992, 18, 4, 3).
event([mv, ' ', addresses, (/), persaddr, (/), smiths, ' ', scrap, (/),
    smiths], home, tuesday, 21, january, 1992, 18, 2, 56).
event([mv, ' ', mail, (/), fromeric, ' ', scrap, (/), fromeric], home,
    tuesday, 21, january, 1992, 18, 1, 54).
event([mv, ' ', mail, (/), joneses, ' ', scrap, (/), joneses], home,
    tuesday, 21, january, 1992, 18, 1, 28).
event([mv, ' ', mail, ' ', scrap], home, tuesday, 21, january, 1992, 18,
    1, 2).
event([cd], home, tuesday, 21, january, 1992, 18, 0, 50).
event([ls, ' ', '-l'], mail, tuesday, 21, january, 1992, 18, 0, 8).
event([cd, ' ', mail], mail, tuesday, 21, january, 1992, 18, 0, 3).
event([mv, ' ', addresses, (/), persaddr, (/), joneses, ' ', mail, (/),
    joneses], home, tuesday, 21, january, 1992, 17, 59, 57).
event([mv, ' ', addresses, ' ', persaddr, ' ', joneses, ' ', mail],
    home, tuesday, 21, january, 1992, 17, 58, 49).
event([cd], home, tuesday, 21, january, 1992, 17, 57, 31).
event([cd, ' ', scrap], scrap, tuesday, 21, january, 1992, 17, 57, 24).
event([ls, ' ', '-l'], home, tuesday, 21, january, 1992, 17, 57, 11).
event([mkdir, ' ', scrap], home, tuesday, 21, january, 1992, 17, 57, 6).
event([cd], home, tuesday, 21, january, 1992, 17, 55, 32).
event([ls, ' ', '-l'], busin, tuesday, 21, january, 1992, 17, 54, 12).
event([chmod, ' ', 700, ' ', complaintin], busin, tuesday, 21, january,
    1992, 17, 54, 9).
event([ls, ' ', '-l'], busin, tuesday, 21, january, 1992, 17, 53, 47).
event([cd, ' ', busin], busin, tuesday, 21, january, 1992, 17, 53, 42).
event([cd, ' ', business], business, tuesday, 21, january, 1992, 17, 53,
    38).
event([ls], busout, tuesday, 21, january, 1992, 17, 53, 17).
event([rm, ' ', addout], busout, tuesday, 21, january, 1992, 17, 53, 2).
event([ls], busout, tuesday, 21, january, 1992, 17, 52, 31).
event([mv, ' ', enquirout, ' ', inquireout], busout, tuesday, 21,
    january, 1992, 17, 52, 29).
event([ls, ' ', '-l'], busout, tuesday, 21, january, 1992, 17, 50, 44).
event([cd, ' ', busout], busout, tuesday, 21, january, 1992, 17, 50,
```

```
    41).
event([ls, ' ', '-l'], business, tuesday, 21, january, 1992, 17, 50,
    22).
event([cd, ' ', business], business, tuesday, 21, january, 1992, 17, 50,
    18).
event([cd, ' ', mail], mail, tuesday, 21, january, 1992, 17, 50, 12).
event([cd, ' ', personal], personal, tuesday, 21, january, 1992, 17, 50,
    9).
event([ls, ' ', '-l'], persin, tuesday, 21, january, 1992, 17, 49, 48).
event([cd, ' ', persin], persin, tuesday, 21, january, 1992, 17, 49,
    43).
event([cd, ' ', personal], personal, tuesday, 21, january, 1992, 17, 49,
    31).
event([chmod, ' ', 700, ' ', tojim], persout, tuesday, 21, january,
    1992, 17, 48, 1).
event([ls, ' ', '-l'], persout, tuesday, 21, january, 1992, 17, 47, 21).
event([cd, ' ', persout], persout, tuesday, 21, january, 1992, 17, 47,
    17).
event([ls, ' ', '-l'], personal, tuesday, 21, january, 1992, 17, 46,
    57).
event([cd, ' ', personal], personal, tuesday, 21, january, 1992, 17, 46,
    53).
event([chmod, ' ', 700, ' ', personal], mail, tuesday, 21, january,
    1992, 17, 46, 47).
event([chmod, ' ', 700, ' ', business], mail, tuesday, 21, january,
    1992, 17, 46, 39).
event([ls, ' ', '-l'], mail, tuesday, 21, january, 1992, 17, 45, 45).
event([ls], mail, tuesday, 21, january, 1992, 17, 45, 38).
event([cd, ' ', mail], mail, tuesday, 21, january, 1992, 17, 45, 34).
event([cd], home, tuesday, 21, january, 1992, 17, 45, 31).
event([ls, ' ', '-l'], busaddr, tuesday, 21, january, 1992, 17, 45, 3).
event([cd, ' ', busaddr], busaddr, tuesday, 21, january, 1992, 17, 44,
    58).
event([ls], addresses, tuesday, 21, january, 1992, 17, 44, 51).
event([cd, ' ', addresses], addresses, tuesday, 21, january, 1992, 17,
    44, 47).
event([ls, ' ', '-l'], home, tuesday, 21, january, 1992, 17, 44, 32).
event([cd], home, tuesday, 21, january, 1992, 17, 41, 59).
event([ls, ' ', '-l'], persaddr, tuesday, 21, january, 1992, 17, 41,
    26).
event([rm, ' ', bloggs], persaddr, tuesday, 21, january, 1992, 17, 41,
    22).
event([rm, ' ', johnsons], persaddr, tuesday, 21, january, 1992, 17, 41,
    14).
event([ls, ' ', '-l'], persaddr, tuesday, 21, january, 1992, 17, 40,
    21).
event([cd, ' ', persaddr], persaddr, tuesday, 21, january, 1992, 17, 40,
    15).
event([ls, ' ', '-l'], addresses, tuesday, 21, january, 1992, 17, 40,
    2).
event([chmod, ' ', 700, ' ', persaddr], addresses, tuesday, 21, january,
    1992, 17, 39, 57).
event([ls, ' ', '-l'], addresses, tuesday, 21, january, 1992, 17, 39,
    32).
event([chmod, ' ', 400, ' ', persaddr], addresses, tuesday, 21, january,
    1992, 17, 39, 28).
event([ls, ' ', '-l'], addresses, tuesday, 21, january, 1992, 17, 37,
    42).
event([ls], addresses, tuesday, 21, january, 1992, 17, 37, 23).
event([cd, ' ', addresses], addresses, tuesday, 21, january, 1992, 17,
    37, 21).
event([ls], home, tuesday, 21, january, 1992, 17, 37, 3).
event([cd], home, tuesday, 21, january, 1992, 17, 37, 1).
event([ls], mailin, tuesday, 21, january, 1992, 17, 36, 44).
event([ls, ' ', '-l'], mailin, tuesday, 21, january, 1992, 17, 35, 43).
```

```
event([cd, ' ', mailin], mailin, tuesday, 21, january, 1992, 17, 35,
    33).
event([ls, ' ', '-l'], home, tuesday, 21, january, 1992, 17, 34, 3).
event([chmod, ' ', 200, ' ', home, (/), addresses, (/), persaddr], mail,
    wednesday, 8, jan, 1992, 10, 26, 27).
event([chmod, ' ', 100, ' ', business], mail, wednesday, 8, jan, 1992,
    10, 26, 27).
event([mkfile, ' ', business, (/), busout, (/), addout], mail,
    wednesday, 8, jan, 1992, 10, 26, 27).
event([mkfile, ' ', fromeric], mail, wednesday, 8, jan, 1992, 10, 26,
    27).
event([cd, ' ', mail], mail, wednesday, 8, jan, 1992, 10, 26, 27).
event([mkdir, ' ', mailin], home, wednesday, 8, jan, 1992, 10, 26, 27).
event([cd], home, wednesday, 8, jan, 1992, 10, 26, 26).
event([mkfile, ' ', johnsons], persaddr, wednesday, 8, jan, 1992, 10,
    26, 26).
event([mkfile, ' ', bloggs], persaddr, wednesday, 8, jan, 1992, 10, 26,
    26).
event([mkfile, ' ', joneses], persaddr, wednesday, 8, jan, 1992, 10, 26,
    26).
event([mkfile, ' ', smiths], persaddr, wednesday, 8, jan, 1992, 10, 26,
    26).
event([cd, ' ', addresses, (/), persaddr], persaddr, wednesday, 2, may,
    1991, 10, 26, 26).
event([mkfile, ' ', custaddr], busaddr, wednesday, 8, jan, 1992, 10, 26,
    25).
event([mkfile, ' ', suppaddr], busaddr, wednesday, 8, jan, 1992, 10, 26,
    25).
event([cd, ' ', busaddr], busaddr, wednesday, 8, jan, 1992, 10, 26, 25).
event([mkdir, ' ', persaddr], addresses, wednesday, 8, jan, 1992, 10,
    26, 25).
event([mkdir, ' ', busaddr], addresses, wednesday, 8, jan, 1992, 10, 26,
    25).
event([cd, ' ', business, (/), mail, (/), home, (/), addresses],
    addresses, wednesday, 8, jan, 1992, 10, 26, 25).
event([mkfile, ' ', enquirout], busout, wednesday, 8, jan, 1992, 10, 26,
    25).
event([cd, ' ', business, (/), busout], busout, wednesday, 8, jan, 1992,
    10, 26, 24).
event([chmod, ' ', 400, ' ', complaintin], busin, wednesday, 2, may,
    1991, 10, 26, 24).
event([mkfile, ' ', complaintin], busin, wednesday, 8, jan, 1992, 10,
    26, 24).
event([cd, ' ', busin], busin, wednesday, 8, jan, 1992, 10, 26, 24).
event([mkdir, ' ', busout], business, wednesday, 8, jan, 1992, 10, 26,
    24).
event([mkdir, ' ', busin], business, wednesday, 8, jan, 1992, 10, 26,
    24).
event([cd, ' ', business], business, wednesday, 8, jan, 1992, 10, 26,
    23).
event([chmod, ' ', 0, ' ', personal], mail, wednesday, 8, jan, 1992, 10,
    26, 23).
event([cd, ' ', personal, (/), mail], mail, wednesday, 8, jan, 1992, 10,
    26, 23).
event([chmod, ' ', 0, ' ', tojim], persout, wednesday, 8, jan, 1992, 10,
    26, 23).
event([mkfile, ' ', tojim], persout, wednesday, 8, jan, 1992, 10, 26,
    23).
event([cd, ' ', personal, (/), persout], persout, wednesday, 2, may,
    1991, 10, 26, 23).
event([chmod, ' ', 0, ' ', frommary], persin, wednesday, 8, jan, 1992,
    10, 26, 23).
event([mkfile, ' ', frommary], persin, wednesday, 8, jan, 1992, 10, 26,
    22).
event([chmod, ' ', 0, ' ', fromjim], persin, wednesday, 8, jan, 1992,
```

```
event([mkfile, ' ', fromjim], persin, wednesday, 8, jan, 1992, 10, 26,
    22).
event([cd, ' ', persin], persin, wednesday, 8, jan, 1992, 10, 26, 22).
event([mkdir, ' ', persout], personal, wednesday, 8, jan, 1992, 10, 26,
    22).
event([mkdir, ' ', persin], personal, wednesday, 8, jan, 1992, 10, 26,
    21).
event([cd, ' ', personal], personal, wednesday, 8, jan, 1992, 10, 26,
    21).
event([mkdir, ' ', business], mail, wednesday, 8, jan, 1992, 10, 26,
    21).
event([mkdir, ' ', personal], mail, wednesday, 8, jan, 1992, 10, 26,
    21).
event([cd, ' ', mail], mail, wednesday, 8, jan, 1992, 10, 26, 21).
event([mkdir, ' ', addresses], home, wednesday, 8, jan, 1992, 10, 26,
    21).
event([mkdir, ' ', mail], home, wednesday, 8, jan, 1992, 10, 26, 21).


/* nonevent/12 */

nonevent([cd, ' ', permail], reference, personal, permail, personal,
    tuesday, 21, january, 1992, 18, 47, 51).
nonevent([cd, ' ', busmail], reference, busmail, busmail, busmail,
    tuesday, 21, january, 1992, 18, 41, 17).
nonevent([rmdir, ' ', addresses], lexical, nul, nul, home, tuesday, 21,
    january, 1992, 18, 39, 29).
nonevent([rd], lexical, nul, nul, home, tuesday, 21, january, 1992, 18,
    39, 7).
nonevent([rmdir, ' ', persaddr], reference, addresses, persaddr,
    addresses, tuesday, 21, january, 1992, 18, 38, 50).
nonevent([mv, ' ', custaddr, ' ', home, (/), business, (/), busaddr],
    reference, scrap, custaddr, scrap, tuesday, 21, january, 1992, 18,
    37, 16).
nonevent([mv, ' ', busmail, (/), business, (/), home, (/), scrap, (/),
    complaintin], reference, scrap, complaintin, busmailin, tuesday, 21,
    january, 1992, 18, 32, 53).
nonevent([mv, ' ', busmail, (/), business, (/), home, (/), complaintin],
    reference, home, complaintin, busmailin, tuesday, 21, january, 1992,
    18, 32, 25).
nonevent([mkdir, ' ', busaddr], duplication, business, busaddr,
    business, tuesday, 21, january, 1992, 18, 29, 4).
nonevent([mv, ' ', inquireout, ' ', scrap, (/), inquireout], reference,
    busout, scrap, busout, tuesday, 21, january, 1992, 18, 23, 46).
nonevent([mv, ' ', inquireout, ' ', home, (/), inquireout], reference,
    busout, home, busout, tuesday, 21, january, 1992, 18, 23, 26).
nonevent([mv, ' ', to, ' ', jim, ' ', personal, (/), persmail, (/),
    tojim], reference, home, to, home, tuesday, 21, january, 1992, 18,
    17, 32).
nonevent([mv, ' ', tojim, ' ', personnal, (/), persmail, (/), tojim],
    reference, home, personnal, home, tuesday, 21, january, 1992, 18,
    17, 8).
nonevent([mv, ' ', tojim, ' ', personal, (/), permail, (/), tojim],
    reference, personal, permail, home, tuesday, 21, january, 1992,18,
    16, 40).
nonevent([mv, ' ', home, (/), scrap, (/), tojim], reference, persmail,
    home, persmail, tuesday, 21, january, 1992, 18, 13, 27).
nonevent([mv, ' ', scrap, (/), tojim], reference, persmail, scrap,
    persmail, tuesday, 21, january, 1992, 18, 12, 57).
nonevent([mkdir, ' ', persaddr], duplication, personal, persaddr,
    personal, tuesday, 21, january, 1992, 18, 10, 38).
nonevent([mv, ' ', mail, ' ', scrap], reference, mail, mail, mail,
    tuesday, 21, january, 1992, 18, 0, 46).
nonevent([mv, ' ', address, ' ', persaddr, ' ', joneses, ' ', mail],
```

