

USER SUPPORT

11

OVERVIEW

- Users have different requirements for support at different times.
- User support should be:
 - available but unobtrusive
 - accurate and robust
 - consistent and flexible.
- User support comes in a number of styles:
 - command-based methods
 - context-sensitive help
 - tutorial help
 - online documentation
 - wizards and assistants
 - adaptive help.
- Design of user support must take account of:
 - presentation issues
 - implementation issues.

11.1 INTRODUCTION

There is often an implicit assumption that if an interactive system is properly designed it will be completely intuitive to use and the user will require little or no help or training. This may be a grand ideal but it is far from true with even the best-designed systems currently available. It is even perhaps an unhelpful ideal: a computer is a complex piece of equipment – what other such equipment do we expect people to use without instruction or help? A more helpful approach is to assume that the user will require assistance at various times and design this help into the system.

The type of assistance users require varies and is dependent on many factors: their familiarity with the system, the job they are trying to do, and so on. There are four main types of assistance that users require:

- quick reference
- task-specific help
- full explanation
- tutorial.

Quick reference is used primarily as a reminder to the user of the details of tools he is basically familiar with and has used before. It may, for example, be used to find a particular command option, or to remind the user of the syntax of the command. Task-specific help is required when the user has encountered a problem in performing a particular task or when he is uncertain how to apply the tool to his particular problem. The help that is offered is directly related to what is being done. The more experienced or inquisitive user may require a full explanation of a tool or command to enable him to understand it more fully. This explanation will almost certainly include information that the user does not need at that time. The fourth type of support required by users is tutorial help. This is particularly aimed at new users of a tool and provides step-by-step instruction (perhaps by working through examples) of how to use the tool.

Each of these types of user support is complementary – they are required at different points in the user's experience with the system and fulfill distinct needs. Within these types of required support there will be numerous pieces of information that the user wants – definitions, examples, known errors and error recovery information, command options and accelerators, to name but a few. Some of these may be provided within the design of the interface itself but others must be included within the help or support system. We will look at appropriate ways of supporting these requirements. The different types of help required also imply the need for provision of different types of help system. In this chapter, we will look at a number of different types of user support system and will try to determine how to design a good user support system.

A distinction is often made between help systems and documentation. Help systems are problem oriented and specific, whereas documentation is system oriented and generic. This is an artificial distinction when considering the design of such systems since the same principles apply to both, and indeed there is a lot of overlap

between the two. Instead of drawing a fixed line between the two, we will consider all types of user support in terms of the requirements they fulfill. We will also concentrate on online support, although much of what is said will be helpful in designing paper documentation and tutorials. Before we look in more detail at the different approaches to providing user support, we will think for a while about the general requirements that the ideal help system should have.

11.2 REQUIREMENTS OF USER SUPPORT

If we were to design the ideal help system, what would it look like? This is a difficult question to answer, but we can point to some features that we might like our help system to have. Not every help system will have all of these features, sometimes for good reason, but they are useful as benchmarks against which we can test the support tools we design. Then, if our system does not have these features, it will be by design and not by accident! Some of these terms have also been used in Chapter 7 in discussing principles for usability. The use of the terms here is more constrained but related.

11.2.1 Availability

The user needs to be able to access help at any time during his interaction with the system. In particular, he should not have to quit the application he is working on in order to open the help application. Ideally, it should run concurrently with any other application. This is obviously a problem for non-windowed systems if the help system is independent of the application that is running. However, in windowed systems there is no reason why a help facility should not be available constantly, at the press of a button.

11.2.2 Accuracy and completeness

It may seem obvious to state that the assistance provided should be accurate and complete. But in an age where applications are frequently updated, and different versions may be active at the same time, it is not a trivial problem. However, if the assistance provided proves not to match the actual behavior of the system the user will, at best, become disillusioned with the help facilities, and, at worst, get into difficulties. As well as providing an accurate reflection of the current state of the system, help should cover the *whole* system. This completeness is very important if the help provided is to be used effectively. The designer cannot predict the parts of the system the user will need help with, and must therefore assume that all parts must be supported. Finding no help available on a topic of interest is guaranteed to frustrate the user.

11.2.3 Consistency

As we have noted, users require different types of help for different purposes. This implies that a help system may incorporate a number of parts. The help provided by each of these must be consistent with all the others and within itself. Online help should also be consistent with paper documentation. It should be consistent in terms of content, terminology and style of presentation. This is also an issue where applications have internal user support – these should be consistent across the system. It is unhelpful if a command is described in one way here and in another there, or if the way in which help is accessed varies across applications. In fact, consistency itself can be thought of as a means of supporting the user since it reinforces learning of system usage.

