ENGINEERING THE HUMAN-COMPUTER INTERFACE

DOW

ENGINEERING THE HUMAN–COMPUTER INTERFACE

Edited by
Andy Downton
Department of Electronic Systems Engineering
University of Essex

McGRAW-HILL BOOK COMPANY

London · New York · St Louis · San Francisco · Auckland
Bogotá · Caracas · Hamburg · Lisbon · Madrid · Mexico · Milan
Montreal · New Delhi · Panama · Paris · San Juan · São Paulo
Singapore · Sydney · Tokyo · Toronto

QA
76.9
.H85
E53
1991

Published by
McGRAW-HILL Book Company (UK) Limited
SHOPPENHANGERS ROAD · MAIDENHEAD · BERKSHIRE · ENGLAND
TEL: 0628 23432
FAX: 0628 770224

British Library Cataloguing in Publication Data

Engineering the human–computer interface. – (Essex series in
telecommunication and information systems).
1. Man. Interactions with computer systems
I. Downton, Andy II. Series
004.019

ISBN 0-07-707321-5

Library of Congress Cataloging-in-Publication Data

Engineering the human–computer interface/editor, Andy Downton.
p. cm.—(Essex series in telecommunication and information systems)
Includes bibliographical references and index.
ISBN 0-07-707321-5
1. Human–computer interaction. 2. User interfaces (Computer systems)
I. Downton, A. C. II. Series.
QA76.9.H85E53 1991
005.1—dc20 90-24975 CIP

1234 CUP 94321

Typeset by Computape (Pickering) Ltd, Pickering, North Yorkshire
and printed and bound in Great Britain at the University Press, Cambridge

4 Dialogue styles: basic techniques and guidelines

ANDY DOWNTON

4.1 Introduction

4.1.1 History

Computer dialogue styles have been transformed over the last twenty-five years by the introduction of first minicomputers and then microprocessors. The introduction of the first minicomputer (the DEC PDP8), though primarily viewed as a breakthrough in dedicated real-time computing, also represented a landmark in human–computer interaction. Whereas mainframes had typically been run in a batch mode, minicomputers were used interactively, introducing for the first time the need to provide a programmed interface between the user and the computer.

Since the early 1970s the microprocessor has become a ubiquitous part of most electronic systems. The widespread availability of microcomputers based upon microprocessor technology has accelerated the trends first begun by minicomputers, by putting significant raw computing power in the hands of inexperienced users for the first time. System performance in this case is clearly dependent upon maximizing the *usability* of the computer rather than using its processing power efficiently, and thus much of the processing power of personal computers is now expended not in processing data but in facilitating communication with the user. Even where the basic level of communication (via the operating system) is rather traditional (typically, a terse command line dialogue where the user is expected to know the names of commands acceptable to the machine), applications such as word processors, database managers, spreadsheets, etc., overlay this with simpler menu-selection command mechanisms.

Increasingly, machines like the Apple Macintosh model their user interface upon familiar metaphors such as the office desk, and use a pointer device (the mouse) to select and manipulate objects. This example illustrates the close interrelationship between the capabilities of the technology and the style of the user interface, because the WIMP (variously an acronym for windows, icons, mouse, pull-down menus, or windows, icons, menus, pointers) style of user interaction was fully explored and evaluated in the early 1970s at the Xerox Palo Alto Research Center using dedicated minicomputer workstations. In spite of their apparent visual appeal and friendliness, however, the software complexity of such graphics-oriented interfaces is high, and thus it was not until 16- and 32-bit microprocessors were introduced that sufficient processing power was available to enable such systems to become practicable at low cost.

4.1.2 Application areas

Although the archetypal concept of a human–computer dialogue conjures up images of the user sitting in front of a terminal interacting with a keyboard and video display, the expression admits a much wider range of interpretations than this. Many familiar home, office and entertainment systems now contain embedded microcontrollers with which dialogues of a sort are conducted.

A minimal example of such a system is the familiar digital watch (Figure 4.1). Typically, this incorporates, in addition to clock, day and date functions, a stopwatch with lap timer, a presettable countdown timer, and one or more alarms. It may seem natural and obvious with hindsight that all these functions can be readily controlled and initialized with only four pushbuttons, but in the absence of a known solution the problem would be much more taxing. Furthermore, given only the specification of the required functionality, and the availability of a single four-bit input, it is clear that many possible solutions could be generated, varying widely in simplicity, consistency, flexibility and ease of use.

Other examples of domestic equipment containing a microcontroller are now commonplace: video cassette recorders, teletext TV sets, microwave and conventional ovens, and automatic washing machines

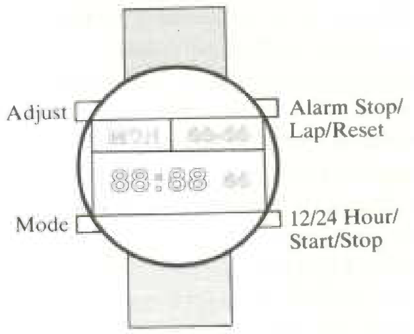


Figure 4.1 The digital watch—a minimal example of human–computer dialogue

all enable the mode of operation to be programmed using a rudimentary pushbutton dialogue, typically augmented by feedback from a display panel. Such dialogues can generally be categorized as simple forms of menu selection, and are readily formalized using state transition diagram techniques (see Sections 6.5 and 7.5.1).

In the office, business and industrial environment, too, examples of embedded systems abound: photocopiers, intelligent VDU terminals, telephones, PABXs, machine tools and industrial process controllers are all controlled using dialogues of greater or lesser sophistication. Word processors, video games and personal computers running databases, spreadsheets, graphics and communication packages represent the conventional concept of computers and provide a more versatile and flexible potential for human–computer interaction. The availability of video displays, keyboards and graphic input devices opens up opportunities for much richer, more powerful and more extensive dialogues using menus, form-filling, command languages and /or direct manipulation to specify commands.

4.1.3 Dialogue design objectives

In all these application areas, the concept of developing the engineering design using top-down techniques starting from systems analysis and specification is well understood: what is often overlooked is that exactly the same process of step-wise refinement should and can be applied to the process of interface and dialogue design. Instead, the user–interface design is often based upon unsupported implicit assumptions made by the design engineer about the nature of the task, the user's model of the task, and the characteristics of the users.

As part of the design process, the designer needs to be armed with a comprehensive understanding of the types of interfaces and dialogue styles available, their appropriateness for different categories of users, and their system implications in terms of display, input device and processing power requirements. The objective of this chapter is to provide an overview and taxonomy of different dialogue styles; Chapters 6 and 7 then discuss some of the techniques and tools available to support different human–computer dialogue models and illustrate the models and dialogue styles using some example research systems. Chapter 5, on knowledge analysis of tasks, is concerned with ways of eliciting knowledge of user models and methods which can provide a specification for the required dialogue.

4.2 Dialogue properties

Before reviewing specific styles of dialogue in detail, it is worth while considering some desirable generic characteristics of these dialogues. Alison Kidd (1982) cites five important properties:

- initiative;
- flexibility;
- complexity;
- power;
- information load;

to which can also be added:

- consistency;
- feedback;
- observability;
- controllability;
- efficiency;
- balance.

These properties are considered in more detail below.

4.2.1 Initiative

Initiative is the most fundamental property of any dialogue, since it defines the overall style of communication and thus, to a large extent, the type of user for whom the system is intended. The two most common styles are computer-initiated and user-initiated. In *computer-initiated* dialogues, the user responds to explicit prompts from the computer to input commands or command parameters: typically, (s)he will be presented with a series of options from which a selection must be made (menu selection), or a number of boxes into which parameters must be inserted (form-filling), or questions to which answers must be specified in a more or less restricted way (for example yes/no vs. natural language). The key characteristic is that the dialogue consists of a closed set of options defined by the computer.

By contrast, *user-initiated* dialogues are open-ended in nature: the user is expected to know a valid set of command words and their allowable syntax, and possibly a semantic structure for the system if operation in several different modes is possible. A typical example of a user-initiated dialogue would be the command language of a computer operating system; superimposed upon this could be the semantic structure of a variety of other command language-driven programs such as editors and debuggers, each with their own distinct style of dialogue.

Dialogues need not be purely computer-initiated or user-initiated. *Variable-initiative* dialogues are becoming more common, particularly in systems intended for wide ranges of users. In many current systems, the level of user initiative required is chosen by the users themselves (for example by selecting between user-initiated and computer-initiated modes), but *adaptive* dialogue styles are also becoming more common.

At their simplest, adaptive dialogue systems adjust the level of assistance by monitoring, for example, the delay in the user inputting a command or parameter: when a prespecified limit is exceeded, additional assistance on options available is displayed. Alternatively, when parameters are omitted from a command line, they can be prompted for by the system. In either case, a user-initiated dialogue is partially transformed to a computer-initiated style. Conversely, a computer-initiated menu selection system can be adapted to a user-initiated mode by allowing selections from subsequent menus to be given without waiting for the intervening menu(s) to be displayed (see Section 4.12 for an example of this). More complex techniques, for example the use of knowledge-based systems to predict users' capabilities by monitoring the content of their interactions (adaptive intelligent dialogues—see Chapter 7) remain a research problem at present.