```
    21).
nonevent([cd, ' ', busout], reference, busout, busout, busout, tuesday,
    21, january, 1992, 17, 53, 30).
nonevent([cd, ' ', mail], reference, persin, mail, persin, tuesday, 21,
    january, 1992, 17, 50, 4).
nonevent([cd, ' ', person], reference, persout, person, persout,
    tuesday, 21, january, 1992, 17, 49, 23).
nonevent([ls, ' ', '-l'], lexical, nul, nul, persout, tuesday, 21,
    january, 1992, 17, 49, 12).
nonevent([cdls], lexical, nul, nul, persout, tuesday, 21, january, 1992,
    17, 49, 8).
nonevent([cd, ' ', person], reference, persout, person, persout,
    tuesday, 21, january, 1992, 17, 48, 31).
nonevent([cd, ' ', persin], reference, persout, persin, persout,
    tuesday, 21, january, 1992, 17, 48, 9).
nonevent([mkdir, ' ', personal], duplication, home, personal, home,
    tuesday, 21, january, 1992, 17, 44, 22).
nonevent([mv, ' ', address, ' ', personal], reference, home, address,
    home, tuesday, 21, january, 1992, 17, 43, 51).
nonevent([cd, ' ', persaddr], execute, addresses, persaddr, addresses,
    tuesday, 21, january, 1992, 17, 38, 19).
nonevent([cd, ' ', addresses], reference, mailin, addresses, mailin,
    tuesday, 21, january, 1992, 17, 36, 34).


/* owner/1 */

owner(S3).


/* totaltime/3 */

totaltime(0, 59, 34).


/* file/3 */

file(joneses, persaddr, 700).
file(smiths, persaddr, 700).
file(custaddr, busaddr, 700).
file(suppaddr, busaddr, 700).
file(accountsaddr, busaddr, 700).
file(complaintin, busmailin, 700).
file(thanksin, busmailin, 700).
file(inquireout, busmailout, 700).
file(compreply, busmailout, 700).
file(frommary, persmail, 0).
file(fromjim, persmail, 0).
file(tojim, persmail, 0).


/* dir/4 */

dir(root, 0, nul, [home]).
dir(personal, 700, home, [persaddr, persmail]).
dir(persaddr, 700, personal, [smiths, joneses]).
dir(business, 700, home, [busaddr, busmail]).
dir(busaddr, 700, business, [accountsaddr, suppaddr, custaddr]).
dir(busmailin, 700, busmail, [thanksin, complaintin]).
dir(busmailout, 700, busmail, [compreply, inquireout]).
dir(home, 700, root, [business, personal]).
dir(busmail, 300, business, [busmailout, busmailin]).
dir(persmail, 0, personal, [frommary, fromjim, tojim]).
```

```
/* event/9 */

event([chmod, ' ', 0, ' ', personal, (/), persmail], home, monday, 10,
    february, 1992, 20, 16, 48).
event([cd], home, monday, 10, february, 1992, 20, 16, 36).
event([chmod, ' ', 0, ' ', tojim], persmail, monday, 10, february, 1992,
    20, 16, 33).
event([chmod, ' ', 0, ' ', fromjim], persmail, monday, 10, february,
    1992, 20, 16, 26).
event([chmod, ' ', 0, ' ', frommary], persmail, monday, 10, february,
    1992, 20, 16, 18).
event([ls], persmail, monday, 10, february, 1992, 20, 16, 3).
event([cd, ' ', personal, (/), persmail], persmail, monday, 10,
    february, 1992, 20, 15, 58).
event([chmod, ' ', 700, ' ', personal, (/), persmail], home, monday, 10,
    february, 1992, 20, 15, 49).
event([chmod, ' ', 0, ' ', personal, (/), persmail], home, monday, 10,
    february, 1992, 20, 14, 58).
event([chmod, ' ', 300, ' ', business, (/), busmail], home, monday, 10,
    february, 1992, 20, 13, 34).
event([cd], home, monday, 10, february, 1992, 20, 12, 53).
event([ls], busaddr, monday, 10, february, 1992, 20, 12, 33).
event([cd, ' ', busaddr], busaddr, monday, 10, february, 1992, 20, 12,
    30).
event([ls], business, monday, 10, february, 1992, 20, 12, 22).
event([cd, ' ', business], business, monday, 10, february, 1992, 20, 12,
    21).
event([cd], home, monday, 10, february, 1992, 20, 12, 12).
event([ls], busmailout, monday, 10, february, 1992, 20, 12, 5).
event([cd, ' ', busmail, (/), busmailout], busmailout, monday, 10,
    february, 1992, 20, 12, 4).
event([ls], busmailin, monday, 10, february, 1992, 20, 11, 50).
event([cd, ' ', busmailin], busmailin, monday, 10, february, 1992, 20,
    11, 49).
event([ls], busmail, monday, 10, february, 1992, 20, 11, 42).
event([cd, ' ', busmail], busmail, monday, 10, february, 1992, 20, 11,
    39).
event([ls], business, monday, 10, february, 1992, 20, 11, 32).
event([cd, ' ', business], business, monday, 10, february, 1992, 20, 11,
    30).
event([cd], home, monday, 10, february, 1992, 20, 11, 24).
event([ls], persmail, monday, 10, february, 1992, 20, 11, 14).
event([cd, ' ', personal, (/), persmail], persmail, monday, 10,
    february, 1992, 20, 11, 13).
event([ls], persaddr, monday, 10, february, 1992, 20, 10, 45).
event([cd, ' ', persaddr], persaddr, monday, 10, february, 1992, 20, 10,
    44).
event([ls], personal, monday, 10, february, 1992, 20, 10, 33).
event([cd, ' ', personal], personal, monday, 10, february, 1992, 20, 10,
    31).
event([ls], home, monday, 10, february, 1992, 20, 10, 17).
event([cd], home, monday, 10, february, 1992, 20, 10, 13).
event([rmdir, ' ', addresses], home, monday, 10, february, 1992, 20, 10,
    7).
event([ls, ' ', '-l', ' ', addresses], home, monday, 10, february, 1992,
    20, 9, 59).
event([rmdir, ' ', mailin], home, monday, 10, february, 1992, 20, 9,
    46).
event([ls, ' ', '-l', ' ', mailin], home, monday, 10, february, 1992,
    20, 9, 26).
event([ls, ' ', mailin], home, monday, 10, february, 1992, 20, 9, 17).
event([ls], home, monday, 10, february, 1992, 20, 8, 47).
event([rmdir, ' ', mail], home, monday, 10, february, 1992, 20, 8, 44).
event([cd], home, monday, 10, february, 1992, 20, 8, 39).
```

event([rm, ' ', fromfile], mail, monday, 10, february, 1992, 20, 8, 26).
event([ls], mail, monday, 10, february, 1992, 20, 8, 4).
event([cd, ' ', mail], mail, monday, 10, february, 1992, 20, 8, 2).
event([mv, ' ', addout, ' ', business, (/), busmail, (/), busmailout, (/), compreply], home, monday, 10, february, 1992, 20, 7, 31).
event([mv, ' ', enquirout, ' ', business, (/), busmail, (/), busmailout, (/), inquireout], home, monday, 10, february, 1992, 20, 6, 42).
event([ls], home, monday, 10, february, 1992, 20, 5, 17).
event([cd], home, monday, 10, february, 1992, 20, 5, 14).
event([mkfile, ' ', busmailin, (/), thanksin], busmail, monday, 10, february, 1992, 20, 4, 45).
event([mv, ' ', business, (/), home, (/), complaintin, ' ', busmailin], busmail, monday, 10, february, 1992, 20, 4, 12).
event([mkdir, ' ', busmailout], busmail, monday, 10, february, 1992, 20, 3, 19).
event([mkdir, ' ', busmailin], busmail, monday, 10, february, 1992, 20, 3, 13).
event([cd, ' ', business, (/), busmail], busmail, monday, 10, february, 1992, 20, 3, 1).
event([mkfile, ' ', business, (/), busaddr, (/), accountsaddr], home, monday, 10, february, 1992, 20, 1, 53).
event([mv, ' ', suppaddr, ' ', business, (/), busaddr], home, monday, 10, february, 1992, 20, 0, 53).
event([ls], home, monday, 10, february, 1992, 19, 59, 57).
event([mv, ' ', custaddr, ' ', business, (/), busaddr], home, monday, 10, february, 1992, 19, 59, 46).
event([mkdir, ' ', business, (/), busaddr], home, monday, 10, february, 1992, 19, 58, 26).
event([rmdir, ' ', addresses, (/), busaddr], home, monday, 10, february, 1992, 19, 57, 59).
event([mv, ' ', addresses, (/), busaddr, (/), suppaddr, ' ', suppaddr], home, monday, 10, february, 1992, 19, 57, 35).
event([ls, ' ', addresses, (/), busaddr], home, monday, 10, february, 1992, 19, 57, 15).
event([mv, ' ', addresses, (/), busaddr, (/), custaddr, ' ', custaddr], home, monday, 10, february, 1992, 19, 55, 58).
event([mkdir, ' ', business, (/), busmail], home, monday, 10, february, 1992, 19, 55, 26).
event([mkdir, ' ', business], home, monday, 10, february, 1992, 19, 54, 50).
event([rmdir, ' ', mail, (/), business], home, monday, 10, february, 1992, 19, 54, 40).
event([rmdir, ' ', mail, (/), business, (/), busin], home, monday, 10, february, 1992, 19, 54, 32).
event([rmdir, ' ', mail, (/), business, (/), busout], home, monday, 10, february, 1992, 19, 54, 22).
event([mv, ' ', mail, (/), business, (/), busout, (/), enquirout, ' ', enquirout], home, monday, 10, february, 1992, 19, 53, 57).
event([ls, ' ', mail, (/), business, (/), busout], home, monday, 10, february, 1992, 19, 53, 3).
event([mv, ' ', mail, (/), business, (/), busout, (/), addout, ' ', addout], home, monday, 10, february, 1992, 19, 52, 26).
event([mv, ' ', mail, (/), business, (/), busin, (/), complaintin, ' ', complaintin], home, monday, 10, february, 1992, 19, 51, 52).
event([ls, ' ', mail, (/), business, (/), busin], home, monday, 10, february, 1992, 19, 51, 10).
event([ls, ' ', mail, (/), business], home, monday, 10, february, 1992, 19, 50, 58).
event([rm, ' ', temp], home, monday, 10, february, 1992, 19, 49, 0).
event([ls], home, monday, 10, february, 1992, 19, 48, 36).
event([cd], home, monday, 10, february, 1992, 19, 48, 33).
event([ls, ' ', personal, (/), persmail], home, monday, 10, february, 1992, 19, 47, 23).
event([mv, ' ', frommary, ' ', personal, (/), persmail], home, monday, 10, february, 1992, 19, 47, 4).

event([mv, ' ', fromjim, ' ', personal, (/), persmail, (/), fromjim], home, monday, 10, february, 1992, 19, 46, 31).
event([mv, ' ', tojim, ' ', personal, (/), persmail, (/), tojim], home, monday, 10, february, 1992, 19, 46, 12).
event([mv, ' ', smiths, ' ', personal, (/), persaddr, (/), smiths], home, monday, 10, february, 1992, 19, 45, 17).
event([mv, ' ', joneses, ' ', personal, (/), persaddr, (/), joneses], home, monday, 10, february, 1992, 19, 44, 56).
event([mkdir, ' ', personal, (/), persaddr], home, monday, 10, february, 1992, 19, 43, 52).
event([rmdir, ' ', addresses, (/), persaddr], home, monday, 10, february, 1992, 19, 43, 30).
event([rm, ' ', addresses, (/), persaddr, (/), johnsons], home, monday, 10, february, 1992, 19, 43, 11).
event([rm, ' ', addresses, (/), persaddr, (/), bloggs], home, monday, 10, february, 1992, 19, 42, 54).
event([mv, ' ', addresses, (/), persaddr, (/), smiths, ' ', smiths], home, monday, 10, february, 1992, 19, 42, 30).
event([mv, ' ', addresses, (/), persaddr, (/), joneses, ' ', joneses], home, monday, 10, february, 1992, 19, 42, 10).
event([mkdir, ' ', personal, (/), persmail], home, monday, 10, february, 1992, 19, 40, 31).
event([mkdir, ' ', personal], home, monday, 10, february, 1992, 19, 39, 48).
event([rmdir, ' ', mail, (/), personal], home, monday, 10, february, 1992, 19, 39, 16).
event([cd], home, monday, 10, february, 1992, 19, 39, 2).
event([rmdir, ' ', persout], personal, monday, 10, february, 1992, 19, 38, 44).
event([rmdir, ' ', persin], personal, monday, 10, february, 1992, 19, 38, 38).
event([mv, ' ', persout, (/), tojim, ' ', mail, (/), home, (/), tojim], personal, monday, 10, february, 1992, 19, 38, 11).
event([mv, ' ', persin, (/), fromjim, ' ', mail, (/), home, (/), fromjim], personal, monday, 10, february, 1992, 19, 37, 48).
event([mv, ' ', persin, (/), frommary, ' ', mail, (/), home, (/), frommary], personal, monday, 10, february, 1992, 19, 37, 21).
event([ls], personal, monday, 10, february, 1992, 19, 34, 19).
event([cd, ' ', mail, (/), personal], personal, monday, 10, february, 1992, 19, 33, 59).
event([cd], home, monday, 10, february, 1992, 19, 31, 24).
event([cd], home, monday, 10, february, 1992, 19, 29, 53).
event([ls, ' ', '-l', ' ', persout], personal, monday, 10, february, 1992, 19, 29, 42).
event([ls, ' ', '-l', ' ', persin], personal, monday, 10, february, 1992, 19, 29, 29).
event([ls, ' ', '-l', ' ', persin], personal, monday, 10, february, 1992, 19, 29, 5).
event([ls, ' ', '-l'], personal, monday, 10, february, 1992, 19, 28, 42).
event([cd, ' ', mail, (/), personal], personal, monday, 10, february, 1992, 19, 28, 35).
event([chmod, ' ', 700, ' ', mail, (/), business, (/), busin, (/), complaintin], home, monday, 10, february, 1992, 19, 28, 3).
event([ls, ' ', '-l', ' ', mail, (/), business, (/), busin], home, monday, 10, february, 1992, 19, 27, 15).
event([ls, ' ', '-l', ' ', mail, (/), business, (/), busout], home, monday, 10, february, 1992, 19, 26, 30).
event([ls, ' ', '-l', ' ', mail, (/), business], home, monday, 10, february, 1992, 19, 25, 54).
event([chmod, ' ', 700, ' ', mail, (/), personal], home, monday, 10, february, 1992, 19, 25, 42).
event([chmod, ' ', 700, ' ', mail, (/), business], home, monday, 10, february, 1992, 19, 25, 31).
event([ls, ' ', '-l', ' ', mail], home, monday, 10, february, 1992, 19,