11.2.4 Robustness

Help systems are often used by people who are in difficulty, perhaps because the system is behaving unexpectedly or has failed altogether. It is important then that the help system itself should be robust, both by correct error handling and predictable behavior. The user should be able to rely on being able to get assistance when required. For these reasons robustness is even more important for help systems than for any other part of the system.

11.2.5 Flexibility

Many help systems are rigid in that they will produce the same help message regardless of the expertise of the person seeking help or the context in which they are working. A flexible help system will allow each user to interact with it in a way appropriate to his needs. This will range from designing a modularized interactive help system, through context-sensitive help, to a full-blown adaptive help system, which will infer the user's expertise and task. We will look at context-sensitive and adaptive help in more detail later in the chapter. However, any help system can be designed to allow greater interactivity and flexibility in the level of help presented. For example, help systems built using hypertext principles allow the user to browse through the help, expanding topics as required. The top level provides a map of the subjects covered by the help and the user can get back to this level at any point. Although hypertext may not be appropriate for all help systems, the principle of flexible access is a useful one.

11.2.6 Unobtrusiveness

The final principle for help system design is unobtrusiveness. The help system should not prevent the user from continuing with normal work, nor should it interfere with the user's application. This is a problem at both ends of the spectrum. At one end the textual help system on a non-windowed interface may interrupt the user's work. A possible solution to this if no alternative is available is to use a split-screen presentation.

At the other end of the spectrum, an adaptive help system that can provide help actively on its own initiative, rather than at the request of the user, can intrude on the user and so become a hindrance rather than a help. It is important with these types of system that the 'suggest' option can be overridden by the user and switched off!

11.3 APPROACHES TO USER SUPPORT

As we noted in the previous section, there are a number of different approaches to providing help, each of which meets a particular need. These vary from simple captions to full adaptive help and tutoring systems. In this section we will concentrate on the styles of help provided rather than any particular help system (although we will use real help systems for illustration). We will then go on to look at adaptive help in more detail.

11.3.1 Command assistance

Perhaps the most basic approach to user support is to provide assistance at the command level – the user requests help on a particular command and is presented with a help screen or manual page describing it. This is the approach used in the UNIX *man* help system and the DOS *help* command, as well as through the search in Windows help.

This type of help is simple and efficient if the user knows what he wants to know about and is seeking either a reminder or more detailed information. However, it assumes that the user does know what he is looking for, which is often not the case. In any complex computer system there will be some commands that the user knows well and can use and some of which he is aware but uses rarely. Command assistance deals well with these. However, there will also be commands that the user does not know about but needs, and even commands that the user thinks exist but which do not. Command assistance cannot provide the user with help for these two groups of command.

11.3.2 Command prompts

In command line interfaces, command prompts provide help when the user encounters an error, usually in the form of correct usage prompts. Such prompts are useful if the error is a simple one, such as incorrect syntax, but again they assume knowledge of the command.

Another form of command prompting, which is not specifically intended to provide help but which supports the user to a limited degree, is the use of menus and selectable icons. These provide an aid to memory as well as making explicit what commands are available at a given time. However, they still assume a certain amount of knowledge about what the commands are for, so additional support is still required.