Finally, we can distinguish *mixed-initiative* dialogues as being common in natural-language communication with computers: here, the free format of both input and output to the system allows the possibility of either the computer or the user leading the dialogue at different times.

4.2.2 Flexibility

A flexible system is one in which a particular objective can be achieved in a variety of different ways. This is not simply a matter of providing a very large command repertoire in the hope of covering every possibility, but should follow from an analysis of users' models of the activity. The objective is then to map the system onto representative users' models, rather than to force the users to work within a framework defined by the designer.

One of the less frequently recognized reasons for the success of the Apple Macintosh personal computer is that a great deal of attention has been paid in the design of its interface to this concept. Most functions can be accomplished in several different ways, corresponding to different models which the user may have of his or her activity (e.g., see Figure 4.2). Generally, users are unaware of this characteristic, since there is little incentive to investigate alternative methods of accomplishing a function once success has been achieved, but they may notice a higher success rate in trying out new functions than is common with other machines (which itself will tend to generate a positive response to the system).

Flexibility can also be conferred by providing opportunities for the user to *customize* and extend the interface to meet their own personal requirements. This capability is most commonly observed in the provision of programmable function keys on microcomputers, or programmable control code alternatives (*power keys*) to pull-down or pop-up menu selection options on WIMP interfaces.

The price paid for flexibility is in the complexity and efficiency of the

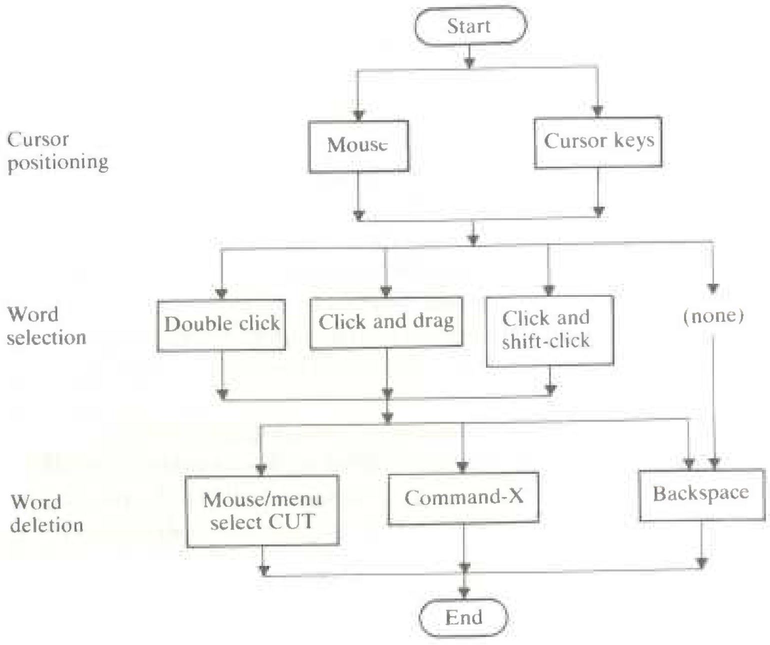


Figure 4.2 Word deletion strategies under Apple™ Macintosh MacWrite

interface software: clearly there is an overhead in providing additional functionality over and above the minimum required to perform the task. This overhead can however by minimized by careful organization of the command structure to eliminate duplication (see Section 4.2.6 below).

4.2.3 Complexity

In general, there is no benefit in making an interface more complex than necessary. Often, an apparently complex interface is in fact a symptom of a failure to analyse the structure of the interaction and organize the commands accordingly. Logical grouping is important in reinforcing the user's model of the system, and is commonly achieved by the use of *hierarchy* or *orthogonality* or both.

Hierarchy

This is the structuring of commands according to related characteristics and their relative importance. Rather than having a flat command structure requiring memorization of a large number of individual unrelated commands, the commands are grouped into a hierarchical tree, where related commands are associated in different branches of the

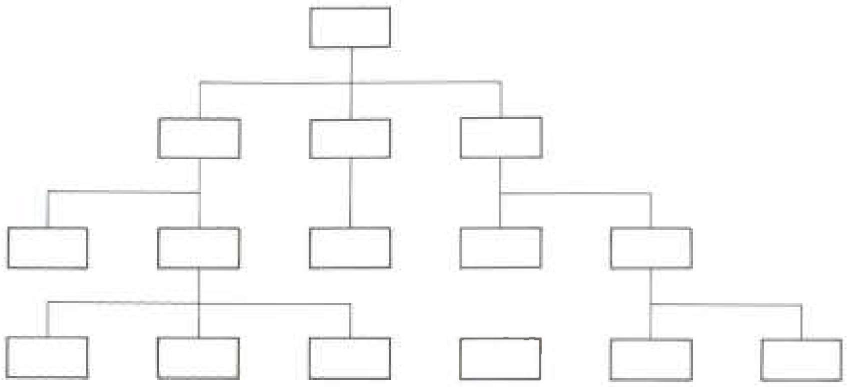


Figure 4.3 Tree structure of commands

tree (Figure 4.3). This maximizes the chance of memorizing the commands by recoding groups of commands as individual 'chunks', while at the same time simplifying decision making by offering a restricted set of options according to the current position in the tree.

Orthogonality

This is the structuring of commands according to independent characteristics (Figure 4.4). For example, specifying three independent command characteristics A, B and C, each of which is chosen from a set of 10 options, provides the capability to represent up to 1000 distinct commands, while requiring only 30 independent items (A1 ... A10, B1 ... B10, C1 ... C10) to be memorized. This technique is most commonly exploited in specifying command parameters.

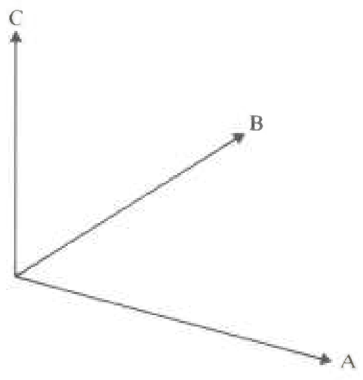


Figure 4.4 Orthogonality in command structures

4.2.4 Power

Power is defined as the amount of work accomplished by the system per user command. Users (particularly expert and experienced users) generally react positively to the availability of powerful commands, and conversely can be irritated by the pedantry of a system that requires excessive user input to achieve a specific function. Note that a requirement for powerful commands may conflict with the need for flexibility, and may affect the complexity of the system.

4.2.5 Information load

The information load which the dialogue imposes on the user in terms of both memory and decision making should be appropriate to the level of user. If too high a load is incurred, this leads to anxiety on the part of the user, which has a negative effect both on cognitive processing capability and attitude towards the system. If there is too low a load, this leads to resentment since the user sees the system as inhibiting his or her performance. Matching the information load to the user presents particular difficulties, since different loads are optimal for different categories of users, but any user who uses a system regularly will become more proficient and hence change categories over a period of time. Variable initiative dialogues provide one possible way of coping with this problem.

4.2.6 Consistency

Consistency is an important attribute in helping the user to develop his mental model of any computer system. A consistent system will encourage this by helping the user to extrapolate successfully from his or her current knowledge to explore new commands and command options. Once the user has used a particular command in one context, it is reasonable for him or her to expect that it will also work in any other context which he or she perceives as similar.

Consistency should apply to all aspects of user interface design. Commands should have a standardized syntax and parameter ordering; displays should have a consistent layout; the data entry format should be compatible and consistent with the data display format.

4.2.7 Feedback

Immediate feedback is needed for any user input. The feedback should unambiguously indicate the type of activity taking place, for example text input, mode selection, command input, pointing. It is amazing how many widely used systems fail to observe this fundamental requirement!

4.2.8 Observability

An observable system is one in which the system function is apparent and clear at a surface level, even if the underlying processing is complex. This can sometimes be difficult to achieve, particularly where a simple model of a complex underlying activity is presented to the user. Difficulties arise where the user exceeds the bounds of the model (for example due to errors) and the system is no longer able to present responses which the user can understand in terms of the model.

4.2.9 Controllability

Controllability is the converse of observability, and implies that the user is always in control of the system. For this to be true, the interface must provide means by which the user can determine:

- where he has been;
- where he is now;
- where he can go from here.

4.2.10 Efficiency

Efficiency in a closely coupled human–computer system is defined in terms of the throughput achieved by the person and the computer working together. Thus, although the efficiency of the engineering and software aspects of the system may be important if they affect the response time or display rate of the system, often the designer can afford to buy his way out of trouble (by specifying a more powerful computer if necessary), knowing that developments in technology will minimize the cost of this decision in due course. In contrast, the cost of skilled staff is increasing all the time, emphasizing the importance of using their time as effectively as possible, even at the expense of devoting a large proportion of the total available processing power to the user interface. The development and exploitation of workstations and computer-aided engineering vividly illustrate this point.

4.2.11 Balance

A basic design strategy for any human–computer system should be to subdivide the tasks in an optimum way between the person and the computer. Table 4.1 illustrates some of the relative capabilities of each. Essentially these differences reflect the complementary strengths and weaknesses of humans and computers: humans cope best with changing circumstances, uncertainty and incomplete knowledge, while computers are better suited to dealing with repetitive and routine activity, reliable

Table 4.1 Relative aptitudes of humans and computers (adapted from Shneiderman, 1987)

Human aptitudes	Computer aptitudes
Estimation	Accurate calculation
Intuition	Logical deduction
Creativity	Repetitive activity
Adaptation	Consistency
Subconscious concurrency	Multitasking
Abnormal/exceptional processing	Routine processing
Associative memory	Data storage and retrieval
Non-deterministic decision making	Deterministic decision making
Pattern recognition	Data processing
World knowledge	Domain knowledge
Error proneness	Freedom from error

storage and retrieval of data, and accurate computation of both numerical and logical functions.