```
event([ls, ' ', '-l', ' ', addresses, (/), busaddr], home, monday, 10,
    february, 1992, 19, 23, 43).
event([ls, ' ', addresses, (/), busaddr], home, monday, 10, february,
    1992, 19, 22, 33).
event([ls, ' ', addresses, (/), persaddr], home, monday, 10, february,
    1992, 19, 22, 0).
event([chmod, ' ', 700, ' ', addresses, (/), persaddr], home, monday,
    10, february, 1992, 19, 21, 50).
event([ls, ' ', addresses], home, monday, 10, february, 1992, 19, 20,
    8).
event([ls, ' ', mailin], home, monday, 10, february, 1992, 19, 19, 20).
event([ls], home, monday, 10, february, 1992, 19, 17, 42).
event([chmod, ' ', 200, ' ', home, (/), addresses, (/), persaddr], mail,
    wednesday, 8, jan, 1992, 10, 26, 27).
event([chmod, ' ', 100, ' ', business], mail, wednesday, 8, jan, 1992,
    10, 26, 27).
event([mkfile, ' ', business, (/), busout, (/), addout], mail,
    wednesday, 8, jan, 1992, 10, 26, 27).
event([mkfile, ' ', fromeric], mail, wednesday, 8, jan, 1992, 10, 26,
    27).
event([cd, ' ', mail], mail, wednesday, 8, jan, 1992, 10, 26, 27).
event([mkdir, ' ', mailin], home, wednesday, 8, jan, 1992, 10, 26, 27).
event([cd], home, wednesday, 8, jan, 1992, 10, 26, 26).
event([mkfile, ' ', johnsons], persaddr, wednesday, 8, jan, 1992, 10,
    26, 26).
event([mkfile, ' ', bloggs], persaddr, wednesday, 8, jan, 1992, 10, 26,
    26).
event([mkfile, ' ', joneses], persaddr, wednesday, 8, jan, 1992, 10, 26,
    26).
event([mkfile, ' ', smiths], persaddr, wednesday, 8, jan, 1992, 10, 26,
    26).
event([cd, ' ', addresses, (/), persaddr], persaddr, wednesday, 2, may,
    1991, 10, 26, 26).
event([mkfile, ' ', custaddr], busaddr, wednesday, 8, jan, 1992, 10, 26,
    25).
event([mkfile, ' ', suppaddr], busaddr, wednesday, 8, jan, 1992, 10, 26,
    25).
event([cd, ' ', busaddr], busaddr, wednesday, 8, jan, 1992, 10, 26, 25).
event([mkdir, ' ', persaddr], addresses, wednesday, 8, jan, 1992, 10,
    26, 25).
event([mkdir, ' ', busaddr], addresses, wednesday, 8, jan, 1992, 10, 26,
    25).
event([cd, ' ', business, (/), mail, (/), home, (/), addresses],
    addresses, wednesday, 8, jan, 1992, 10, 26, 25).
event([mkfile, ' ', enquirout], busout, wednesday, 8, jan, 1992, 10, 26,
    25).
event([cd, ' ', business, (/), busout], busout, wednesday, 8, jan, 1992,
    10, 26, 24).
event([chmod, ' ', 400, ' ', complaintin], busin, wednesday, 2, may,
    1991, 10, 26, 24).
event([mkfile, ' ', complaintin], busin, wednesday, 8, jan, 1992, 10,
    26, 24).
event([cd, ' ', busin], busin, wednesday, 8, jan, 1992, 10, 26, 24).
event([mkdir, ' ', busout], business, wednesday, 8, jan, 1992, 10, 26,
    24).
event([mkdir, ' ', busin], business, wednesday, 8, jan, 1992, 10, 26,
    24).
event([cd, ' ', business], business, wednesday, 8, jan, 1992, 10, 26,
    23).
event([chmod, ' ', 0, ' ', personal], mail, wednesday, 8, jan, 1992, 10,
    26, 23).
event([cd, ' ', personal, (/), mail], mail, wednesday, 8, jan, 1992, 10,
    26, 23).
event([chmod, ' ', 0, ' ', tojim], persout, wednesday, 8, jan, 1992, 10,
```

```
event([mkfile, ' ', tojim], persout, wednesday, 8, jan, 1992, 10, 26,
    23).
event([cd, ' ', personal, (/), persout], persout, wednesday, 2, may,
    1991, 10, 26, 23).
event([chmod, ' ', 0, ' ', frommary], persin, wednesday, 8, jan, 1992,
    10, 26, 23).
event([mkfile, ' ', frommary], persin, wednesday, 8, jan, 1992, 10, 26,
    22).
event([chmod, ' ', 0, ' ', fromjim], persin, wednesday, 8, jan, 1992,
    10, 26, 22).
event([mkfile, ' ', fromjim], persin, wednesday, 8, jan, 1992, 10, 26,
    22).
event([cd, ' ', persin], persin, wednesday, 8, jan, 1992, 10, 26, 22).
event([mkdir, ' ', persout], personal, wednesday, 8, jan, 1992, 10, 26,
    22).
event([mkdir, ' ', persin], personal, wednesday, 8, jan, 1992, 10, 26,
    21).
event([cd, ' ', personal], personal, wednesday, 8, jan, 1992, 10, 26,
    21).
event([mkdir, ' ', business], mail, wednesday, 8, jan, 1992, 10, 26,
    21).
event([mkdir, ' ', personal], mail, wednesday, 8, jan, 1992, 10, 26,
    21).
event([cd, ' ', mail], mail, wednesday, 8, jan, 1992, 10, 26, 21).
event([mkdir, ' ', addresses], home, wednesday, 8, jan, 1992, 10, 26,
    21).
event([mkdir, ' ', mail], home, wednesday, 8, jan, 1992, 10, 26, 21).


/* nonevent/12 */

nonevent([cd, ' ', personal, (/), persmail], execute, personal,
    persmail, home, monday, 10, february, 1992, 20, 15, 32).
nonevent([chmod, ' ', 0, ' ', ' ', personal, (/), persmail], reference,
    home, ' ', home, monday, 10, february, 1992, 20, 14, 24).
nonevent([rm, ' ', mailin], logical, nul, nul, home, monday, 10,
    february, 1992, 20, 9, 39).
nonevent([rmdir, ' ', mail], reference, mail, mail, mail, monday, 10,
    february, 1992, 20, 8, 31).
nonevent([mv, ' ', enquirout, ' ', business, (/), busmail, (/), busout,
    (/), inquireout], reference, busmail, busout, home, monday, 10,
    february, 1992, 20, 6, 9).
nonevent([mv, ' ', custaddr, ' ', busaddr], duplication, business,
    busaddr, home, monday, 10, february, 1992, 19, 58, 57).
nonevent([rmdir, ' ', addresses, (/), busaddr], nonempty, nul, nul,
    home, monday, 10, february, 1992, 19, 56, 58).
nonevent([rmdir, ' ', busaddr], reference, home, busaddr, home, monday,
    10, february, 1992, 19, 56, 9).
nonevent([mv, ' ', mail, (/), business, (/), busout, (/), enquireout, '
    ', enquireout], reference, busout, enquireout, home, monday, 10,
    february, 1992, 19, 53, 29).
nonevent([mv, ' ', mail, (/), business, (/), busout, (/), enqout, ' ',
    enqout], reference, busout, enqout, home, monday, 10, february,
    1992, 19, 52, 46).
nonevent([mv, ' ', mail, (/), business, (/), busout, (/), addout],
    reference, busout, addout, home, monday, 10, february, 1992, 19, 52,
    9).
nonevent([mv, ' ', mail, (/), business, (/), busin, (/), complaintin],
    reference, busin, complaintin, home, monday, 10, february, 1992, 19,
    50, 36).
nonevent([mv, ' ', mail, (/), business, (/), in, (/), complaintin],
    reference, business, in, home, monday, 10, february, 1992, 19, 50,
    20).
nonevent([mkdir, ' ', business], duplication, home, business, home,
```

```
nonevent([mv, ' ', tomary, ' ', personal, (/), persmail], reference,
    home, tomary, home, monday, 10, february, 1992, 19, 46, 45).
nonevent([mkdir, ' ', personal, (/), persaddr], duplication, personal,
    persaddr, home, monday, 10, february, 1992, 19, 40, 42).
nonevent([rmdir], syntax, nul, nul, personal, monday, 10, february,
    1992, 19, 38, 57).
nonevent([mv, ' ', persin, (/), frommary, ' ', mail, (/), home, (/),
    temp, (/), frommary], reference, home, temp, personal, monday, 10,
    february, 1992, 19, 36, 20).
nonevent([mv, ' ', persout, (/), frommary, ' ', mail, (/), home, (/),
    temp, (/), frommary], reference, persout, frommary, personal,
    monday, 10, february, 1992, 19, 35, 29).
nonevent([mkdir, ' ', personal], duplication, home, personal, home,
    monday, 10, february, 1992, 19, 31, 40).
nonevent([cd, ' ', temp], logical, nul, nul, home, monday, 10, february,
    1992, 19, 30, 5).
nonevent([cd, ' ', (/), temp], reference, home, (/), home, monday, 10,
    february, 1992, 19, 30, 0).
nonevent([ls, ' ', addresses, (/), busaddr, (/), custaddr], logical,
    nul, nul, home, monday, 10, february, 1992, 19, 23, 16).
nonevent([ls, ' ', addresses, (/), persaddr], read, addresses, persaddr,
    home, monday, 10, february, 1992, 19, 20, 53).
nonevent([lsaddresse], lexical, nul, nul, home, monday, 10, february,
    1992, 19, 20, 40).
nonevent([ls, ' ', (/), mailin], reference, home, (/), home, monday, 10,
    february, 1992, 19, 19, 4).
nonevent([ls, ' ', '-l'], lexical, nul, nul, home, monday, 10, february,
    1992, 19, 18, 15).
nonevent(['ls-l'], lexical, nul, nul, home, monday, 10, february, 1992,
    19, 17, 56).


/* owner/1 */

owner(S4).


/* totaltime/3 */

totaltime(1, 26, 55).


/* file/3 */

file(temp, home, 700).
file(suppaddr, mailin, 700).
file(custaddr, mailin, 700).
file(joneses, mailin, 700).
file(smiths, mailin, 700).
file(fromeric, mailin, 700).
file(enquirout, mailin, 700).
file(addout, mailin, 700).
file(complaintin, mailin, 700).
file(frommary, mailin, 700).
file(fromjim, mailin, 700).
file(tojim, mailin, 700).