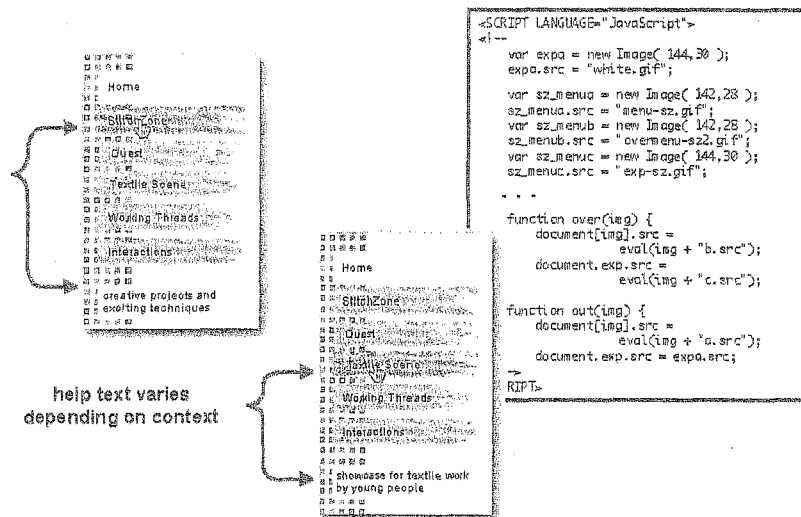


Figure 11.1 Context sensitive help on a web page using JavaScript rollovers

11.3.3 Context-sensitive help

Some help systems are context sensitive. These range from those that have specific knowledge of the particular user (which we will consider under adaptive help) to those that provide a simple help key or function that is interpreted according to the context in which it is called and will present help accordingly. Such systems are not necessarily particularly sophisticated. However, they do move away from placing the onus on the user to remember the command. They are often used in menu-based systems to provide help on menu options. The Microsoft Office *What's This?* option, tool-tips and some kinds of web page rollover are examples of this. When enabled, explanatory text is displayed when the cursor is over a screen widget (see Figure 11.1). The invocation of help is interpreted in terms of the context in which it is made.

11.3.4 Online tutorials

Online tutorials allow the user to work through the basics of an application within a test environment. The user can progress at his own speed and can repeat parts of the tutorial if needed. He will also get a feel for how the application works by experimenting with examples, albeit small ones, or by watching an automated demonstration of how to perform a task.

Most online tutorials have no intelligence: they know nothing about the user and his previous experience, nor about the domain nor even about teaching style. Intelligent tutoring systems, which use similar techniques to adaptive help systems (see Section 11.4), attempt to address this issue but, apart from tutoring programming applications, are impractical as tutorials for most applications. Online

tutorials are therefore inflexible and often unforgiving. Some will fail to recognize the correct answer to a problem, simply because it is not formatted as expected.

An alternative to the traditional online tutorial is to allow the user to learn the system by exploring and experimenting with a version with limited functionality. This is the idea behind the *Training Wheels* interface proposed by Carroll and his colleagues at IBM [60]. The user is presented with a version of the full interface in which some of the functionality has been disabled. He can explore the rest of the system freely but if he attempts to use the blocked functionality he is told that it is unavailable. This approach allows the user freedom to investigate the system as he pleases but without risk. It was found that new users spent more time using this system than they did the full version, spent less time recovering from errors and gained a better understanding of the operation of the system.

11.3.5 Online documentation

Online documentation effectively makes the existing paper documentation available on computer. This makes the material available continually (assuming the machine is running!) in the same medium as the user's work and, potentially, to a large number of users concurrently. However, it can be argued that the type of (usually large) manuals that are appropriate as paper reference systems are less appropriate online. Paper is a familiar medium to most of us, and it is still the case that people prefer reading text on paper than on a computer screen. We have developed quite sophisticated browsing skills with a paper medium and books are designed to provide cues to aid this, such as indexing, contents and page numbering, as well as having physical cues such as position in the book. These features are not reproduced in most documentation systems. But paper manuals get lost easily, are constrained to one physical location, and are invariably somewhere else when you want them. Online documentation is one way of avoiding these problems.

Documentation is designed to provide a full description of the system's functionality and behavior in a systematic manner. It provides generic information that is not directed at any particular problem. The amount of information contained in manual pages is usually high, which can in itself create problems for the user — there is too much detail and this effectively 'masks' the information the user wants to find. Perhaps for this reason, online documentation is often used by more expert users as a resource or reference, sometimes to enable them to advise less experienced users. The experts may not know the information off the top of their head but they know where to find it and how to extract the details that are relevant to a given problem.

The use of hypertext can make online documentation more accessible to the inexperienced user (see Chapter 21 for more details of hypertext). Hypertext stores text in a network and connects related sections of text using selectable links. By clicking on a link, the user can go to a related subject instantly. Documentation structured using hypertext supports browsing and usually includes different media (for example, diagrams and visual examples). An example is the help system shown in Figure 11.2, and most Windows applications' help systems.