At the interface between the human and the computer, the complementary aptitudes must be brought together in an appropriate way, otherwise the overall efficiency of the human–computer system will be degraded. Interface design therefore involves choices which affect not only the hardware configuration, but also the way in which the tasks are divided, and the structure of the dialogue between the person and the machine.

4.3 Human characteristics

Human characteristics, as they affect the design of systems and their user interfaces, can in some senses be viewed as an onion-like structure. At the centre is the system itself, and around this in concentric layers are the various human characteristics (Figure 4.5). Human cognition should be taken into account in designing the dialogue organization, whereas perception and motor control should respectively influence the graphical structure of the interface and the design of input devices. The dialogue style should be chosen with reference to the experience and expertise of the user. The influence on interface design of each of these characteristics is briefly reviewed below.

4.3.1 Ergonomics

The basic physical characteristics of humans provide initial criteria for judging the efficacy of human–computer interfaces, and indeed for

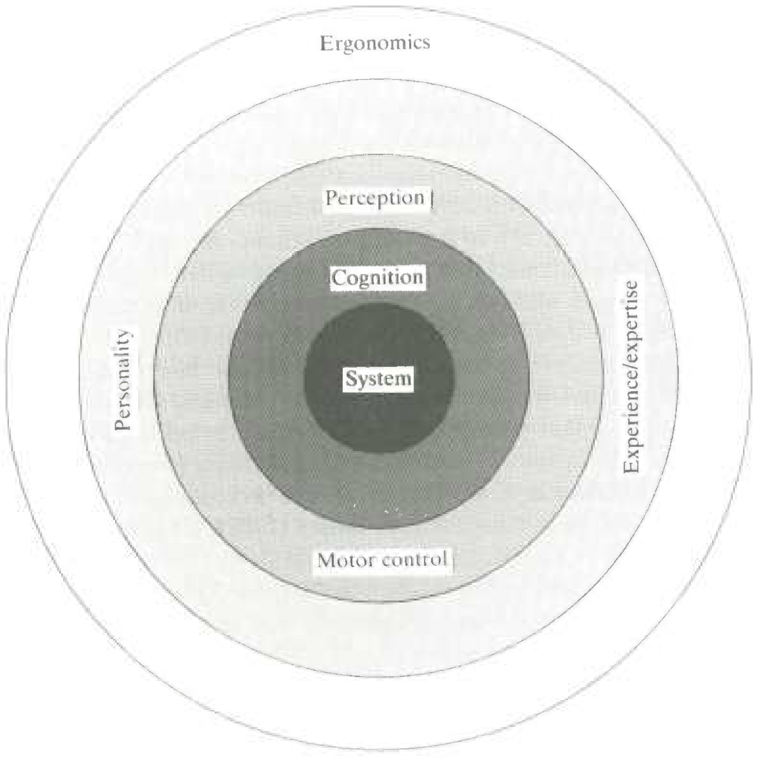


Figure 4.5 Onion-skin structure of human–computer interaction

specifying the surrounding environment (lighting, seating, table height, keyboard and screen angles, etc.). Anthropometric data concerning human dimensions and their statistical ranges and distribution are widely available (e.g., see Bailey, 1982, Chapter 5 for a review and sample tables). Although statistical averages can be calculated for any parameter, Bailey illustrates the fallacy of designing for the 'average' person by demonstrating that among a sample of 4063 men measured in a survey (Daniels and Churchill, 1952), no two were average in all 10 dimensions commonly used in clothing design. ('Average' was considered to be the middle 30 per cent of the measured range for each dimension.) A more satisfactory strategy is therefore to design for a particular percentile range, for example 5–95 per cent. In many cases (e.g., keyboards) a single design can meet this requirement, but where the range of human variation is large (e.g., seat height) then user control can be provided to allow optimization.

4.3.2 Perception, cognition and motor skills

In human–computer interface design, dynamic as well as static physical characteristics are important. Since these depend not only on human physical dimensions but also on muscular control, they are often categorized under the heading of *motor control* parameters (see Chapter 2). Parameters concerned with arm, hand and finger movement speeds are generally of most concern to interface designers, though voice control is also of fundamental importance to those in the speech-processing field, and has led to the development of complex and detailed models of the vocal tract (e.g., see Holmes, 1988).

If motor control represents the output of the human information processor, then *perception* provides the inputs. Vision and hearing are by far the most important perceptual faculties from the point of view of interface design, and the more important characteristics of each are briefly reviewed in Chapter 2. Further detail can be found in Card, Moran and Newell (1983), Chapter 2; in Bailey (1982), Chapters 4 and 7; and in Monk (1984), Chapter 1.

Cognitive models of the type developed in Chapter 2 provide the most specifically useful information and insight into user interface design. Short-term memory, long-term memory and the communication channel between them are used extensively in interactions with a computer; in closely-coupled activity between the human and computer, decision making and problem solving form the major human contribution to the dialogue.

4.3.3 Personality factors

Though less readily measured than more basic physical, perceptual, cognitive and motor characteristics, *personality* and *motivation* should also be considered in human–computer dialogue design. Characteristics such as extroversion and introversion, not to mention different behaviour patterns between sexes, can have a major impact on dialogues. Similarly, *tolerance*, *harassment*, *mood* and *fatigue* can all make the difference between ready acceptance and total rejection of a computer system. Unfortunately, these factors tend to introduce an element of instability in human–computer dialogues: for example, anxiety increases the difficulty in learning about a new computer system, which in turn reduces performance, which increases anxiety, and so on.

4.3.4 Experience and expertise

Background knowledge concerning the experience and expertise of the expected user population is important in designing the interface. *Experience* relates to the general understanding which the user has of the problem, and of computer technology in general; *expertise* implies the more specific detailed knowledge which the designer may expect the user to have in carrying out the task using the system. Clearly, neither of these factors is likely to remain static unless the system is used very infrequently; nevertheless, this situation can arise in the case of, for example, computer-based public information systems.

However, the concept of an 'expert' as someone with extensive knowledge of the system has been shown to be simplistic, at least in large systems with open learning (Draper, 1984). In a survey of use of the UNIX operating system and its screen editor, *vi*, by 94 users over 8 months, it was shown that there were no 'experts' using the system in the sense of people who used all the commands. Only 69 per cent of the total available UNIX commands were used at all over the period of the survey, and the highest individual vocabulary was 60 per cent of these. Users' expertise was idiosyncratic: 18 individuals used one or more commands not used by any other users. Thus schemes for classifying users, such as that proposed in the next section, should ideally take account of the context in which the user is working—an argument in favour of adaptive interface styles.

The same survey also showed that 'experts' were by far the heaviest users of the UNIX manuals, whereas the reverse might be expected to be the case. A possible inference from this is that expertise is not so much based on innate knowledge about the system, but is more a case of understanding how to use the system and its documentation to tackle and solve new problems. This is analogous to the skill of a librarian, whose expertise in navigating through the library's cross-referencing systems is often called upon by library users to help solve their own specialized problems. Thus expertise may be more associated with the acquisition of techniques and strategies for problem solving rather than any defined static body of knowledge.

4.3.5 Common user classifications

The characteristics described in the previous section can be represented as three orthogonal axes on a graph (Figure 4.6). Although any combination of characteristics can occur in principle (the axes are continuous), there are nevertheless a few frequently encountered user types which characterize many human–computer interface scenarios.

Casual users

These are not very knowledgeable about the task or knowledge domain which they wish to interact with, and are also unfamiliar with the system they will be using. Frequency of access may be low, and hence there is little motivation to study manuals or other training aids. Use of the system and its logical structure must be self-evident. Examples of systems intended for casual users include databases such as Prestel and

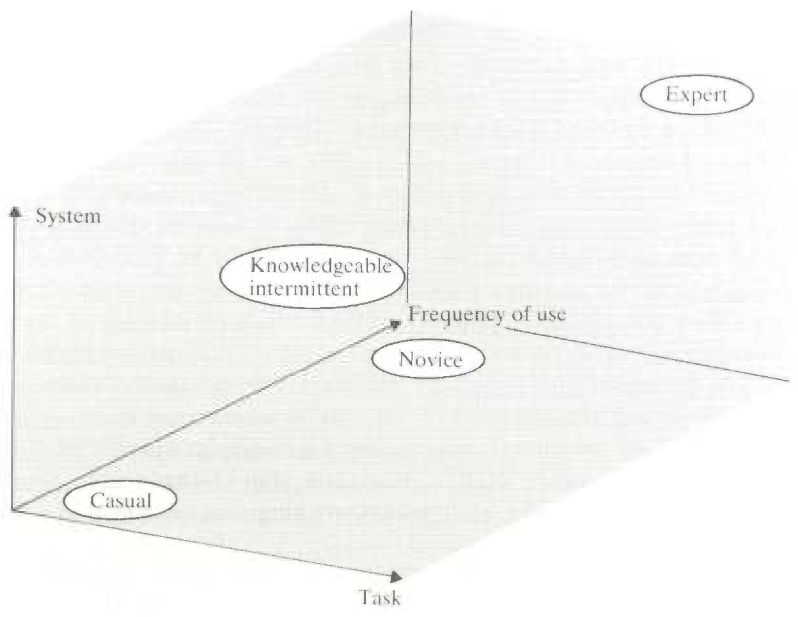


Figure 4.6 Dimensions of human capability in human–computer interaction

Teletext, computerized library search facilities, and computer-based banking facilities accessible to the general public. Most computer messaging systems ought to be designed for casual users, but are not!