/* dir/4 */

dir(root, 0, nul, [home]).
dir(mailin, 700, home, [tojim, fromjim, frommary, complaintin, addout,
    enquirout, fromeric, smiths, joneses, custaddr, suppaddr]).
dir(home, 700, root, [mailin, mail, temp]).
```

```
/* event/9 */

event([ls], home, tuesday, 11, february, 1992, 19, 48, 23).
event([rmdir, ' ', mail, (/), business], business, tuesday, 11,
    february, 1992, 19, 47, 50).
event([cd, ' ', mail, (/), business], business, tuesday, 11, february,
    1992, 19, 47, 14).
event([pwd], home, tuesday, 11, february, 1992, 19, 46, 52).
event([rmdir, ' ', mail, (/), personal], home, tuesday, 11, february,
    1992, 19, 46, 39).
event([rmdir, ' ', mail, (/), personal, (/), persout], home, tuesday,
    11, february, 1992, 19, 46, 1).
event([rmdir, ' ', mail, (/), personal, (/), persin], home, tuesday, 11,
    february, 1992, 19, 45, 28).
event([rmdir, ' ', mail, (/), business, (/), busin], home, tuesday, 11,
    february, 1992, 19, 45, 1).
event([pwd], home, tuesday, 11, february, 1992, 19, 44, 30).
event([rmdir, ' ', mail, (/), business, (/), busout], home, tuesday, 11,
    february, 1992, 19, 43, 22).
event([rmdir, ' ', addresses], home, tuesday, 11, february, 1992, 19,
    42, 36).
event([rmdir, ' ', addresses, (/), busaddr], home, tuesday, 11,
    february, 1992, 19, 42, 22).
event([rmdir, ' ', addresses, (/), persaddr], home, tuesday, 11,
    february, 1992, 19, 41, 49).
event([ls], home, tuesday, 11, february, 1992, 19, 38, 19).
event([cd], home, tuesday, 11, february, 1992, 19, 38, 16).
event([mv, ' ', tojim, ' ', personal, (/), mail, (/), home, (/),
    mailin], persout, tuesday, 11, february, 1992, 19, 37, 30).
event([cd, ' ', personal, (/), persout], persout, tuesday, 11, february,
    1992, 19, 36, 48).
event([mv, ' ', fromjim, ' ', personal, (/), mail, (/), home, (/),
    mailin], persin, tuesday, 11, february, 1992, 19, 36, 21).
event([mv, ' ', frommary, ' ', personal, (/), mail, (/), home, (/),
    mailin], persin, tuesday, 11, february, 1992, 19, 35, 43).
event([cd, ' ', mail, (/), personal, (/), persin], persin, tuesday, 11,
    february, 1992, 19, 35, 5).
event([cd], home, tuesday, 11, february, 1992, 19, 34, 42).
event([mv, ' ', complaintin, ' ', business, (/), mail, (/), home, (/),
    mailin], busin, tuesday, 11, february, 1992, 19, 34, 22).
event([cd, ' ', busin], busin, tuesday, 11, february, 1992, 19, 33, 3).
event([cd, ' ', business], business, tuesday, 11, february, 1992, 19,
    32, 50).
event([mv, ' ', addout, ' ', business, (/), mail, (/), home, (/),
    mailin], busout, tuesday, 11, february, 1992, 19, 32, 33).
event([mv, ' ', enquirout, ' ', business, (/), mail, (/), home, (/),
    mailin], busout, tuesday, 11, february, 1992, 19, 31, 50).
event([cd, ' ', mail, (/), business, (/), busout], busout, tuesday, 11,
    february, 1992, 19, 30, 43).
event([cd], home, tuesday, 11, february, 1992, 19, 30, 15).
event([mv, ' ', fromeric, ' ', home, (/), mailin], mail, tuesday, 11,
    february, 1992, 19, 30, 5).
event([cd, ' ', mail], mail, tuesday, 11, february, 1992, 19, 28, 33).
event([cd], home, tuesday, 11, february, 1992, 19, 28, 24).
event([mv, ' ', smiths, ' ', addresses, (/), home, (/), mailin],
    persaddr, tuesday, 11, february, 1992, 19, 28, 13).
event([mv, ' ', joneses, ' ', addresses, (/), home, (/), mailin],
    persaddr, tuesday, 11, february, 1992, 19, 27, 45).
event([cd, ' ', persaddr], persaddr, tuesday, 11, february, 1992, 19,
    27, 2).
event([cd, ' ', addresses], addresses, tuesday, 11, february, 1992, 19,
    26, 48).
```

```
event([mv, ' ', custaddr, ' ', addresses, (/), home, (/), mailin],
    busaddr, tuesday, 11, february, 1992, 19, 26, 1).
event([pwd], busaddr, tuesday, 11, february, 1992, 19, 22, 58).
event([mv, ' ', suppaddr, ' ', addresses, (/), home, (/), mailin],
    busaddr, tuesday, 11, february, 1992, 19, 22, 25).
event([ls], busaddr, tuesday, 11, february, 1992, 19, 15, 36).
event([cd, ' ', addresses, (/), busaddr], busaddr, tuesday, 11,
    february, 1992, 19, 15, 32).
event([cd], home, tuesday, 11, february, 1992, 19, 14, 53).
event([cd], home, tuesday, 11, february, 1992, 19, 13, 23).
event([ls], busin, tuesday, 11, february, 1992, 19, 12, 29).
event([cd, ' ', business, (/), busin], busin, tuesday, 11, february,
    1992, 19, 12, 25).
event([ls], busout, tuesday, 11, february, 1992, 19, 9, 21).
event([cd, ' ', busout], busout, tuesday, 11, february, 1992, 19, 9,
    17).
event([ls], business, tuesday, 11, february, 1992, 19, 9, 6).
event([cd, ' ', business], business, tuesday, 11, february, 1992, 19, 9,
    3).
event([cd, ' ', mail], mail, tuesday, 11, february, 1992, 19, 8, 47).
event([cd], home, tuesday, 11, february, 1992, 19, 8, 42).
event([ls], persout, tuesday, 11, february, 1992, 19, 7, 39).
event([ls], persout, tuesday, 11, february, 1992, 19, 7, 26).
event([cd, ' ', persout], persout, tuesday, 11, february, 1992, 19, 7,
    22).
event([cd, ' ', personal], personal, tuesday, 11, february, 1992, 19, 7,
    14).
event([ls], persin, tuesday, 11, february, 1992, 19, 6, 6).
event([cd, ' ', persin], persin, tuesday, 11, february, 1992, 19, 6, 2).
event([ls], personal, tuesday, 11, february, 1992, 19, 5, 37).
event([cd, ' ', personal], personal, tuesday, 11, february, 1992, 19, 4,
    32).
event([chmod, ' ', 700, ' ', personal], mail, tuesday, 11, february,
    1992, 19, 4, 21).
event([cd, ' ', mail], mail, tuesday, 11, february, 1992, 19, 2, 39).
event([cd], home, tuesday, 11, february, 1992, 19, 2, 29).
event([ls], business, tuesday, 11, february, 1992, 19, 2, 8).
event([cd, ' ', business], business, tuesday, 11, february, 1992, 19, 2,
    4).
event([chmod, ' ', 700, ' ', business], mail, tuesday, 11, february,
    1992, 19, 1, 54).
event([cd, ' ', mail], mail, tuesday, 11, february, 1992, 19, 1, 24).
event([cd, ' ', business], business, tuesday, 11, february, 1992, 19, 0,
    17).
event([cd], home, tuesday, 11, february, 1992, 18, 57, 51).
event([cd, ' ', mail], mail, tuesday, 11, february, 1992, 18, 57, 32).
event([cd], home, tuesday, 11, february, 1992, 18, 57, 15).
event([ls], busaddr, tuesday, 11, february, 1992, 18, 54, 29).
event([cd, ' ', busaddr], busaddr, tuesday, 11, february, 1992, 18, 54,
    24).
event([ls], addresses, tuesday, 11, february, 1992, 18, 54, 9).
event([cd, ' ', addresses], addresses, tuesday, 11, february, 1992, 18,
    54, 6).
event([ls], home, tuesday, 11, february, 1992, 18, 53, 54).
event([cd], home, tuesday, 11, february, 1992, 18, 53, 45).
event([rm, ' ', johnsons], persaddr, tuesday, 11, february, 1992, 18,
    53, 18).
event([rm, ' ', bloggs], persaddr, tuesday, 11, february, 1992, 18, 52,
    37).
event([ls], persaddr, tuesday, 11, february, 1992, 18, 51, 35).
event([cd, ' ', persaddr], persaddr, tuesday, 11, february, 1992, 18,
    51, 30).
event([chmod, ' ', 700, ' ', persaddr], addresses, tuesday, 11,
    february, 1992, 18, 51, 20).
```

```
event([cd, ' ', addresses], addresses, tuesday, 11, february, 1992, 18,
    51, 2).
event([cd], home, tuesday, 11, february, 1992, 18, 50, 56).
event([chmod, ' ', 400, ' ', persaddr], addresses, tuesday, 11,
    february, 1992, 18, 49, 54).
event([cd, ' ', addresses], addresses, tuesday, 11, february, 1992, 18,
    46, 0).
event([cd], home, tuesday, 11, february, 1992, 18, 45, 50).
event([ls], home, tuesday, 11, february, 1992, 18, 44, 4).
event([cd], home, tuesday, 11, february, 1992, 18, 44, 0).
event([ls], mail, tuesday, 11, february, 1992, 18, 43, 17).
event([cd, ' ', mail], mail, tuesday, 11, february, 1992, 18, 43, 13).
event([cd], home, tuesday, 11, february, 1992, 18, 43, 8).
event([ls], mailin, tuesday, 11, february, 1992, 18, 42, 31).
event([cd, ' ', mailin], mailin, tuesday, 11, february, 1992, 18, 42,
    29).
event([cd], home, tuesday, 11, february, 1992, 18, 42, 20).
event([ls], addresses, tuesday, 11, february, 1992, 18, 40, 46).
event([cd, ' ', addresses], addresses, tuesday, 11, february, 1992, 18,
    40, 41).
event([cd], home, tuesday, 11, february, 1992, 18, 40, 30).
event([ls], home, tuesday, 11, february, 1992, 18, 39, 42).
event([cd], home, tuesday, 11, february, 1992, 18, 39, 36).
event([ls], mailin, tuesday, 11, february, 1992, 18, 39, 0).
event([cd, ' ', mailin], mailin, tuesday, 11, february, 1992, 18, 38,
    52).
event([cd], home, tuesday, 11, february, 1992, 18, 32, 34).
event([pwd], home, tuesday, 11, february, 1992, 18, 32, 11).
event([ls], home, tuesday, 11, february, 1992, 18, 23, 20).
event([pwd], home, tuesday, 11, february, 1992, 18, 22, 54).
event([chmod, ' ', 200, ' ', home, (/), addresses, (/), persaddr], mail,
    wednesday, 8, jan, 1992, 10, 26, 27).
event([chmod, ' ', 100, ' ', business], mail, wednesday, 8, jan, 1992,
    10, 26, 27).
event([mkfile, ' ', business, (/), busout, (/), addout], mail,
    wednesday, 8, jan, 1992, 10, 26, 27).
event([mkfile, ' ', fromeric], mail, wednesday, 8, jan, 1992, 10, 26,
    27).
event([cd, ' ', mail], mail, wednesday, 8, jan, 1992, 10, 26, 27).
event([mkdir, ' ', mailin], home, wednesday, 8, jan, 1992, 10, 26, 27).
event([cd], home, wednesday, 8, jan, 1992, 10, 26, 26).
event([mkfile, ' ', johnsons], persaddr, wednesday, 8, jan, 1992, 10,
    26, 26).
event([mkfile, ' ', bloggs], persaddr, wednesday, 8, jan, 1992, 10, 26,
    26).
event([mkfile, ' ', joneses], persaddr, wednesday, 8, jan, 1992, 10, 26,
    26).
event([mkfile, ' ', smiths], persaddr, wednesday, 8, jan, 1992, 10, 26,
    26).
event([cd, ' ', addresses, (/), persaddr], persaddr, wednesday, 2, may,
    1991, 10, 26, 26).
event([mkfile, ' ', custaddr], busaddr, wednesday, 8, jan, 1992, 10, 26,
    25).
event([mkfile, ' ', suppaddr], busaddr, wednesday, 8, jan, 1992, 10, 26,
    25).
event([cd, ' ', busaddr], busaddr, wednesday, 8, jan, 1992, 10, 26, 25).
event([mkdir, ' ', persaddr], addresses, wednesday, 8, jan, 1992, 10,
    26, 25).
event([mkdir, ' ', busaddr], addresses, wednesday, 8, jan, 1992, 10, 26,
    25).
event([cd, ' ', business, (/), mail, (/), home, (/), addresses],
    addresses, wednesday, 8, jan, 1992, 10, 26, 25).
event([mkfile, ' ', enquirout], busout, wednesday, 8, jan, 1992, 10, 26,
    25).
event([cd, ' ', business, (/), busout], busout, wednesday, 8, jan, 1992,
```

```
event([chmod, ' ', 400, ' ', complaintin], busin, wednesday, 2, may,
    1991, 10, 26, 24).
event([mkfile, ' ', complaintin], busin, wednesday, 8, jan, 1992, 10,
    26, 24).
event([cd, ' ', busin], busin, wednesday, 8, jan, 1992, 10, 26, 24).
event([mkdir, ' ', busout], business, wednesday, 8, jan, 1992, 10, 26,
    24).
event([mkdir, ' ', busin], business, wednesday, 8, jan, 1992, 10, 26,
    24).
event([cd, ' ', business], business, wednesday, 8, jan, 1992, 10, 26,
    23).
event([chmod, ' ', 0, ' ', personal], mail, wednesday, 8, jan, 1992, 10,
    26, 23).
event([cd, ' ', personal, (/), mail], mail, wednesday, 8, jan, 1992, 10,
    26, 23).
event([chmod, ' ', 0, ' ', tojim], persout, wednesday, 8, jan, 1992, 10,
    26, 23).
event([mkfile, ' ', tojim], persout, wednesday, 8, jan, 1992, 10, 26,
    23).
event([cd, ' ', personal, (/), persout], persout, wednesday, 2, may,
    1991, 10, 26, 23).
event([chmod, ' ', 0, ' ', frommary], persin, wednesday, 8, jan, 1992,
    10, 26, 23).
event([mkfile, ' ', frommary], persin, wednesday, 8, jan, 1992, 10, 26,
    22).
event([chmod, ' ', 0, ' ', fromjim], persin, wednesday, 8, jan, 1992,
    10, 26, 22).
event([mkfile, ' ', fromjim], persin, wednesday, 8, jan, 1992, 10, 26,
    22).
event([cd, ' ', persin], persin, wednesday, 8, jan, 1992, 10, 26, 22).
event([mkdir, ' ', persout], personal, wednesday, 8, jan, 1992, 10, 26,
    22).
event([mkdir, ' ', persin], personal, wednesday, 8, jan, 1992, 10, 26,
    21).
event([cd, ' ', personal], personal, wednesday, 8, jan, 1992, 10, 26,
    21).
event([mkdir, ' ', business], mail, wednesday, 8, jan, 1992, 10, 26,
    21).
event([mkdir, ' ', personal], mail, wednesday, 8, jan, 1992, 10, 26,
    21).
event([cd, ' ', mail], mail, wednesday, 8, jan, 1992, 10, 26, 21).
event([mkdir, ' ', addresses], home, wednesday, 8, jan, 1992, 10, 26,
    21).
event([mkdir, ' ', mail], home, wednesday, 8, jan, 1992, 10, 26, 21).