Novice users

Although these may start out with the same characteristics as casual users, they are distinguished by being much more frequent users. There is therefore much more motivation for them to read manuals and other off-line training aids, and thus less need for on-line support. As experience and use increase, they may eventually become expert users. This category covers many white-collar workers who use office and data-processing systems regularly.

Knowledgeable intermittent users

Typically, these are professional staff who use a wide range of different types of computer support equipment. They generally have a clear idea of the task they wish to accomplish, and a good general understanding of the capabilities of computer technology, but only a limited amount of familiarity with the specific systems they need to use. Motivation to use the system is high in the general sense that they are aware of the 'power' improvements which the use of computers can bring, but tolerance is limited since they cannot afford to expend too much time learning the syntax of a system which will be used infrequently.

Expert users

Sometimes known as *power users*, these are fully conversant with both the task and the system they are using. Their main objective is to exploit to the maximum the capabilities the system provides for improving their performance. Excessive feedback and support is generally an irritant for these users, who are more interested in exploring ways of increasing the power of their interaction.

4.4 Task characteristics

Task characteristics are the complementary component of background knowledge which is required alongside knowledge of the characteristics of the human users before dialogue design can begin. Determination of task characteristics is accomplished using task analysis, which is analogous to the use of systems analysis to define requirements in other engineering fields. Task analysis can be thought of as comprising two major components, *task taxonomy* and *developing the user model.* Although determination of these components is best achieved using the formal techniques described in Chapter 5, in many cases a less formal approach can still yield valuable insights. Simply being aware of the need to envisage the task from the user's point of view, and taking steps to involve users in the early stages of design and prototyping are major positive steps.

4.4.1 Task taxonomy

Task taxonomy describes the somewhat mechanistic process of determining the set of tasks the system will be required to perform. The taxonomy will typically be represented as some kind of tree structure, with major tasks at its root and subtasks of each major task as the branches. The full structure may require several levels to map down from the primary functions to the basic atomic actions. The objective of the process is to develop a clear and logical hierarchy of functionality around which the dialogue can be built. In addition, producing the taxonomy will *per se* give the designer insights into how the system should be organized, both for clarity to the user and efficient engineering, by suggesting logical groupings of functions, and highlighting incompatibilities. Many methods of generating task and dialogue taxonomies have been proposed, for example command language grammar (CLG) (Moran 1981), task, action, language (TAL)

(Reisner, 1981, 1982) and goals, operators, methods and selection rules (GOMS) (Card *et al.*, 1983). Some of these have been discussed in more depth in Chapter 3.

4.4.2 User's task models

The task taxonomy is produced primarily according to the *designer's* perception of the required structure of system functionality, but this may differ markedly from the user's view. Many of the formal methods of task analysis (see Chapter 5) are intended to elicit an explicit *user's* task model by questioning users about their knowledge of a particular task, both in terms of the structure of the task and the objects and actions on those objects which constitute carrying out the task. By questioning a variety of users, and establishing the representativeness and commonality of particular structures, objects and actions, an objective view of the task is obtained.

This process is not the same as design-by-committee, which produced little more than a list of 'wants' with no inherent structure or association of value with function. An example of this latter strategy is embodied in the widely used UNIX *vi* editor, which provides an enormous number of functions, most of which are very rarely or never used by most programmers (Draper, 1984). Two principle reasons for this can be suggested. First, the command structure is flat: commands are not grouped in any logical way to encourage users to explore a variety of different methods of accomplishing a task. As a result, users tend to learn a minimum functional set of commands and add new ones to their repertoire only rarely. Second, the naming conventions for commands are haphazard: it is apparent that *vi* was designed and developed incrementally over a period of time. Had the full design been specified at the outset, a much more logical command set could have been chosen.

4.4.3 Creative system design

If human–computer system design consisted merely of mechanizing existing methods of performing a task, the opportunities for creative engineering design would be very limited. The opportunity for creativity is provided where the designer attempts to reconcile the information which emanates from the procedures of Sections 4.4.1 and 4.4.2 in a coherent way. The designer should not necessarily be bound to 'use the user's model' (Gaines, 1981), even though this is almost always better than using the designer's model! More important is that, whatever the model used, it should be readily discernible, predictable and consistent to the user.

Thimbleby points out the fallacy in always basing new designs upon their functional predecessor, using the car and horseless carriage as his example (Monk, 1984, Chapter 10). The problem is that organizing a system and its interface in a way which is inherently familiar to the user may expose technological limitations which substantially constrain the overall system performance. The job of the designer is to balance these purely technical constraints against the needs of the user to produce a compromise solution which provides optimum performance from the user and system in partnership.

4.5 Dialogue style classification

Sections 4.6–4.10 discuss dialogue styles under five headings: menus, form-filling, command languages, natural language and direct manipulation. These headings represent a reasonable classification of styles, but of course the boundaries are indistinct and some specific examples are not readily categorizable under any particular heading. For example, where does the boundary lie between a command language and a natural language dialogue? Early 'adventure' games used a form of so-called 'natural language' where the user could input any statement she or he wished, but the system only parsed the first four letters of each of the first two words. Undoubtedly the dictionary of valid words was larger than the typical command language, but was this really natural language, even if it appeared so to the user?

Another problem of classification is that several different dialogue styles may well be exploited for different purposes within a single system. For example, the Apple Macintosh variously uses direct manipulation (e.g., for file copying and transfer operations as part of its desktop metaphor), menus (e.g., pull-down menus for selecting options from the basic command list in the top status line), and form-filling (e.g. for specifying file names). In addition it provides other mechanisms such as dialogue boxes (for presenting system messages) and a variety of buttons for activating functions.

Martin (1973) defined over 20 styles of dialogue for alphanumeric display terminals alone, but Kidd (1982) noted that these could all be classified within the general categories of *menus, form-filling, command languages* and *natural language*. Her review was however written before the widespread introduction of graphics-based workstations and the WIMP style of interface, and thus largely ignored the concept of *direct manipulation* (Shneiderman, 1983). In view of the current prevalence of window system front-ends in computer workstations, together with the widespread use of graphically based displays in other areas such as industrial process control, air traffic systems and defence, it now seems reasonable to treat direct manipulation graphic systems as another distinctive dialogue style.

Sections 4.6–4.10 therefore review the distinctive characteristics of the five basic dialogue styles: Section 4.11 then presents various principles and guidelines which are relevant to all types of dialogue. Finally, Section 4.12 presents a case study of dialogue design based on a commercial teletext subtitle origination system.

4.6 Menus

4.6.1 Structures

The primary design problem in organizing a menu-based dialogue results from the fact that most realistic tasks require more commands than can conveniently be represented on a single menu. One possible solution is a linear (or circular) series of menus where the last option of each menu is 'other options'. This can work well for a limited number of menus where options are ordered according to their frequency of use, but is obviously impractical for large menu systems.

The most widely used technique is the hierarchical tree (Figure 4.3), which allows selection from a large number of choices with a relatively small number of options in each menu and few levels of the hierarchy. For example, a tree with 3 levels and 8 choices at each level can choose between 512 options. The structure need not be purely hierarchical. Additional paths allowing particular nodes to be reached by more than

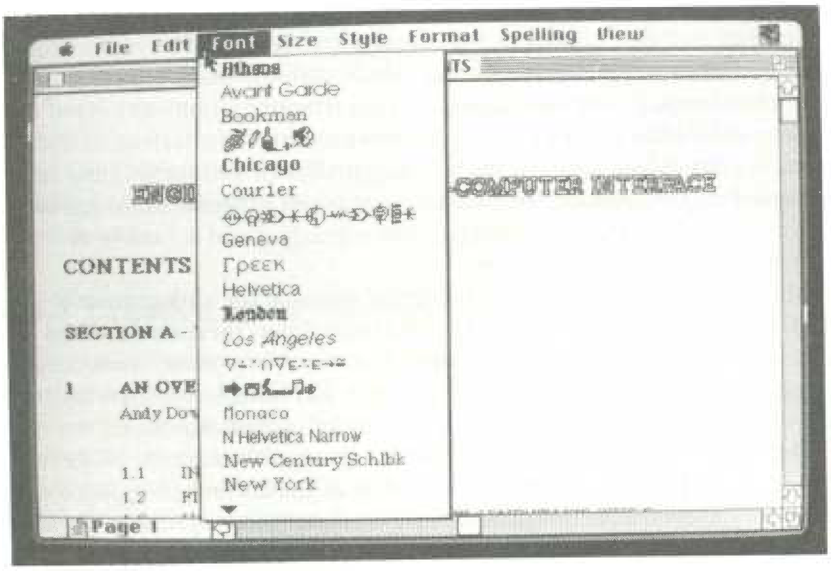


Figure 4.7 Example scrollable pull-down menu. (© 1989 Claris Corporation. All rights reserved. Claris and MacWrite are trademarks of Claris Corporation. Macintosh is a trademark of Apple Computer Inc.)

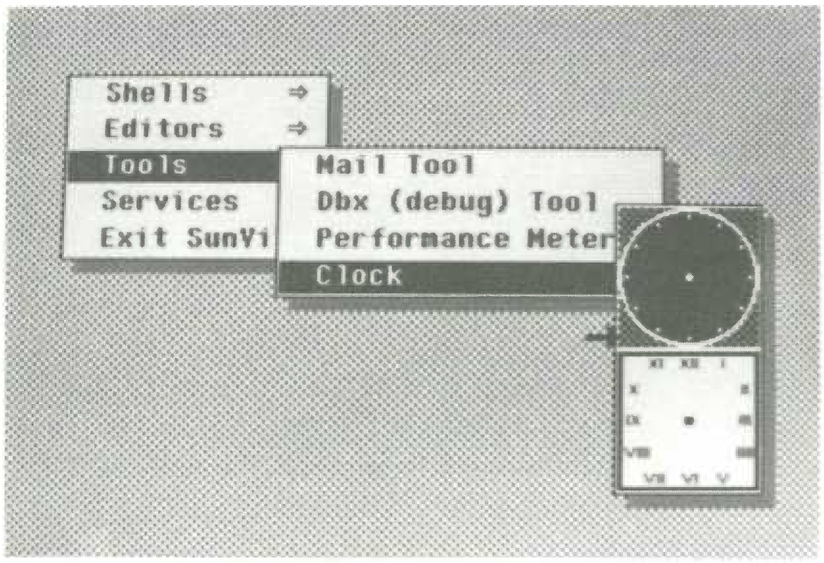


Figure 4.8 Example of pop-up menus with submenus. (Sun workstation)

one route can often be useful in representing alternative navigation strategies, and return paths offering shortcuts back up the hierarchy can be valuable where multiple selections must be made.

The hierarchy may be represented as a series of sequentially presented menus, but other formats are also possible. *Pull-down* menus (as on the Apple Macintosh, Figure 4.7), Microsoft Windows and other window systems) are principally a two-level hierarchy, while *pop-up* menus (e.g., on Sun workstations (Figure 4.8)) may allow submenus to be selected by dragging the mouse to the right from one of the initial selections, potentially to a depth of several levels (a similar capability has been added to Macintosh pull-down menus in recent software releases). Where a large number of menu options are required (for example to show alternative fonts in a word processor) a *scrollable* menu may be used, which shows only a subset of the available options initially.

4.6.2 Breadth versus depth: number of menu options

A binary decision such as:

```
Do you need help (y/n)?
```

represents a minimal menu, but the desirable maximum number of choices on a single screen is less clear (and in any case depends to some extent on character size and screen resolution). A considerable amount of research has been carried out on the menu depth-versus-breadth issue, but results are generally equivocal. There seems some consensus that menus should not contain more than about 12 items per frame to minimize search time and maintain a clear uncluttered display; equally it is generally agreed (and has been demonstrated experimentally) that providing less than 4 choices per frame slows down navigation in large hierarchical menu selection systems.

It is not necessary for every menu to contain the same number of options; a more important criterion is that groups of options contained within a single menu should be logically compatible and consistent. Appropriate groupings should normally be defined by the task taxonomy (Section 4.4.1). However, where menu length varies care should be taken to ensure that style, layout and structure are consistent between different menus. For example, menu selection methods (see next section) should not change from one menu to the next, list structure should be consistent (e.g., alphabetic, or most frequently used items first), and screen layout should remain consistent so that the choices and cursor position do not vary.

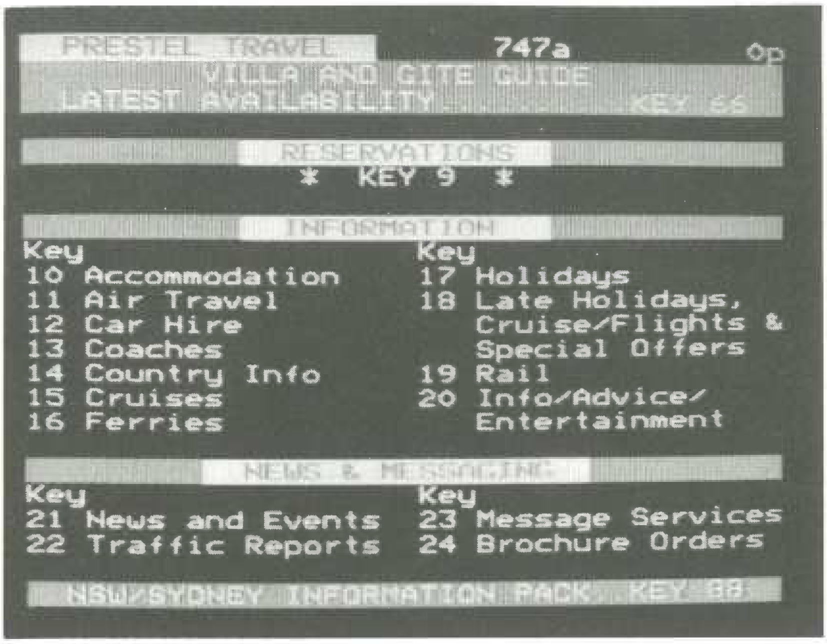
4.6.3 Selection mechanisms

Three methods of selection are commonly used for menus. The first and most obvious is to list the options numerically and choose between them by typing the number of the required selection, as for example in the Prestel system (Figure 4.9). An alternative is to select the item by typing its name, or more commonly, an abbreviation of this name (see Section 4.12 for an example). This can have advantages in being able to construct memorable acronyms by concatenating command abbreviations at several successive levels of the menu hierarchy, such as IOT—'Input Offline Titles' (from the example in Section 4.12).

Finally, the menu choice can be made by pointing to the item, using cursor keys, a mouse, a joystick, a touch screen or any of the pointing devices discussed in Chaper 8. Typically, feedback will be provided to indicate the currently selected choice by the use of reverse video or a different colour (see Figure 4.8); the choice is then confirmed by clicking the mouse button or carrying out some other confirmatory act.

4.6.4 Organization

Menu organization and screen design (see Sections 4.11.2 and 4.11.3) are seldom given the attention they deserve by design engineers whose training and experience is usually more technically oriented. Aesthetic appeal and the ability to visualize the system from the user's point of view are vitally important; unfortunately, these skills vary widely among



```
PRESTEL TRAVEL          747a          0p
          VILLA AND GITE GUIDE
LATEST AVAILABILITY.........    KEY 66

             RESERVATIONS
              *  KEY 9  *

              INFORMATION
Key                        Key
10 Accommodation           17 Holidays
11 Air Travel              18 Late Holidays,
12 Car Hire                   Cruise/Flights &
13 Coaches                    Special Offers
14 Country Info            19 Rail
15 Cruises                 20 Info/Advice/
16 Ferries                    Entertainment

           NEWS & MESSAGING
Key                        Key
21 News and Events         23 Message Services
22 Traffic Reports         24 Brochure Orders

  NSW/SYDNEY INFORMATION PACK. KEY 88
```

Figure 4.9 Numerical menu choice on the British Prestel system

engineers and computer scientists and are seldom formally developed or evaluated as part of training. In the same way as possession of a personal computer and desk-top publishing software does not guarantee any ability to construct appealing printed material, dialogue design tools do not automatically produce the most effective and attractive interface design.

The organization of menus in a multiple-menu system is the single factor which most affects the user's perception of the interface, since the underlying dialogue and model of the system are defined by this organization. There is strong experimental evidence that both error rates and access time are reduced when menus are structured meaningfully rather than randomly. The semantic organization should normally be in accordance with the user's model of the task, rather than some other unrelated structure such as alphabetic ordering; hence the need to establish task characteristics before proceeding to interface design.

Prototyping tools for menu-based systems are widely available (Chapter 7), and can be used to simulate the user interface of systems not yet built, allowing early evaluation and thus providing feedback to the designer, either confirming the validity of the interface model or highlighting inconsistencies.

4.6.5 Advantages and disadvantages

The advantages and disadvantages of menu selection as a dialogue technique are summarized in Table 4.2. Compared with other methods, *minimal typing* is required (none if a mouse or other pointing device is used), hence there is less opportunity for keying errors, a particular advantage for infrequent keyboard users. Similarly, the *low memory load* and frequent closure encountered in a menu selection task, and the *well-defined structure*, help to simplify decision making for inexperienced users, who are always presented with a fixed, limited set of options to select from. These same advantages also mean that menu selection systems can be used effectively even in interruptive surroundings where attention has to be divided between the system and other activities.

Software design for menu-based systems is generally straightforward, though care must be taken with data validation, display layout and error handling. Methods for navigating backwards as well as forwards through the menus should be provided, by means of *abort*, *backtrack*, and/or *undo* mechanisms. The design of hierarchical menu structures is inherently compatible with structured software design but in addition, dialogue design CAD tools are now becoming available for many menu-based systems, allowing rapid prototyping and permitting aesthetic characteristics such as screen layout to be designed by specialists in graphic design who may not have the requisite software design skills.