/* nonevent/12 */

nonevent([cd], lexical, nul, nul, home, tuesday, 11, february, 1992, 19,
    48, 16).
nonevent([pwd], lexical, nul, nul, business, tuesday, 11, february,
    1992, 19, 48, 6).
nonevent([rmdir, ' ', mail], reference, business, mail, business,
    tuesday, 11, february, 1992, 19, 47, 58).
nonevent([rmdir, ' ', mail, ' ', personal], syntax, nul, nul, home,
    tuesday, 11, february, 1992, 19, 46, 24).
nonevent([rmdir, ' ', busin], reference, home, busin, home, tuesday, 11,
    february, 1992, 19, 44, 13).
nonevent([rmdir, ' ', business, (/), busin], reference, home, business,
    home, tuesday, 11, february, 1992, 19, 43, 50).
nonevent([rmdir, ' ', addresses], nonempty, nul, nul, home, tuesday, 11,
    february, 1992, 19, 40, 45).
nonevent([rmdiraddresses], lexical, nul, nul, home, tuesday, 11,
    february, 1992, 19, 40, 28).
```

nonevent([cd, ' ', syntax, nul, nul, busin, tuesday, 11, february, 1992, 19, 34, 40).
nonevent([mv, ' ', fromeric, ' ', mail, (/), home, (/), mailin], reference, mail, mail, mail, tuesday, 11, february, 1992, 19, 29, 6).
nonevent([cd, ' ', persaddr], reference, home, persaddr, home, tuesday, 11, february, 1992, 19, 26, 35).
nonevent([mv, ' ', custaddr, ' ', addresses, (/), root, (/), mailin], reference, addresses, root, busaddr, tuesday, 11, february, 1992, 19, 25, 28).
nonevent([mv, ' ', busaddr, ' ', addresses, (/), root, (/), mailin], reference, busaddr, busaddr, busaddr, tuesday, 11, february, 1992, 19, 24, 32).
nonevent([cd, ' ', suppaddr], logical, nul, nul, busaddr, tuesday, 11, february, 1992, 19, 16, 19).
nonevent([cd, ' ', custaddr], logical, nul, nul, busaddr, tuesday, 11, february, 1992, 19, 16, 0).
nonevent([cd, ' ', smiths], logical, nul, nul, home, tuesday, 11, february, 1992, 19, 14, 45).
nonevent([cd, ' ', addresses, (/), persaddr, (/), joneses], logical, nul, nul, home, tuesday, 11, february, 1992, 19, 14, 17).
nonevent([cd, ' ', complaintin], logical, nul, nul, busin, tuesday, 11, february, 1992, 19, 13, 2).
nonevent([cd, ' ', busout, (/), business, (/), busin], reference, busout, busout, busout, tuesday, 11, february, 1992, 19, 11, 59).
nonevent([cd, ' ', enquirout], logical, nul, nul, busout, tuesday, 11, february, 1992, 19, 10, 57).
nonevent([cd, ' ', busout], reference, busout, busout, busout, tuesday, 11, february, 1992, 19, 10, 20).
nonevent([cd, ' ', enquireout], reference, busout, enquireout, busout, tuesday, 11, february, 1992, 19, 10, 2).
nonevent([cd, ' ', addout], logical, nul, nul, busout, tuesday, 11, february, 1992, 19, 9, 44).
nonevent([cd, ' ', tojim], logical, nul, nul, persout, tuesday, 11, february, 1992, 19, 8, 1).
nonevent([cd, ' ', to, ' ', jim], syntax, nul, nul, persout, tuesday, 11, february, 1992, 19, 7, 47).
nonevent([cd, ' ', fromjim], logical, nul, nul, persin, tuesday, 11, february, 1992, 19, 6, 59).
nonevent([cd, ' ', frommary], logical, nul, nul, persin, tuesday, 11, february, 1992, 19, 6, 46).
nonevent([cd, ' ', mail], reference, mail, mail, mail, tuesday, 11, february, 1992, 19, 2, 59).
nonevent([cd, ' ', personal], execute, mail, personal, mail, tuesday, 11, february, 1992, 19, 2, 48).
nonevent([cd, ' '], syntax, nul, nul, business, tuesday, 11, february, 1992, 19, 2, 26).
nonevent([chmod, ' ', 700, ' ', business], reference, business, business, business, tuesday, 11, february, 1992, 19, 1, 2).
nonevent([ls], read, mail, business, business, tuesday, 11, february, 1992, 19, 0, 21).
nonevent([cd, ' ', fromeric], logical, nul, nul, mail, tuesday, 11, february, 1992, 19, 0, 1).
nonevent([cd, ' ', mail], lexical, nul, nul, mail, tuesday, 11, february, 1992, 18, 59, 53).
nonevent([mkdir], lexical, nul, nul, home, tuesday, 11, february, 1992, 18, 58, 32).
nonevent([cd, ' ', mail], reference, mail, mail, mail, tuesday, 11, february, 1992, 18, 57, 47).
nonevent([cd, ' ', personal], execute, mail, personal, mail, tuesday, 11, february, 1992, 18, 57, 41).
nonevent([cd, ' ', addresses], reference, addresses, addresses, addresses, tuesday, 11, february, 1992, 18, 50, 44).
nonevent([cd, ' ', persaddr], execute, addresses, persaddr, addresses, tuesday, 11, february, 1992, 18, 50, 7).