The main disadvantages of menu mode result from the requirement to display a large amount of auxiliary data to assist in menu selection. Depending upon display rate, this may result in irritatingly *slow response* (more commonly perceived as a problem by experienced and expert users than by infrequent users), and also requires substantial *display space*, which may constrain the choice of display type, or restrict the amount of other material which can be simultaneously displayed.

However, type-ahead or bypass mechanisms can allow the user to input the next selection without waiting for all menu options to be displayed (as, for example, on Prestel), and provide an elegant method for novices and experts to be accommodated compatibly on the same

Table 4.2 Advantages and disadvantages of menu-mode dialogues

Advantages	Disadvantages
Minimal typing	Sometimes slow
Low memory load	Consumes screen space
Well-defined structure	Not suited to data entry
Straightforward software design	Not suited to user-initiated dialogues
CAD tools available	Not suited to mixed initiative dialogues

system. An alternative approach is simply to provide a high-performance display system with sufficiently rapid response, powerful commands and aesthetic appeal to satisfy the 'power' user as well as the novice or intermittent user, as exemplified in window system menu-based command mechanisms.

A more fundamental problem concerns the type of interaction required. Menu selection is very suitable for *decision-making* tasks, but unsatisfactory for *data entry*, where large amounts of text need to be keyed into the system. If, for example, a number of parameters to a command must be specified, and these cannot readily be encoded with a menu (for example in an airline reservation where source and destination airports, departure and arrival times, flight number, and the name of the passenger must be specified), then a form-filling mode of dialogue will be much more suitable.

4.6.6 Conclusions

Menu-based dialogues are very suitable for casual, intermittent and novice users, all of whom can benefit from the explicit structure and simple interaction inherent in menu selection. For expert users, menu systems may be acceptable if system response and display rate are fast enough to avoid annoying delays.

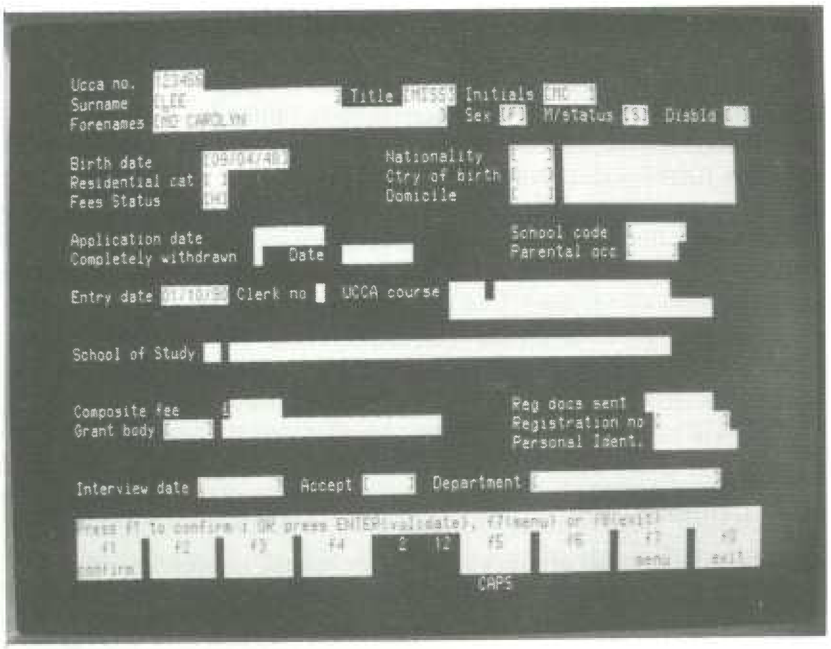
4.7 Form-filling

4.7.1 Structures and organization

Form-filling is a useful metaphor in human–computer dialogues because humans are inherently familiar with the concept of filling in forms, and because computers are widely used to manipulate and process databases of information, which are stored as records containing different fields of related information. These records are generally most conveniently visualized as forms. Familiar examples include airline bookings, library records, address lists, parts lists, personnel records, etc. (Figure 4.10).

The key benefit of form-filling cited by Shneiderman (1987, p. 122) is that all information in the record is simultaneously visible, giving the user a feeling of control over the dialogue. This implies that, whereas it is possible to implement a menu selection dialogue on any alphanumeric display device including a printing terminal, form-filling dialogues require a terminal which supports cursor control, so that the full screen of form information can be displayed and the cursor then moved around to each of the required data entry fields.

Although most users are familiar with the *task* of form-filling, the implementation of this task via the *system* may be less obvious. Unlike menu selection, the user usually has a significant degree of control over the process of data entry, and various application-specific syntactic rules



```
Ucca no.       123456
Surname        CLEE                          Title MISS  Initials CMC
Forenames      MC CAROLYN                    Sex F   M/status S   Disbld

Birth date          09/04/48           Nationality
Residential cat                        Ctry of birth
Fees Status         04                 Domicile

Application date                                  School code
Completely withdrawn        Date                  Parental occ

Entry date 01/10/8 Clerk no   UCCA course

School of Study

Composite fee    £                                Reg docs sent
Grant body                                        Registration no
                                                  Personal Ident.

Interview date          Accept [   ]   Department

Press F1 to confirm ; OK press ENTER(val:date), f7(menu) or f8(exit)
  f1       f2       f3       f4      2  12   f5      f6      f7      f8
confirm                                                     menu    exit
                                         CAPS
```

Figure 4.10 Example form-filling dialogue (from a university admissions database)

must therefore be learnt before proficiency is attained. Typical syntactic variables might include the following:

- *Display protection*—some areas of the display (generally all except the data entry fields) may not be accessible to the user.
- *Display field constraints*—data fields may be of fixed or variable length; user data entries may be constrained or free format.
- *Field content*—the user generally has to have some idea of permissible field contents: guidance may or may not be included as part of the form display.
- *Optional fields*—some fields may be optional; is this indicated textually or by some other display convention such as lower intensity level, different display colour, etc.?
- *Defaults*—are default entries possible? If so, are they indicated in the protected display area or in the data entry area?
- *Help*—additional help concerning filling in different fields may be available but concealed from the basic form display; if so, how is this additional information accessed?
- *Field termination*—data entry to a field may be terminated by the ENTER key, the RETURN key, or by filling the last available character space, or by moving to another field.

- *Navigation*—the cursor may be moved around the form using the TAB key in a fixed sequence, or using cursor control keys or another pointing device in any desired order.
- *Error correction*—the user may be able to correct errors by backspacing, by overwriting, by clearing and re-entering the field content, etc.
- *Completion*—how is completion of the whole form indicated?

As with menu systems, dialogue design tools for defining form-filling dialogues are readily available. In this case however, since there is a widespread need for tailored databases to support a variety of different database manipulation tasks, the dialogue design tools are often integrated with database application generator programs (for example Aston-Tate's dBASE and Microsoft's Excel).

4.7.2 Advantages and disadvantages

The advantages and disadvantages of form-filling as a dialogue technique are summarized in Table 4.3. The most significant advantage of this method is the basic familiarity with the *concept* of form-filling which any user has—all users can thus be classified as knowledgeable intermittent users, even though they may in fact be casual or novice users. Like menu mode, form-filling is a *structured dialogue* with *low memory load* for the user because it is computer-initiated. Data entry is simplified, but not as restricted as for menu mode, hence some *training* is required so that the user is aware of what constitutes a 'reasonable' response for each field. This may be viewed as an advantage compared with the flexibility of command and natural languages (see below), but a disadvantage compared with the minimal dialogue requirements of menu mode.

Software design is again relatively straightforward owing to the constrained nature of the dialogue, but more care is required in parsing and validating data entries than for menu mode. Many form-filling *dialogue design tools* are available as part of standard database

Table 4.3 Advantages and disadvantages of form-filling dialogues

Advantages	Disadvantages
Form-filling metaphor familiar	Sometimes slow
Simplified data entry	Consumes screen space
Limited training required	Not ideal for command selection
Low memory load	Requires display cursor control
Well-defined structure	Navigation mechanism not explicit
Straightforward software design	Some training needed
Dialogue design tools widely used	

application generators, but these may not be useful (except for interface simulation purposes) if the form-filling dialogue is intended to be part of a larger embedded system whose scope extends beyond the capabilities of the standard database package.

The basic disadvantages of form-filling mode are similar to menu mode: potentially *slow response* and the need for a large amount of *screen space*. In some ways, form-filling is complementary to menu mode in that the two dialogue styles are optimized for different dialogue types: form-filling is better for parameter entry, but less suited to command selection. Other disadvantages result from the need for users to have some familiarity with the *syntax* of the required dialogue, and the need for the display to be *cursor-addressable*—a possible hardware constraint.

4.7.2 Conclusions

Form-filling dialogues can be made suitable for all types of users, since all users are familiar with the basic concept of their use. Casual users may not have the specific system knowledge to complete the form unless any system-specific syntax is explicitly indicated as part of the on-screen data. Form-filling is better suited to parameter entry than command selection, and thus can often be used to advantage as a complementary dialogue technique where both styles of interaction are required.

To a large extent, the quality of form-filling dialogue depends upon the organization, presentation and content of the information provided in the form. Parameter fields need to be presented in a logical sequence while textual content needs to be succinct yet informative and unambiguous, form layout should be clear and uncluttered, and straightforward error recovery procedures are needed. Guidance on these requirements is presented in Section 4.11.

4.8 Command languages

4.8.1 Structures and organization

Command language dialogues are user-initiated and generally consist of the user typing a command or command string of syntactically correct words without prompting or help from the system. The most frequently encountered example of a command language for most computer users is the operating system language for their computer system. Other common examples of command languages include languages for text editors such as *vi* on UNIX systems. The distinctive feature of any command language dialogue is that no explicit support is provided to the user to show him the allowable set of commands: instead the user is expected to know (or learn) these commands. One implication of this is that the choice of command names is of particular importance in command language dialogues because these names must be memorized.

As with the menu dialogue mode, several command structures are possible. The simplest is a command list, as for example with the command set for the *vi* editor, which uses nearly every character of the ASCII character set (including control characters) to define some command (see Table 4.4 for a small example)! A disadvantage of this approach is the flat structure which results, making it difficult to remember the full command set, particularly since command names are not generally related in any logical or consistent way to their function. In consequence, this approach is advised only where the set of discrete commands required is fairly small. (It is generally a mistake to provide too much functionality in a system, just as it is to provide too little. Task analysis should reveal common strategies for achieving an objective; implementing additional '1-per-cent' functionality simply increases the amount of code and documentation, and thus may both reduce execution speed and slow user familiarization.)

Hierarchical command lists are also possible with command languages, and exactly parallel menu hierarchies, except that no explicit

Table 4.4 Sample cursor control commands for the Unix *vi* screen editor

Command	Function
(space)	right one character
∧B	back one page
∧D	scroll down half a page
∧F	forward one page
∧H	left one character
∧N	down one character
∧P	up one character
∧U	scroll up half a page
+	start of next line
−	start of previous line
/<string>	scan forwards for <string>
?<string>	scan backwards for <string>
B	back one word, ignoring punctuation
G<line no.>	go to line <line number>
H	start of top line
M	start of middle line
L	start of bottom line
W	forward one word, ignoring punctuation
b	back one word
e	end of current word
w	forward one word

prompting of allowable commands is provided in the command language. It is common to use subsequent words after the initial command word as qualifiers or parameters which specify the action of the command in more detail. The meaning of the subsequent words (or fields) of the command may be specific to each particular command (in which case it is accurate to view the command string as representing a hierarchy), or it may be possible for each of the words to identify a particular orthogonal characteristic of the command string, in which case the language may be viewed as an orthogonal structure. (Figure 4.4).

A practical problem in attempting to construct an orthogonal command language is that, in most cases, the required functionality cannot easily be accommodated by this structure. For example, the generic three-dimensional structure for an operating system command language might be of the form:

```
operation parameter filename
```

While this might fit some commands well (e.g., `edit`, `delete`, `compile`), others would not map easily onto the template (e.g., `copy` (requires both source and destination filenames), `directory` (no filename required)). In consequence, many command languages exhibit characteristics both of hierarchy and of orthogonality: typically the initial word specifies the basic command, with subsequent words adding command parameters (hierarchy), but at the same time there is some effort to generalize parameter structure across all commands (orthogonality).

4.8.2 Command syntax

Given this typical mixture of command language hierarchy and orthogonality, three command syntax styles have emerged, as follows:

Positional syntax

A positional syntax interprets the command words strictly according to their position within the command string. This type of syntax has been widely used in simple microcomputer operating systems such as CP/M and MSDOS. For example,

```
COPY TEMP FILE1
```

in a CP/M system might be assumed to have the meaning 'copy the file temp to filename file1'. In fact, the command copies in the reverse direction, succinctly illustrating one of the greatest dangers in using a positional syntax, namely, errors due to incorrect sequencing of the parameters. In this case, the 'logical' sequence of parameters is incorrect indicating an obvious error in the command structure, but in many cases the appropriate sequence for parameters in unclear, both to the designer and to the user. Where several parameters can follow the command, scope for error is increased: for 2 parameters only 1 incorrect sequence is possible, but for 3, five are possible and for 4, twenty-three.

Keyword syntax

Keyword syntax identifies each command parameter by an immediately preceding keyword. The actual sequence of parameters is then unimportant. Thus the previous example might be expressed validly by either of the following expressions using a keyword syntax:

```
COPY FROM FILE1 TO TEMP
COPY TO TEMP FROM FILE1
```

The delimiter between the keyword and its associated parameter is part of the syntax of the particular command language; space, equals (=) and hyphen (-) symbols are all commonly used. This method eliminates the possibility of sequencing errors, but at the expense of including additional redundant characters in the command line.

Mixed syntax

Mixed syntax simply combines the features of keyword and positional syntaxes to increase the allowable options, as for example in the UNIX command:

```
cc -o outfile cfile.c
```

where the parameter '-o' is used as a keyword to indicate that the next field specifies an output filename, and the field 'cfile.c' specifies a source file using positional notation. Although widely used and apparently more flexible than the use of keyword or positional syntax alone, mixed syntax can introduce confusion as to the required command string format where positional and keyword syntax rules conflict.

4.8.3 Advantages and disadvantages

Table 4.5 summarizes the advantages and disadvantages of command language dialogues. The advantages quoted are all only true for expert users: command languages have no real advantages for other types of users and many disadvantages.

The disadvantages highlight the reasons why command languages are suited to expert users. The need for substantial *training* and *regular use* to maintain proficiency define an expert user, and are likely to discourage any except the highly motivated. The reliance on the user's knowledge of allowable commands imposes a *high memory load*, but eliminates the need for extensive displays of menu options or other

Table 4.5 Advantages and disadvantages of command language dialogues

Advantages	Disadvantages
Fast	Long training
Efficient	Needs regular use
Precise	High memory load
Concise	Poor error handling
Flexible	
User-initiated	
Appealing	

support, leading to *concise* and compact use of the display. Providing meaningful *error messages* is much more difficult for command languages because the input is much less constrained, and the variety of possible errors is much larger than for menus or form-filling.

4.8.4 Conclusions

In summary, the command language dialogue style has several attractive advantages for frequent and experienced users, but is usually very discouraging for any other type of user. Where command languages are a legitimate choice, care is still needed to minimize training requirements and errors. Kidd (1982) gives the following succinct advice to minimize memory load and typing errors:

- choose memorable, non-confusable command words;
- use consistent command formats;
- keep command strings short;
- provide an explanatory backup online 'Help' facility;
- use the 'natural' ordering sequence for command parameters where possible;
- place optional and/or least used items at the end of the command list;
- use defaults to reduce typing where appropriate;
- provide clear and explicit messages;

and, if frequent errors persist,

- revert to a computer-initiated style!

4.9 Natural language

4.9.1 Justification

Science fiction has for many years propagated the idea of natural-language dialogue between humans and computers, but this superficial view is not supported by practical experience in the HCI field. The basis of the view may be an extrapolation from the fact that humans converse successfully with each other using 'natural language', but the flaw here lies in equating the computer with one of the humans involved in the dialogue.

As indicated in Table 4.1, human and computer aptitudes are complementary rather than equivalent, hence an optimal dialogue should aim to exploit the strengths of each partner. For example, natural-language dialogue between two humans normally assumes a symmetric communication channel: speech input and recognition speed in one direction is matched by speech output speed in the other. But human dialogue with computers generally uses asymmetric channels: the computer can output text to the screen very much faster than the human can type it in. Furthermore, the computer may have the capability to provide graphical or pictorial output as well as text. With these constraints on communication speed, it makes good sense to minimize typed input by the user (for example by use of menu selection), whereas extensive textual output can be supported.

Another aspect of the symmetry problem is the comparison between the style of input and output. In human–human communication each participant uses natural language, but the analogy fails when one human is replaced by a computer. On the one hand, if the computer uses natural language to 'speak' to the user, the underlying structure of the system being used immediately becomes more opaque and less readily determined; on the other, if the structure is made clearer, for example by means of forms or menus of options, then what is the point of requiring a (largely redundant) natural language input? (One class of system where the opacity of natural language is actually exploited is the ubiquitous adventure game: not only can the player attribute reassuringly 'human' capabilities to the computer, but also determining the underlying 'game model' represents the major intellectual function of the interaction.)

To add to these conceptual problems, natural-language dialogue is substantially more complex to program than any other dialogue style discussed. Natural-language recognition still represents a major linguistic research problem, and current natural-language processing systems are generally constrained to operate within limited knowledge domains using constrained syntax and restricted vocabularies.

An exception to this generally negative view of the potential of natural-language dialogues may exist in the case of natural-language speech communication with computers. Where data input is required to be by means of speech (in hands-off command and control operations, for example, or in speech input to computers via a telephone line—see Chapter 14), alternative dialogue modes such as single-word discrete input may require significant user training, ruling them out for casual and intermittent users. Acceptance of natural-language input for speech-controlled systems eliminates the need for the user to acquire specific syntactic knowledge about the interaction style.

4.9.2 Advantages and disadvantages

The advantages and disadvantages of natural-language dialogue are summarized in Table 4.6. The main advantage of natural language is the fact that *no special syntax* is needed. Natural language is also potentially *flexible* and *powerful*, though in practice these advantages depend to a large extent on how restrictive the specific dialogue implementation is. A further advantage is the potential for supporting mixed initiative dialogues, in contrast to the other dialogue styles discussed.