```
nonevent([chmod, ' ', 400], syntax, nul, nul, addresses, tuesday, 11,
    february, 1992, 18, 47, 46).
nonevent([cd, ' ', persaddr], execute, addresses, persaddr, addresses,
    tuesday, 11, february, 1992, 18, 46, 14).
nonevent([cd, ' ', temp], logical, nul, nul, home, tuesday, 11,
    february, 1992, 18, 44, 20).
nonevent([cd, ' ', addresses], reference, mailin, addresses, mailin,
    tuesday, 11, february, 1992, 18, 39, 23).
nonevent([cd, ' ', address], reference, mailin, address, mailin,
    tuesday, 11, february, 1992, 18, 39, 15).
nonevent([cd, ' ', business], reference, home, business, home, tuesday,
    11, february, 1992, 18, 36, 36).
nonevent([ls, ' ', '-l'], lexical, nul, nul, home, tuesday, 11,
    february, 1992, 18, 31, 14).
nonevent(['ls-l'], lexical, nul, nul, home, tuesday, 11, february, 1992,
    18, 31, 3).
```

Appendix K
Protocol Analysis Commentaries

Appendix K contains commentaries on all of the context errors committed by all subjects involved in the protocol analysis evaluative exercise. These were produced from synchronised video tapes and interaction logs from a session with each subject. Something in the order of six hours of video tape was scrutinised, in conjunction with printed versions of the log files, so that precise descriptions of the error situation could be formulated. Subjects' comments and replies to questions formed the basis of judgments concerning misconceptions about system states and functions, which are used to estimate the accuracy of the stated misconception diagnosis criteria. An estimation of the likely effectiveness of help information, based on this diagnosis, is then made.

nonevent(Command, Errortype, Parent, Target, Cwd, Day, Date, Month, Year, Hour, Minute, Second).

nonevent([cd, ' ', persaddr], execute, addresses, persaddr, addresses, monday, 20, january, 1992, 18, 4, 44).

Description: This was an attempt to make the execute-protected directory 'persaddr' the working directory. The subject was not aware that having execute permission is a precondition of this action.
Commentary: A prior action in the log, removing execute permission from the directory would provide evidence for a state misconception here. However, this mode had been set in the startup file and was not one of the subject's own actions and is thus misleading. The absence of such evidence would lead to the provision of state and functional information regarding the mode set on 'persaddr' and the precondition of execute permission on directory changes, along with directions for using the 'chmod' command. Such would seem adequate in the current situation.

nonevent([chmod, ' ', 100, ' ', root, (/), home, (/), addresses], reference, addresses, root, addresses, monday, 20, january, 1992, 18, 6, 23).

Description: In trying to remedy the previous error, a reference error occurred, arising from the (popular) misconception that paths always start from the *home* or *root* directory instead of the *cwd*.
Commentary: This reference error would be dealt with without the benefit of any evidence from the log and help information would thus be based on relevant aspects of the current state, such as the *cwd* and location of the target, and violated preconditions on referencing via paths, such as that each item in the path list must be the parent or child of its predecessor in the list and that the first item must be a child or parent of the *cwd*.

nonevent([ls, ' ', '-l'], read, addresses, persaddr, persaddr, monday, 20, january, 1992, 18, 12, 19).

Description: In granting execute permission on the directory 'persaddr' by giving it a code of 100, the subject had removed read permission, which is a precondition of the 'ls' command.
Commentary: The prior mode setting action would be taken as evidence for a state misconception here but it is not clear that this is correct. It is more likely that the subject was unaware of the precondition of read-permission on the 'ls' command when the 'chmod' command was used, as there was probably an intention to list the directory even at that time. The criteria would suggest a state misconception, when the facts point to a functional misconception, indicating a weakness in the criteria adopted.

whether this kind of error could be addressed as a planning error at an earlier time (when, for example, read permission was removed) would depend upon whether the intention to list the directory could be reliably predicted as part of some larger plan.

nonevent([chmod, ' ', 400, ' ', persaddr], reference, persaddr, persaddr, persaddr, monday, 20, january, 1992, 18, 13, 17).

Description: In attempting to reassign read permission to 'persaddr' from within 'persaddr', this reference error occurred. The error was greeted with "Where am I?".
Commentary: It seems that the subject had forgotten the identity of the *cwd*. The last directory change would be taken as the causal antecedent action and would thus provide evidence for a state misconception concerning the *cwd*, based on the idea that this action had been forgotten. The criteria of logical and psychological proximity would support this diagnosis, which in this case, is correct. State information concerning the *cwd*, the location of the target and the last directory change would seem appropriate to addressing this misconception.

nonevent([cd, ' ', persaddr], reference, persaddr, persaddr, persaddr, monday, 20, january, 1992, 18, 16, 34).

Description: 'persaddr' was already the *cwd* when the subject tried to make it so; the classic "please close the window" error.
Commentary: The error of trying to make a state of affairs the case which already obtains is a logical error which can be dealt with at a high level of abstraction, although the reason for the error may require detailed examination. The previous directory change would operate evidentially in the same way as in 18:13:17.

nonevent([cd, ' ', root, (/), home, (/), addresses], reference, persaddr, root, persaddr, monday, 20, january, 1992, 18, 17, 37).

Description: Repetition of the reference error stemming from the misconception that paths must start from the 'home' or 'root' directory.
Commentary: See 18:6:23

nonevent([cd, ' ', persaddr, (/), addresses], reference, persaddr, persaddr, persaddr, monday, 20, january, 1992, 18, 18, 10).

Description: Another attempt to change directory to what was already the *cwd*.
Commentary: See 18:13:17.

nonevent([cd, ' ', persaddr], execute, addresses, persaddr, addresses, monday, 20, january, 1992, 18, 20, 30).

Description: This time, the subject had changed the mode setting on 'persaddr' to get read-permission and, at the same time, removed execute-permission. This attempt to make 'persaddr' the *cwd* thus failed; the reverse of the situation in 18:12:19.
Commentary: The prior change of mode logged would have suggested a state misconception and the subject's prior experience with the precondition of execute-permission on the 'cd' command would tend to support this. Information concerning the current mode setting on 'persaddr' would thus be provided. See 18:12:19.

nonevent([cd, ' ', addresses], reference, addresses, addresses, addresses, monday, 20, january, 1992, 18, 22, 4).

Description: Another example of locational disorientation with regard to the *cwd.*
Commentary: See 18:13:17.

nonevent([ls, ' ', '-l'], read, mail, business, business, monday, 20, january, 1992, 18, 31, 59).

Description: An attempt to list 'business' without the presupposed read-permission.
Commentary: See 18:4:44.

nonevent([chmod, ' ', 700, ' ', complaintin], reference, business, complaintin, business, monday, 20, january, 1992, 18, 36, 42).

Description: Here, the subject had moved up a level in the hierarchy and then tried to access a child of the previous *cwd.*
Commentary: This state misconception would be identified on the basis of the last directory change, which, had it not occurred, would have rehabilitated this command. Information on the relative positions of the relevant items would be provided. Details of the presumed causal antecedent action would reinforce the subjects understanding of the causal relationships between actions.

nonevent([cd, ' ', business], reference, business, business, business, monday, 20, january, 1992, 18, 37, 38).

Description: Another case of attempting to bring about something already obtaining.
Commentary: See 18:13:17.

nonevent([cd, ' ', business], reference, home, business, home, monday, 20, january, 1992, 18, 48, 37).

Description: Here, the subject had presumed that she had changed directory to 'mail', when this was not the case. Consequently, she tried to change directory to 'business' from 'home', without supplying 'mail' in the path. "I forgot I'd not gone into it [mail]", she said.
Commentary: No single, prior action in the log constitutes evidence for a forgotten change in state, in this instance. Locational information concerning relevant items and rules concerning accessing remote items would therefore be provided here.

nonevent([cd, ' ', busout], reference, home, busout, home, monday, 20, january, 1992, 18, 48, 40).

Description: This is really the same error as the previous one, committed immediately after it.
Commentary: See 18:48:37.

nonevent([cd, ' ', persin], reference, persout, persin, persout, monday, 20, january, 1992, 18, 51, 4).

Description: An attempt to move laterally in the directory structure without supplying the path. It seems that the last directory move from the parent had been forgotten.
Commentary: Again the last change of directory in the log provides good evidence for the state misconception of assuming a prior state to obtain. Details of the current state and the action which transformed it from the presumed state are pertinent here.

nonevent([mkdir, ' ', business], duplication, home, business, home, monday, 20, january, 1992, 19, 3, 9).

nonevent([mkdir, ' ', business], duplication, home, business, home, monday, 20, january, 1992, 19, 4, 4).

Description: Both of these commands were attempts to duplicate a directory name. The subject was unaware of the bar on duplicated names.
Commentary: The existence of an action in the log, creating or naming the directory 'business' would be taken as evidence for a state misconception concerning the existence of that directory. This could produce incorrect diagnoses, in that the creation of an item does not imply knowledge about rules concerning duplication. This indicates that a possible extension to the criteria for diagnosis might be some kind of representation of the implications for user knowledge of the various commands which might be found in the log. The use of the 'chmod' command might indicate

the use of 'mkdir' does not indicate any familiarity with the associated rules of duplication. We have indicated that this kind of inference is rather tenuous but it could prove a useful source of negative evidence to suppress incorrect diagnoses, rather than as support for particular diagnoses.

nonevent([mv, ' ', mail, (/), business, (/), busout, (/), enquirout, ' ', home, (/), 'temp]'], reference, home, home, home, monday, 20, january, 1992, 19, 14, 6).

Description: Here, the *cwd* ('home') was placed at the head of a path. Interestingly, the first argument path correctly omits the *cwd*, indicating that this was not a genuine misconception. The consequent error message was greeted with "'Cos I'm in 'home', stupid girl!" and the error was quickly detected, which also supports the idea that this was a 'slip'.
Commentary: The 'undoing' of no single prior action would have rehabilitated the command, so that state and functional information would be presented. In this instance, a mere reminder of the identity of the *cwd* would have been sufficient but it is better that the user has a little too much information rather than not enough.

nonevent([mv, ' ', complaintin, ' ', temp], duplication, home, temp, busin, monday, 20, january, 1992, 19, 18, 32).

Description: Because of the dual function or ambiguity of the 'mv' command in UNIX; both in moving and renaming items, it is impossible for the system to tell if this is a duplication or reference error. It is treated as the former, when in fact it is the latter. This obviously makes for highly confusing error reports, since the error message bears no relation to the user's intentions. The subject was attempting to move a file ('complaintin') from the *cwd* ('busin') to a 'great uncle' directory ('temp'), without supplying the correct ('business/home') path. The system sees this as an attempt to rename 'complaintin' to 'temp' and rejects this name duplication.
Commentary: One answer here is to remove the inherent ambiguity, so that the system at least has access to the meaning of the action. The pragmatics of the action can then be addressed more accurately. Alternatively, the classification of error type would follow and depend on the possible worlds analysis, so that both interpretations of the error could be pursued and compared. In this case, there is no evidence to support either option over the other, which would lead to information relating to both being provided as alternative user misconceptions. The user could then choose the most appropriate interpretation.

nonevent([mv, ' ', complaintin, ' ', home, (/), temp], reference, busin, home, busin, monday, 20, january, 1992, 19, 19, 9).

Description: Another attempt to achieve the intention of the last erroneous command. Again, the destination path was given incorrectly. It was not clear to an observer whether this stemmed from a lack of awareness of the structure of the hierarchy or was an instance of the function misconception that paths start from the 'home' directory (or both).
Commentary: In the event, no evidence for a state misconception is available, so both kinds of information would be presented.

nonevent([ls, ' ', '-l', ' ', home, (/), temp], reference, busin, home, busin, monday, 20, january, 1992, 19, 21, 53).

Description: A path error related to the previous one.
Commentary: See 19:19:9.

nonevent([ls, ' ', '-l', ' ', temp], reference, busin, temp, busin, monday, 20, january, 1992, 19, 21, 58).

Description: Ditto.
Commentary: See 19:19:9.

nonevent([mv, ' ', custaddr, ' ', busaddr, (/), addresses, ' ', temp], reference, busaddr, busaddr, busaddr, monday, 20, january, 1992, 19, 32, 6).

Description: The *cwd* given as the head of the path. The subject expressed general confusion about the construction of paths and it could be that some of these errors resulted from 'experiments' in path construction.
Commentary: Trying out different 'rules' would tend to provoke the help system to generate state and functional information, since it would tend to imply no particular state misconception. Part of the rationale of the approach here is that the user should be rewarded for experimentation, rather than deterred from it. Thus, it is appropriate for experimental behaviour to engender the production of information, both on aspects of the current state of the system and rules relating to the correct use of the command.

nonevent([ls, ' ', '-l', ' ', home, (/), temp], reference, busaddr, home, busaddr, monday, 20, january, 1992, 19, 35, 34).

nonevent([ls, ' ', '-l', ' ', temp], reference, busaddr, temp, busaddr, monday, 20, january, 1992, 19, 35, 41).

nonevent([cd, ' ', busaddr], reference, home, busaddr, home, monday, 20, january, 1992, 19, 37, 24).

nonevent([cd, ' ', addresses], reference, business, addresses, business, monday, 20, january, 1992, 19, 37, 55).

Description: Again, general confusion concerning the construction of paths resulted in these four errors.
Commentary: Rules of path construction would be provided here, since no causal antecedent actions were present in the log. See 19:19:9.

nonevent([rmdir, ' ', busaddr], nonempty, nul, nul, addresses, monday, 20, january, 1992, 19, 38, 20).

Description: The subject was not aware of the rule that a directory must be empty before it can be deleted.
Commentary: The 'undoing' of no single event in the log would have rendered this command executable, therefore state and functional information would be presented. This would seem to address the problem.

nonevent([rmdir, ' ', addresses, (/), busaddr], reference, temp, addresses, temp, monday, 20, january, 1992, 19, 40, 14).

Description: The subject had forgotten the prior directory change to 'temp' from 'home' and so tried to access 'addresses' as a sibling of the *cwd*, without the necessary path 'home'.
Commentary: The directory change stands as good evidence in detecting this misconception, since its undoing would rehabilitate the command. The state information, along with a reminder of the antecedent action would be given to remedy the misconception.

nonevent([mv, ' ', home, (/), temp, (/), custaddr, ' ', business, (/), busaddr], reference, home, home, home, monday, 20, january, 1992,19, 42, 42).

Description: Locational disorientation concerning the *cwd*.
Commentary: See 18:13:17.

nonevent([mv, ' , temp, (/), suppadr, ' ', business, (/), busaddr], reference, temp, suppadr, home, monday, 20, january, 1992, 19, 44, 14).

Description: 'suppadr' used for 'suppaddr'.
Commentary: Spell checking would have dealt with this error. As it was, the subject took it as a genuine reference error, rather than a typographical one and was consequently puzzled as to the exact nature of the error.

nonevent([cd, ' ', tep], reference, home, tep, home, monday, 20, january, 1992, 19, 44, 24).

Description: 'tep' used for 'temp'.
Commentary: See 19:44:14.

nonevent([mv, ' ', temp, (/), suppadr, ' ', business, (/), busaddr], reference, temp, suppadr, home, monday, 20, january, 1992, 19, 45, 0).

Description: 'suppadr' used for 'suppaddr'.
Commentary: See 19:44:14.

nonevent([mkdir, ' ', personal], duplication, home, personal, home, monday, 20, january, 1992, 19, 49, 24).

Description: Here, the subject had forgotten the existence of the directory 'personal' and tried to create another directory of that name.
Commentary: See 19:3:9.

nonevent([mv , ' ', mail, (/), personal, (/), persout, (/), tojim, ' ', persout, (/), personal, (/), mail, (/), home, (/), temp], reference, home, persout, home, monday, 20, january, 1992, 19, 55, 18).

Description: In this case, the destination path was given in reverse; from target to *cwd*. The subject had not really grasped the rules of path construction throughout the session.
Commentary: See 18:6:23. Even verbal explanations proved unsuccessful here and so automated help would stand little chance of success. It could be that the pressure of the situation exacerbated this confusion and that the opportunity to explore the system in less stressful circumstances would have provided a remedy, so long as the system rewarded such exploration by providing information relevant to it.

Summary: There were thirty five context errors in this user's session log. On three occasions, the inferences drawn from the diagnostic criteria were inappropriate to the user's particular situation, in that some of the information germane to the error would not have been provided or that some misleading information would have been given. On three occasions, the user would have received information relating to system rules, as well as relevant state information, when the misconception was state-based. On the other twenty nine occasions, the misconception diagnosed via the stated criteria broadly matched that apparent in the analysis of the recorded protocol and log file.

nonevent([cd, ' ', addresses], reference, mailin, addresses, mailin, tuesday, 21, january, 1992, 17, 36, 34).

Description: This was an attempt to change the *cwd* to a sibling directory 'addresses' from 'mailin', without supplying the parent as the path. The user said at this point "I think I've lost where I am" indicating that he did was unsure of identity of the *cwd* at this point.
Commentary: The last change of directory from 'home' to 'mailin' would be taken as evidence for the diagnosis of the misconception that 'home' was the *cwd*. The present *cwd* and the location of 'addresses' would thus be deemed as sufficient information to enable the user to recover and correct the misconception. Otherwise, the violated precondition governing directory changes would also be signalled.

nonevent([cd, ' ', persaddr], execute, addresses, persaddr, addresses, tuesday, 21, january, 1992, 17, 38, 19).

Description: An attempt was made to change directory to 'persaddr' which was execute protected. The user was unaware of this protected state. It had been set in the startup file as part of the task definition and not set by the present user.
Commentary: The presence of the command in the log, setting the protection on that directory, would be taken as evidence for psychological proximity of a possible world in which that command had not occurred and in which the directory was therefore not protected. It is assumed that the user himself set this value as the command is in his log. Information designed to correct this misconception would be presented. If no causal antecedent were present, information regarding preconditions on directory changes would also be presented.

nonevent([mv, ' ', address, ' ', personal], reference, home, address, home, tuesday, 21, january, 1992, 17, 43, 51).

Description: 'address' was used for 'addresses'.
Commentary: A spell checker would tackle this error prior to it being interpreted as a context error. If there were causal antecedents relating to an item of that name, for example, a previous deletion, then a state misconception would be diagnosed on the basis of this evidence for psychological proximity of the possible world in which the deletion had not occurred. Absence of such evidence would lead to the presentation of state information, ie: that the item does not exist, along with the relevant precondition of existence on the command.

tuesday, 21, january, 1992, 17, 44, 22).

Description: An attempt was made to create a duplicate directory 'personal'. The user was not aware of the presence of this directory at a lower level.
Commentary: See 17:38:19.

nonevent([cd, ' ', persin], reference, persout, persin, persout, tuesday, 21, january, 1992, 17, 48, 9).

Description: An attempt was made to move directly to a sibling directory. The user said at this point "That failed 'cos I can't hop directories" indicating that the user was unsure of and testing the preconditions on directory changes and also that an earlier diagnosis (17:36:34) may have been unsound.
Commentary: Here a previous directory change would have indicated a state misconception rather than the apparent functional one, confirming that the stated criteria for diagnosis need strengthening.

nonevent([cd, ' ', person], reference, persout, person, persout, tuesday, 21, january, 1992, 17, 48, 31).

Description: 'person' was used for 'personal'.
Commentary: See 17:43:51.

nonevent([cd, ' ', person], reference, persout, person, persout, tuesday, 21, january, 1992, 17, 49, 23).

Description: Another use of 'person' for 'personal'.
Commentary: See 17:43:51.

nonevent([cd, ' ', mail], reference, persin, mail, persin, tuesday, 21, january, 1992, 17, 50, 4).

Description: An attempt was made to move directly to a grandparent directory. The user had forgotten which directory was the *cwd*.
Commentary: If the last directory change had been from the parent or child of the target, this would have indicated a state misconception and prompted the provision of relevant state information. The absence of such a prior action would necessitate information regarding the violated preconditions on directory changes, which would have been superfluous in this instance.

21, january, 1992, 17, 53, 30).

Description: This is a classic 'please close the window' example, where a command is issued in order to achieve a state which actually already obtains. An attempt is made to move to what is already the *cwd.*
Commentary: The move to 'busout' must, by definition, have been the last directory change. This would be taken as evidence for the psychological proximity of a possible world where that change had not occurred and a state misconception to that effect would be diagnosed. Thus, state information indicating the *cwd* would be presented, along with information relating to the action which brought about that state. This would seem to be exactly what was required in this instance.

nonevent([mv, ' ', mail, ' ', scrap], reference, mail, mail, mail, tuesday, 21, january, 1992, 18, 0, 46).

Description: Here again, the user had forgotten the identity of the *cwd* and attempted to access it from within it. There is a complication in that the command also attempted to move a directory, which is not allowed in this simulation. The latter error was repeated once the user had recovered from the reference error.
Commentary: For the reference error see 17:50:4. Compound errors such as this ought to be dealt with simultaneously in a "by the way..." sort of manner, so that users are not forced to repeat errors in order to discover them. Unless there were evidence to support the idea that the user believed there to be a file called 'mail' to which he was referring, the interpretations available are the state misconception that the directory 'mail' is thought to be a file or the functional misconception that directories can be moved. Information regarding these two facts is sufficient to remedy this situation.

nonevent([mkdir, ' ', persaddr], duplication, personal, persaddr, personal, tuesday, 21, january, 1992, 18, 10, 38).

Description: An attempt to duplicate the directory 'persaddr'.
Commentary: See 17:38:19.

nonevent([mv, ' ', scrap, (/), tojim], reference, persmail, scrap, persmail, tuesday, 21, january, 1992, 18, 12, 57).

Description: This compound error arose from the twin misconceptions that (a) all pathnames start from the home directory, whatever the identity of the *cwd* and (b) *mv* is like *ls*, in that the *cwd* is the default argument.
Commentary: See 18:0:46 on compound errors. This should also have been signalled as a syntax error to the effect that *mv* requires two

arguments, one for the source and one for the destination. The reference error would be dealt with without the benefit of any evidence from the log and help information would thus be based on relevant aspects of the current state, such as the *cwd* and location of the target, and violated preconditions on referencing via paths, such as that each item in the path list must be the parent or child of its predecessor in the list and that the first item must be a child or parent of the *cwd*.

nonevent([mv, ' ', home, (/), scrap, (/), tojim], reference, persmail, home, persmail, tuesday, 21, january, 1992, 18, 13, 27).

Description: This is a repeat of the last error.
Commentary: See 18:12:57.

nonevent([mv, ' ', tojim, ' ', personal, (/), permail, (/), tojim], reference, personal, permail, home, tuesday, 21, january, 1992, 18, 16, 40).

Description: A use of 'permail' for 'persmail'.
Commentary: See 17:43:51.

nonevent([mv, '', tojim, ' ', personnal, (/), persmail, (/), tojim], reference, home, personnal, home, tuesday, 21, january, 1992, 18, 17, 8).

Description: A use of 'personnal' for 'personal'.
Commentary: See 17:43:51.

nonevent([mv, ' ', t o, ' ', jim, ' ', personal, (/), persmail, (/), tojim], reference, home, to, ome, tuesday, 21, january, 1992, 8, 17, 32).

Description: A use of 'to jim' for 'tojim'.
Commentary: See 17:43:51.

nonevent([mv, ' ', inquireout, ' ', home, (/), inquireout], reference, busout, home, busout, tuesday, 21, january, 1992, 18, 23, 26).

Description: This error again arose out of the misconception that paths begin from the 'home' directory.
Commentary: See 18:12:57.

nonevent([mv, ' ', inquireout, ' ', scrap, (/), inquireout], reference, busout, scrap, busout, tuesday, 21, january, 1992, 18, 23, 46).

Description: Ditto.
Commentary: See 18:12:57.

nonevent([mkdir, ' ', busaddr], duplication, business, busaddr, business, tuesday, 21, january, 1992, 18, 29, 4).

Description: An attempt to create a directory 'busaddr' when one already existed. On encountering this error the user exclaimed "Where?" and on finding the offending item "...I forgot to take that directory out."
Commentary: See 17:44:22.

nonevent([mv, ' ', busmail, (/), business, (/), home, (/), complaintin], reference, home, complaintin, busmailin, tuesday, 21, january, 1992, 18, 32, 25).

Description: Another instance of the 'default' misconception.
Commentary: See 18:12:57.

nonevent([mv, ' ', busmail, (/), business, (/), home, (/), scrap, (/), complaintin], reference, scrap, complaintin, busmailin, tuesday, 21, january, 1992, 18, 32, 53).

Description: Ditto.
Commentary: See 18:12:57.

nonevent([mv, ' ', custaddr, ' ', home, (/), business, (/), busaddr], reference, scrap, custaddr, scrap, tuesday, 21, january, 1992, 18, 37, 16).

Description: Another 'please close the window' example. The file 'persaddr' had been moved and the move was attempted again.
Commentary: See 17:53:30.

nonevent([rmdir, ' ', persaddr], reference, addresses, persaddr, addresses, tuesday, 21, january, 1992, 18, 38, 50).

Description: This time, the directory 'persaddr', which had been deleted, was 're-deleted'.
Commentary: See 17:53:30.

nonevent([cd, ' ', busmail], reference, busmail, busmail, busmail, uesday, 21, january, 1992, 18, 41, 17).

Description: 'busmail' used for 'busmailin'
Commentary: See 17:43:51.

nonevent([cd, ' ', permail], reference, personal, permail, personal, tuesday, 21, january, 1992, 18, 47, 51).

Description: 'permail' used for 'persmail'.
Commentary: See 17:43:51.

Summary: There were twenty six context errors in this user's session log. On two occasions, the inferences drawn from the diagnostic criteria were inappropriate to the user's particular situation, in that some of the information germane to the error would not have been provided or that some misleading information would have been given. On seven occasions, the user would have received information relating to system rules, as well as relevant state information, when the misconception was state-based. On the other seventeen occasions, the misconception diagnosed via the stated criteria broadly matched that apparent in the analysis of the recorded protocol and log file.

S3

nonevent([ls, ' ', addresses, (/), persaddr], read, addresses, persaddr, home, monday, 10, february, 1992, 19, 20, 53).

Description: Here, an attempt was made to list the read-protected directory 'persaddr'. The user was unaware of this protection, which had been set prior to the session.
Commentary: The most logically proximate possible world compatible with this command is one in which the directory had not been read-protected. A state misconception of read-permission on the directory would have been diagnosed on the basis of a previous action setting this protection, recorded in the interaction log, having been forgotten. The state information relevant to remedying this misconception would have been displayed, ie: that the directory had been read protected and when. This interpretation assumes that the user had set the protection on the directory himself, as the record is in his log. If there were no such record, other interpretations are available, for example, that the user is unaware of the presupposition of read-permission on the *ls* command. In this case, that information would also be imparted.

nonevent([mkdir, ' ', personal], duplication, home, personal, home, monday, 10, february, 1992, 19, 31, 40).

Description: Here, the user was unaware of the presence of a directory with the same name as that which he was trying to create (no duplicate names are allowed in the structure).
Commentary: Again, the log provides evidence that the prior creation of this directory had been forgotten, supporting a belief in the possible world

where this had not occurred. A state misconception is therefore indicated. The absence of such evidence would allow an interpretation of functional misconception regarding the duplication restrictions of the system. Information presented then would concern the existence and provenance of the original item and the presupposition of non-duplication attaching to the *mkdir* command.

nonevent([mv, ' ', persout, (/), frommary, ' ', mail, (/), home, (/), temp, (/), frommary], reference, persout, frommary, personal, monday, 10, february, 1992, 19, 35, 29).

Description: Here, an attempt was made to access a file in the wrong directory. 'frommary' was not in 'persout' but in 'persin' at this point.
Commentary: This was obviously a state misconception and, as the most logically proximate possible world is one in which 'frommary' is in 'persout', this would be the inference here and state information regarding the location of 'frommary' would be indicated.

nonevent([mv, ' ', persin, (/), frommary, ' ', mail, (/), home, (/), temp, (/), frommary], reference, home, temp, personal, monday, 10, february, 1992, 19, 36, 20).

Description: Here, the file 'temp' was assumed to be a directory and placed in the destination path. In fact, this fact had emerged earlier and the '"temp" is a file' message was still visible on the display, exemplifying what has been said regarding users' interests effectively filtering out available, relevant information and indicating the importance of timing in delivering information.
Commentary: The logically closest possible world compatible with this command is one in which 'temp' is a directory, rather than one in which files have children and so a state misconception concerning this fact would be diagnosed. If a directory of that name had been deleted at some point prior to the creation of the file 'temp', this would add weight to the inference, in the form of evidence for psychological proximity.

nonevent([mkdir, ' ', personal, (/), persaddr], duplication, personal, persaddr, home, monday, 10, february, 1992, 19, 40, 42).

Description: An attempt to duplicate the directory 'persaddr'.
Commentary: See 7:31:40.

nonevent([mv, ' ', tomary, ' ', personal, (/), persmail], reference, home, tomary, home, monday, 10, february, 1992, 19, 46, 45).

Description: Here 'tomary' was typed for 'frommary'.
Commentary: It is unlikely that a spell checking algorithm would catch

this error and so it would be treated as a context error. In this case, there is no evidence for the psychologically proximity of any particular possible world but logically, the closest world is one in which a file called 'tomary' exists in the home directory and so this would be the chosen inference. The information that no file of that name exists in that location would be given, prompting the user to check the file name. This is no more than the standard UNIX response, of course, but it is, in this instance, all that is required or possible.

nonevent([mkdir, ' ', business], duplication, home, business, home, monday, 10, february, 1992, 19, 49, 11).

Description: An attempt to duplicate the directory 'business'.
Commentary: See 7:31:40.

nonevent([mv, ' ', mail, (/), business, (/), in, (/), complaintin], reference, business, in, home, monday, 10, february, 1992, 19, 50, 20).

Description: 'in' was used for 'busin'.
Commentary: See 19:46:45.

nonevent([mv, ' ', mail, (/), business, (/), busout, (/), enqout, ' ', enqout], reference, busout, enqout, home, monday, 10, february, 1992, 19, 52, 46).

Description: 'enqout' was used for 'enquirout'.
Commentary: See 19:46:45.

nonevent([mv, ' ', mail, (/), business, (/), busout, (/), enquireout, ' ', enquireout], reference, busout, enquireout, home, monday, 10, february, 1992, 19, 53, 29).

Description: 'enquireout' was used for 'enquirout'.
Commentary: See 19:46:45, although a spelling checker would probably catch this as a lexical error.

nonevent([rmdir, ' ', busaddr], reference, home, busaddr, home, monday, 10, february, 1992, 19, 56, 9).

Description: Here, the user forgot to specify the path and gave only the target name. Thus the directory 'busaddr' was not found in the 'home' directory.
Commentary: There is no evidence here for the prior existence of a directory of that name in 'home'. Neither is there any to support the idea that a directory change (from 'addresses', which contains 'busaddr') has

been forgotten. Logically, the closest possible world, in state terms, is one in which the current working directory is 'addresses'. This would be an incorrect diagnosis of the user's system state model, confirming that logical proximity alone is not a reliable source of evidence. As has been indicated elsewhere, such cases would necessitate the provision of relevant state and functional information, in the form of the identity of the *cwd*, the location of 'busaddr' and a statement of the presuppositions governing the access of items relative to the *cwd*.

nonevent([rmdir, ' ', addresses, (/), busaddr], nonempty, nul, nul, home, monday, 10, february, 1992, 19, 56, 58).

Description: This was an attempt to remove a non-empty directory. The user thought that he had previously cleared this directory.
Commentary: Obviously, failure to perform an action cannot be recorded or used as evidence in determining a user's misconception. There will also be no unique single event establishing the occupied state of a directory, except where just one item has been introduced into it. In its lifetime, a directory's contents may change a great many times and, consequently, there is no obvious psychological relation between the events which effect these changes and the user's current system state model and thus no clear way of establishing psychological proximity of alternative possible worlds. However, the only possible worlds compatible with the command are ones in which the directory is empty and ones where the precondition of emptiness on the *rmdir* command does not prevail. It seems clear then that information regarding the non-emptiness of the directory and the presupposition or precondition of emptiness would be sufficient in dealing with this particular kind of context error.

nonevent([mv, ' ', custaddr, ' ', busaddr], duplication, business, busaddr, home, monday, 10, february, 1992, 19, 58, 57).

Description: This is an interesting case, again involving the ambiguity of the *mv* command. The user was attempting to move the file 'custaddr' into the directory 'busaddr' but failed to give the path required which was 'business/busaddr'. This is interpreted in the rules governing UNIX as an attempt to rename the file 'custaddr' to 'busaddr' and, since there was a directory of that name present, the command failed. Because the user thought that he was moving, rather than renaming, an item, the consequent error message concerning duplication was very confusing to him.
Commentary: To some extent, this situation arises from the ambiguity in the UNIX *mv* command. The possible worlds approach would give a much broader interpretation to this command instead of assuming an attempted renaming. Possible worlds compatible with the command are ones in which the directory 'busaddr' does not exist, ones in which it resides in the 'home' directory, ones in which 'business' is the *cwd* and so on. The proliferation of interpretations again arises form the inherent ambiguity of the

rename, then the logically closest possible world would be one in which 'busaddr' was a child of 'home', there being no evidence for psychological proximity. This would lead to state information being given about the *cwd*, the location of 'busaddr' and the preconditions relating to access, which would at least have been adequate in addressing the user's problem.

nonevent([mv, ' ', enquirout, ' ', business, (/), busmail, (/), busout, (/), inquireout], reference, busmail, busout, home, monday, 10, february, 1992, 20, 6, 9).

Description: Here, the name 'busout' was used for 'busmailout'.
Commentary: See 7:46:45.

nonevent([rmdir, ' ', mail], reference, mail, mail, mail, monday, 10, february, 1992, 20, 8, 31).

Description: The user forgot which directory was the *cwd* and tried to access the *cwd* from within itself.