The major disadvantages of natural-language dialogues are those of natural language itself: by comparison with any purpose-designed artificial language, natural language is *ambiguous*, *imprecise* and *verbose*. In addition, in speech input applications, spoken natural language is significantly less well structured, accurate and syntactically correct than written language. From the designer's viewpoint, the software required to support a reasonably flexible natural-language dialogue is substantial: as a result natural-language input to computers is seldom as efficient, either in dialogue content or interface programming, as other dialogue styles.

A final problem with natural languages concerns the underlying perception of the system which a natural-language dialogue promulgates. Users can easily be misled by this style of dialogue into attributing much more intelligence to the system than is justified. This is due partly to the realistic and apparently thoughtful style of natural language response which such systems are programmed to generate (even when the system has in fact 'understood' very little of its natural language input), and partly to the fact that the difficulty in following the underlying logic of the computer's deductions from its input data readily persuades the user that these processes must be profound.

This point is amply illustrated by ELIZA (Weizenbaum, 1966; Weil,

Table 4.6 Advantages and disadvantages of natural-language dialogues

Advantages	Disadvantages
No special syntax	Ambiguous
Flexible and powerful	Imprecise
Natural	Verbose
Mixed initiative	Opaque
	Complex software design
	Inefficient

1965), a computer program written over twenty years ago to study natural-language dialogue between humans and machines. Its mode of operation was simply to identify key words within a specific knowledge domain from the input (Weil discusses examples where the system was programmed to respond as though it were a psychiatrist), and perform simple transformations to generate its response. For example, by converting first-person pronouns to second person and repeating the input sentence with a question mark at the end, an apparently plausible response is generated, with an implied request for elaboration:

```
Input:    My boyfriend made me come here
Response: Your boyfriend made you come here?
```

By means of a variety of tricks of this type, ELIZA was able to conduct convincing conversations with a variety of users, some of whom believed that they were communicating with a human rather than a computer, yet the underlying basis on which responses were generated was straightforward, and certainly involved no deep analysis of the meaning of the input.

4.9.3 Conclusions

In general, it appears that natural language does not have much to offer as a basis for dialogue design in most applications, being both complex to program and inefficient for most dialogue situations. The main exception to this may be where a speech-based dialogue is required, particularly if both input and output use the speech modality. In this case, the ephemeral nature of speech output, and the difficulty of generating speech input with strict syntactic constraints may justify the use of what otherwise is a verbose, vague and ambiguous method of communication.

4.10 Direct manipulation

4.10.1 Styles and metaphors

Most computer users are by now familiar with the WIMP (Window, Icon, Menus, Pointer) style of dialogue popularized by such machines as the Apple Macintosh, but the desktop metaphor used by this machine is only one example of a direct-manipulation dialogue. The essential characteristic of such dialogues is that some kind of *direct representation* of the task is presented to the user by the system, with the result that the desired operation or command is achieved by *directly manipulating* the virtual reality embodied in the display. The primary advantage of using a *metaphor* to represent the actual function is that operations and commands can be readily suggested by analogy between the computer representation and its real life equivalent.

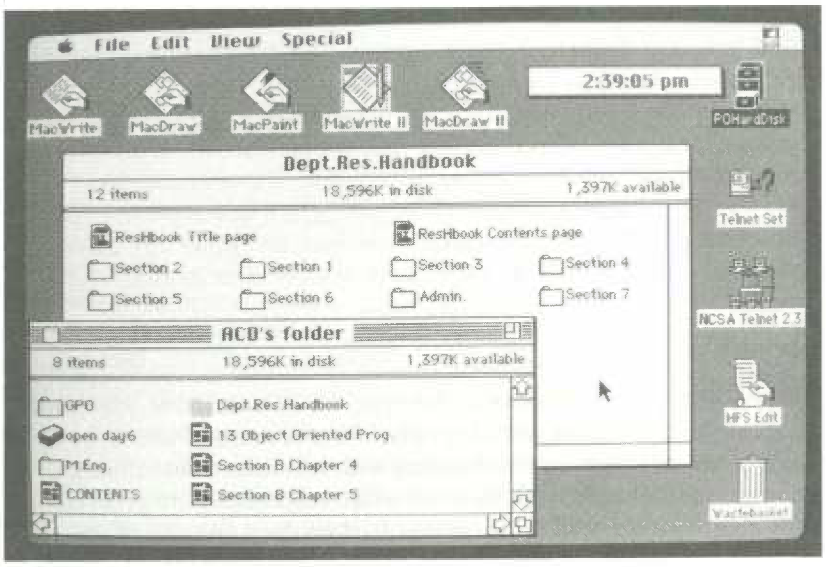


Figure 4.11 Macintosh desktop screen showing icons, folders and windows

Thus in the case of the Macintosh desktop (Figure 4.11), by using the metaphor of a folder to present a disk directory, the user may be expected to deduce that he can open the folder and see what is inside (list the directory in a new window), and then manipulate files within the folder (extract, insert, duplicate, discard into the wastebasket, etc.). The concept of hierarchical directories is readily conveyed by inserting a folder into an existing open folder (window). Open folders become documents (windows) on the desktop (screen) and documents can overlap, be placed on top of each other, and moved around at will. Icons with graphical images suggestive of their function are used to represent application programs such as word processors, graphics and communication packages. Supporting desktop operations requires significant processing power, but the biggest problem faced by the desktop metaphor is the limited size and screen resolution attainable using CRT technology, compared with an actual desk! In addition, direct manipulation commonly assumes the availability of some variety of pointing device, such as a mouse, stylus or finger (for touch screens).

4.10.2 Objects and actions

Although these concepts seem simple and obvious with hindsight, the trick of designing an effective direct-manipulation system lies in choosing a simple and readily understood metaphor. Central to this metaphor is an orthogonal division between *objects* which are represented as physical images (such as icons) on the screen, and *actions*, which are conveyed dynamically by manipulating the objects. The choice of metaphor is thus constrained by the need on the one hand to choose objects which are recognizable and memorable, and on the other to produce a set of actions which can be readily represented graphically and are as consistent as possible across all objects (Rosenburg and Moran, 1984). Furthermore, it may be possible within the metaphor to code particular tasks either as objects or actions, for example, the Xerox Star workstation represents the printer as an icon or resource which can be selected, whereas the Apple Macintosh treats printing as an action chosen using a menu.

Graphic design plays a fundamental part in creating an illusion of manipulable objects. Design of direct-manipulation interfaces requires a new breed of graphical designer who is not only capable of mastering the aesthetics of presenting static images on a bit-mapped graphics display, but also can cope with the dynamic aspects of such displays. Animation may be used in a wide variety of ways: to indicate selection of an object (e.g., inverse video), to indicate different modes (e.g., different cursor types), to indicate progress of a time-consuming operation (e.g., the watch or hourglass icon), to provide a visual focus on the object which is being manipulated. The example above might suggest that using animated graphics is a key to success: as the examples in the next section show, however, not all direct-manipulation interfaces require such complexities.

4.10.3 Examples

Process control

Process control systems in industries such as chemical engineering and power systems generation and distribution have for many years used visual representations of their systems as the interface between the human system controller and the computer-based control system. In most cases this is relatively easily achieved technically since the system being represented is static and so the basic display model can consist mainly of a symbolic pictorial representation of the system on a board. Fixed controls (switches, potentiometers, etc.) are used to adjust parameters of the system, and basic status information is supplied by means of illumination intensity, colour and other simple displays at appropriate points on the board.

Air traffic control and weapons guidance

Radar systems have always relied to a large extent upon the remarkable pattern recognition capabilities of the human to isolate the radar return corresponding to a plane from surrounding clutter. In essence, the three-dimensional world in which the plane flies is translated into a readily interpreted two-dimensional plan. Similarly, weapons guidance systems provide a simplified visual representation of the world with the target and method of controlling the weapon so as to hit the target emphasized.

Simulators and video games

Simulators are widely used in fields such as flight training and power station control to provide cheaper tuition than could be achieved by using the system itself, and to explore aspects of control which would be dangerous or impossible on a real system. Aircraft simulators may be so realistic as to appear almost identical to the actual aircraft visually and perceptually, but even with very much simplified display information, users can learn a great deal about the principles of flight, if the model accurately simulates flight dynamics. One class of video games simulates flight, car racing, and other sports with varying degrees of realism and accuracy, but in all cases the model represents a metaphor of the actual activity. Other video games may be more abstract in nature; nevertheless, there is usually an element of familiarity in the basic theme of the game which encourages analogy and experimentation on the part of the user.

Screen editors

Screen editors, where the computer screen represents a window in a text file and text is identified by moving a cursor on the screen using a mouse or cursor keys or both, are now the norm in most computer systems. The WYSIWYG concept (what you see is what you get) is central to modern word processors. However, in earlier interactive systems line-based editors were common, representing a command language approach to text editing (e.g., see the UNIX editor *ed*). While some gurus still welcome the power of line editors, the vast majority of computer users much prefer the visual clarity of a screen editor.

Graphics

Graphics generation of any sort on computer systems might seem to require graphics displays, pointing devices and digitizers, as embodied in a wide variety of microcomputer drawing, painting and business