Commentary: Any causal antecedent action, such as a directory change from the parent or child of the *cwd*, would indicate a system state misconception and state information would be provided. A lack of such evidence for psychological proximity would also prompt the provision of the relevant information regarding the violated preconditions operating on the command, eg: that the directory being removed must be a descendant of the *cwd*.

nonevent([cd, ' ', personal, (/), persmail], execute, personal, persmail, home, monday, 10, february, 1992, 20, 15, 32).

Description: Here, the attempt to make 'persmail' the *cwd* failed because the user had previously removed execute permission from it. In this case, the 'permission denied' error message was sufficient to enable the user to recover. However, it did take time for the 'penny to drop' as to the cause of the problem.
Commentary: A system state misconception about the mode setting on the directory would be diagnosed on the basis of the evidence of psychological proximity of that possible world, in the form of the (assumed forgotten) prior *chmod* command. It is often useful for users, in structuring their task, as well as understanding the functioning of the system, to know why something has occurred as well as knowing what has occurred.

Summary: There were eighteen context errors in this user's session log. On one occasions, the inferences drawn from the diagnostic criteria were inappropriate to the user's particular situation, in that some of the information germane to the error would not have been provided or that some misleading information would have been given. On three occasions,

as relevant state information, when the misconception was state-based. On the other fourteen occasions, the misconception diagnosed via the stated criteria broadly matched that apparent in the analysis of the recorded protocol and log file.

S4

nonevent([cd, ' ', business], reference, home, business, home, tuesday, 11, february, 1992, 18, 36, 36).

Description: The subject attempted to change directory from 'home' to 'business' without supplying the correct path. She was expecting the command 'cd' to find the directory for her, which amounts to a functional misconception concerning the rules for accessing the directory structure.
Commentary: The particular precondition violated is that an item in the path must be a parent or child of its predecessor (the *cwd* being assumed to be the path start). There is no evidence to support the idea that the subject held any particular state misconception and so both forms of correction are indicated. Assuming that the information on relevant system states (locations of *cwd* and target items) and rules for path construction could be assimilated, the problem could be overcome.

nonevent([cd, ' ', address], reference, mailin, address, mailin, tuesday, 11, february, 1992, 18, 39, 15).

Description: 'address' used for 'addresses'.
Commentary: Spell checking could handle this error if it is treatable as a naming error. If the user rejected this interpretation, then the misconception that a directory called 'address' existed would be dealt with on the basis of the rule precondition that a directory must exist before it can be made the *cwd* and the state information that no such directory exists.

nonevent([cd, ' ', addresses], reference, mailin, addresses, mailin, tuesday, 11, february, 1992, 18, 39, 23).

Description: Here, the subject forgot a previous directory change to a child and so effectively attempted to directly access a sibling directory.
Commentary: The previous directory change provides evidence in the log that supports this interpretation in that the command would have succeeded had that action not occurred. Current state information and its provenance would seem to be adequate to correct matters in this case.

nonevent([cd, ' ', temp], logical, nul, nul, home, tuesday, 11, february, 1992, 18, 44, 20).

attempt to make a file the *cwd*, which is never possible. It could also be viewed as a context error, if duplicate names were allowed and 'temp' was seen as a non-existent or deleted/renamed directory, or as a compound error, where both the preconditions of existence and duplication were being violated. In the event, the subject appeared not to know whether 'temp' was a file or directory, rather than holding a functional misconception concerning directory changes.
Commentary: There being no evidence in the log to support the idea that the subject believed there to be a directory called 'temp', the fact that 'temp' was a file, along with the (violated) precondition that the target item must be a directory would be the information, indicated by the given criteria, which would be provided.

nonevent([cd, ' ', persaddr], execute, addresses, persaddr, addresses, tuesday, 11, february, 1992, 18, 46, 14).

Description: Here, the subject was unaware of the execute protection on the directory 'persaddr' but quickly realised that she needed to change its mode setting.
Commentary: As with other subjects, the action setting execute protection on 'persaddr' was in the log but had not been performed by this subject. Normally, its presence would be taken as evidence for a state misconception, based on the idea that this action had been performed by the subject and subsequently forgotten. This would lead to information regarding the protected state of the directory being presented, along with a reminder of the action which achieved this.

nonevent([cd, ' ', persaddr], execute, addresses, persaddr, addresses, tuesday, 11, february, 1992, 18, 50, 7).

Description: Here, the attempt to *cd* to 'persaddr' failed again. This was because the subject had (correctly) decided that the mode setting needed to be changed but had then changed it to 400, which does not remove execute protection.
Commentary: There is obviously an error at the planning level here, which could conceivably been detected and dealt with at the mode setting stage. The stated criteria would lead to the diagnosis, made on the basis of the mode setting action in the log, that the subject had forgotten this action and information concerning the mode state of 'persaddr' and its provenance would be provided. While this might be helpful, it does not address the actual misconception regarding which mode settings grant execute permission. Clearly, the criteria do not, in themselves, adequately handle this situation. However, as the approach is intended to operate in conjunction with plan and spelling based systems, this is not necessarily a major drawback.

addresses, tuesday, 11, february, 1992, 18, 50, 44).

Description: An attempt to change directory to what was already the *cwd*. Here, the subject appeared not to understand that if the *cd* command failed (to change directory from 'addresses' to 'persaddr'), its intended effects would not be realised. She therefore expected to have to "...go back" to the parent 'addresses' (actually the *cwd*) in order to be able to reset the mode.
Commentary: This highlights the notion of misconceptions concerning general principles of system functionality, as opposed to specific implications for particular commands. In this instance, our criteria might give a diagnosis of state misconception concerning the identity of the *cwd*, based on a logged directory change and perhaps a functional misconception regarding access preconditions on directory changes. However, this situation raises the possibility of reasoning about the user's understanding of general principles of system function. For example, that any failed command leaves the system state unchanged. Using failed commands from the interaction log could therefore also be used as a source of evidence to support hypotheses of user misconception of a more general kind. For example, a number of erroneous commands, compatible with system states which were the intended target of the preceding (contextually) erroneous command, could indicate a general misconception of this type. Having said that, no attempt has been made to accommodate this level of misconception but the approach adopted would have handled the 'local' misconception, had the last diectory chage been from the child or parent of the target of this command.

nonevent([cd, ' ', personal], execute, mail, personal, mail, tuesday, 11, february, 1992, 18, 57, 41).

Description: Here, the subject was unaware of the execute protection on the directory 'personal'.
Commentary: See 18:46:14.

nonevent([cd, ' ', mail], reference, mail, mail, mail, tuesday, 11, february, 1992, 18, 57, 47).

Description: By this time, the subject had developed the strategy of using 'cd (x)' to check if x was the *cwd*. She took the UNIX style 'x not found' error message to be confirmation.
Commentary: Any log evidence would have prompted a 'genuine' confirmation of system state regarding the *cwd* and the target, whereas a lack of it would also engender the provision of information regarding the precondition regarding directory changes. The subjects strategy would therefore prove more reliable than it was under the prevailing circumstances, as can be seen later in the session.

nonevent([cd, ' ', fromeric], logical, nul, nul, mail, tuesday, 11, february, 1992, 19, 0, 1).

Description: This was an attempt to 'cd' to a file. The subject developed a strategy of using the 'cd' command to check if an item was a file or directory.
Commentary: See 18:44:20.

nonevent([ls], read, mail, business, business, tuesday, 11, february, 1992, 19, 0, 21).

Description: The subject was unaware of the read-protection on the directory 'business'.
Commentary: See 6:46:14.

nonevent([chmod, ' ', 700, ' ', business], reference, business, business, business, tuesday, 11, february, 1992, 19, 1, 2).

Description: An attempt to reset the mode on 'business' from within it. The subject was apparently aware of the identity of the *cwd* but there seemed to be some confusion, bound up with the default status given to the *cwd* for the 'ls' command, resulting in a misconception concerning the specification of targets which happen to be the *cwd*.
Commentary: The previous directory change to 'business' would be taken as evidence for a state misconception regarding the identity of the *cwd* and information relating to these facts would be presented, under the given criteria. This might help indirectly but does not address the subject's functional misconception. The inconsistency in the way that UNIX treats default arguments is implicated in the probable cause of this misconception but information regarding the preconditions governing access to the structure is also required in this instance, which the stated criteria fail to recommend.

nonevent([cd, ' ', personal], execute, mail, personal, mail, tuesday, 11, february, 1992, 19, 2, 48).

Description: The subject was unaware of the execute protection on 'mail' and tried to make it the *cwd*.
Commentary: See 18:46:14.

nonevent([cd, ' ', mail], reference, mail, mail, mail, tuesday, 11, february, 1992, 19, 2, 59).

Description: An attempt to make the *cwd* the *cwd*.
Commentary: See 18:50:44.

nonevent([cd, ' ', frommary], logical, nul, nul, persin, tuesday, 11, february, 1992, 19, 6, 46).

Description: An attempt to make a file the *cwd.*
Commentary: See 18:44:20.

nonevent([cd, ' ', fromjim], logical, nul, nul, persin, tuesday, 11, february, 1992, 19, 6, 59).

Description: An attempt to make a file the *cwd.*
Commentary: See 18:44:20.

nonevent([cd, ' ', addout], logical, nul, nul, busout, tuesday, 11, february, 1992, 19, 9, 44).

Description: An attempt to make a file the *cwd.*
Commentary: See 18:44:20.

nonevent([cd, ' ', enquireout], reference, busout, enquireout, busout, tuesday, 11, february, 1992, 19, 10, 2).

Description: 'enquireout' used for 'enquirout'. A compound error also, since enquirout was a file.
Commentary: See 18:39:15.

nonevent([cd, ' ', busout], reference, busout, busout, busout, tuesday, 11, february, 1992, 19, 10, 20).

Description: When this error occurred, the subject exclaimed "I'm in 'busout'...I coudn't have been in enquirout...I spelled it wrong." There are two things to note here. Firstly, the hypothesis mentioned earlier, that the subject did not understand that an error message such as 'file not found' means that the command has failed, seems to be confirmed by this protocol. Secondly, the 'cd' command was being used to check the identity of the *cwd* (see 18:57:47).
Commentary: See 18:50:44.

nonevent([cd, ' ', enquirout], logical, nul, nul, busout, tuesday, 11, february, 1992, 19, 10, 57).

Description: The second error component of 19:10:2 comes home to roost now that the spelling error is resolved. These should be treated concurrently in a cooperative dialogue.
Commentary: See 18:44:20.

busout, busout, tuesday, 11, february, 1992, 19, 11, 59).

Description: Here, the *cwd* was placed at the head of the path. The subject knew the identity of the *cwd* and so this appears to be a functional misconception regarding path construction.
Commentary: See 19:1:2.

nonevent([cd, ' ', complaintin], logical, nul, nul, busin, tuesday, 11, february, 1992, 19, 13, 2).

Description: An attempt to make a file the *cwd*.
Commentary: See 18:44:20.

nonevent([cd, ' ', addresses, (/), persaddr, (/), joneses], logical, nul, nul, home, tuesday, 11, february, 1992, 19, 14, 17).

Description: Ditto.
Commentary: See 18:44:20.

nonevent([cd, ' ', smiths], logical, nul, nul, home, tuesday, 11, february, 1992, 19, 14, 45).

Description: Ditto.
Commentary: See 18:44:20.

nonevent([cd, ' ', custaddr], logical, nul, nul, busaddr, tuesday, 11, february, 1992, 19, 16, 0).

Description: Ditto.
Commentary: See 18:44:20.

nonevent([cd, ' ', suppaddr], logical, nul, nul, busaddr, tuesday, 11, february, 1992, 19, 16, 19).

Description: Ditto.
Commentary: See 18:44:20.

nonevent([mv, ' ', busaddr, ' ', addresses, (/), root, (/), mailin], reference, busaddr, busaddr, busaddr, tuesday, 11, february, 1992, 19, 24, 32).

Description: Use of 'busaddr' for 'custaddr' and 'root' for 'home'. Both of these appeared to be 'slips'. There are also the compound component error in that 'busaddr' is a directory and cannot be moved.

procedures but the second would not. No evidence was available in the log to support a relevant state misconception hypothesis and so relevant state and functional information would be generated regarding the 'root' error. If spell checking did not recommend the target 'custaddr', then the fact that 'busaddr' was a directory would be the major obstacle, rather than its location relative to the *cwd*. This demonstrates the need to rank errors with regard to their precedence and also to take account of when an error obviates the need to pursue lower ranking compound component errors. For example, logical errors would outrank and obviate the pursuance of reference errors. If directories cannot be moved, then there is no point in telling the user where to find the directory for a *mv* operation.

nonevent([mv, ' ', custaddr, ' ', addresses, (/), root, (/), mailin], reference, addresses, root, busaddr, tuesday, 11, february, 1992, 19, 25, 28).

Description: Repeat of 'root' for 'home' error.
Commentary: See 19:24:32.

nonevent([cd, ' ', persaddr], reference, home, persaddr, home, tuesday, 11, february, 1992, 19, 26, 35).

Description: The subject thought 'persaddr' to be in the directory 'home' when it was in 'addresses', a child of 'home'. The error was quickly recognised from the 'x not found' error message.
Commentary: No evidence existed in the log to support a hypothesis of this misconception above any others but the relevant state information would include the identity of the *cwd* and the relative location of 'persaddr'. Presented along with preconditions concerning path construction, this would seem to be pertinent to the situation.

nonevent([mv, ' ', fromeric, ' ', mail, (/), home, (/), mailin], reference, mail, mail, mail, tuesday, 11, february, 1992, 19, 29, 6).

Description: The *cwd* was placed at the head of the path.
Commentary: See 19:11:59.

nonevent([rmdir, ' ', addresses], nonempty, nul, nul, home, tuesday, 11, february, 1992, 19, 40, 45).

Description: Here, the subject attempted to remove a non-empty directory.
Commentary: Any particular action creating contents in the directory could be taken as evidence for the hypothesis that a misconception that the directory was empty was held. In that case, only relevant state and

in that the creation of items in a directory does not imply knowledge about directory deletion and its preconditions, any more than it implies knowledge of the system rules regarding duplication. The question of the implications for knowledge that the various kinds of action have needs to be addressed in order to strengthen the basis of evidence for selecting from alternative hypotheses of user misconception.

nonevent([rmdir, ' ', business, (/), busin], reference, home, business, home, tuesday, 11, february, 1992, 19, 43, 50).

Description: The subject had been specifying the intended path correctly prior to this error and then missed out the 'mail' path component.
Commentary: One would think that no detailed help information is really necessary here but see the next entry. In any event, the situation would be treated as if it were not a 'slip' and relevant state and functional information provided. It might be possible to distinguish between 'slips' and misconceptions by looking for recent successful commands in the log which have the same path specification. However, as the directory structure is inherently unstable, it is not obvious that this would be a trivial task or a reliable method.

nonevent([rmdir, ' ', busin], reference, home, busin, home, tuesday, 11, february, 1992, 19, 44, 13).

Description: Here, the subject had wrongly interpreted the cause of the last error as being the fact the 'business' was the *cwd* and that the error was the specification of 'business' at the start of the path. "I'm in 'business'." She thus tried to access 'busin' directly as a child of what she thought was the *cwd*. When the true identity of the *cwd* was realised, she said "I'm home... how'd I get there?"
Commentary: This demonstrates the importance of the precise handling of errors. An error may be caused by a simple 'slip' but its interpretation by the user is another matter. Error messages need to be as precise and accurate as possible, in order to avoid this kind of misinterpretation. For this error, state and functional information would be presented if the stated criteria were followed.

Summary: There were thirty three context errors in this user's session log. On five occasions, the inferences drawn from the diagnostic criteria were inappropriate to the user's particular situation, in that some of the information germane to the error would not have been provided or that some misleading information would have been given. On nine occasions, the user would have received information relating to system rules, as well as relevant state information, when the misconception was state-based. On the other nineteen occasions, the misconception diagnosed via the stated criteria broadly matched that apparent in the analysis of the recorded

protocol and log file.

General Summary: There were one hundred and twelve context errors in all users' session logs. On eleven occasions, the inferences drawn from the diagnostic criteria were inappropriate to the user's particular situation, in that some of the information germane to the error would not have been provided or that some misleading information would have been given. On twenty two occasions, the user would have received information relating to system rules, as well as relevant state information, when the misconception was state-based. On the other seventy occasions, the misconception diagnosed via the stated criteria broadly matched that apparent in the analysis of the recorded protocol and log file. The overall success rate in approximate percentage terms is thus respectively: fail = 9%, verbose = 21%, correct = 70%.