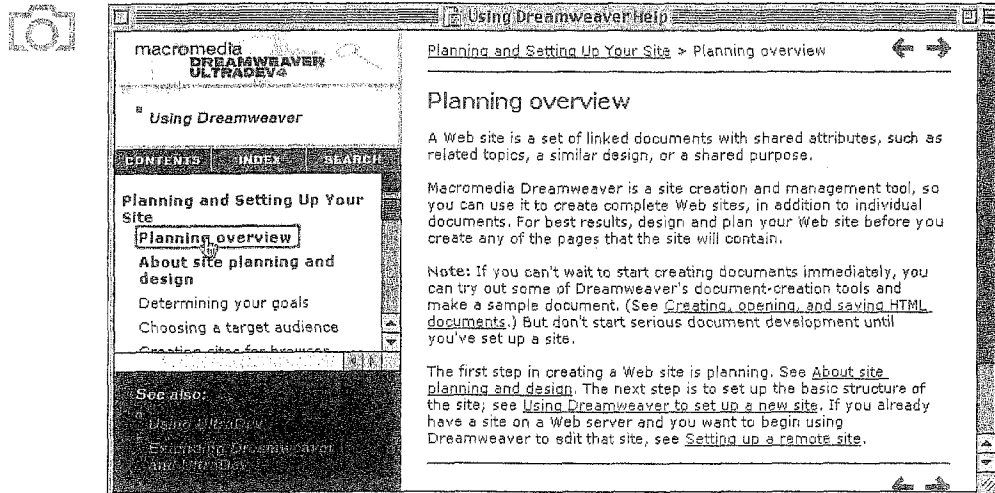


Figure 11.2 Screen shot of the online documentation for Macromedia's Dreamweaver UltraDev version 4.0. Courtesy of Macromedia, Inc.

However, this does suffer from a problem common to hypertext systems – navigation. The user can get lost within the hypertext and lose track of where he is and where he has been. A facility to return ‘home’ (to the top level) is usually provided but this may mean that the user wastes considerable time restarting his search.

An alternative approach which attempts to simplify online documentation and make it more accessible to novice and casual users is the *minimal manual* [58]. This simplifies the documentation by stripping out all but the bare essentials. The documentation that remains is focussed towards the user's tasks and emphasizes error recovery. Experiments with this manual showed that users learned to use the system 40% faster than with the full manual.

Simple guidelines for online documentation

- Use clear structure with headings to provide signposting.
- Organize information according to user tasks.
- Keep sentences short, to the point and jargon free. Use simple but unpatronizing language.
- Set out procedures in order and number steps. Highlight important steps.
- Use examples where possible.
- Support searching via an index, contents, glossary and free search.
- Include a list of error messages.
- Include Frequently Asked Questions (FAQ) with clear answers.

(Adapted from Hobbs and Moore [177])

11.3.6 Wizards and assistants

A *wizard* is a task-specific tool that leads the user through the task, step by step, using information supplied by the user in response to questions along the way. They are distinct from demonstrations in that they allow the user actually to complete the task. For example, if the user wishes to format a resumé, the Microsoft Word resumé wizard will take him through a series of questions on the style and sections required, and ask him to enter some basic personal data. The wizard then creates a resumé matching the parameters submitted.

Wizards are common in modern applications and provide support for task completion. The user can perform quite complex tasks safely, quickly and accurately. This is particularly helpful in the case of infrequent tasks where the learning cost of doing the task manually may be prohibitive or where there are many steps that must be completed in a specific sequence. However, wizards can be unnecessarily constraining, they may not offer the user the options he wants, or they may request information that the user does not have. A well-designed wizard will allow the user to move back a step as well as forward, will provide a progress indicator showing how much of the task is completed and how many steps remain, and will offer sufficient information to allow the user to answer the questions.

Another more recent development in user support is the *assistant*. Assistants are software tools that monitor user behavior and offer suggestions or hints when they recognize familiar sequences. An early example of this is Eager (see also Chapter 4), a software agent that watches users as they work. When it notices the user repeating a sequence of actions, a cat icon appears, suggesting the next action. The user can accept or ignore the suggestion. Eager is unobtrusive and under user control at all times.

More recent examples of assistants have not been so successful. The Microsoft Office Assistant, 'Clippy', the infamous animated paper clip with the expressive eyebrows, was introduced in Office 97 but scrapped just a few years later with the introduction of XP. The official reason is that XP features make the need for the Office Assistant redundant, but the underlying reason is clear from Microsoft's own self-deprecating 'Clippy' homepage – the paper clip assistant was universally despised as irritating and unhelpful. So what went wrong? Remember the requirements for online help that we outlined at the beginning? One of these states that online assistance should be unobtrusive, and Clippy, with his long lists of suggestions, and continuous animations, was anything but. In addition, the suggestions made were often inappropriate – 'It looks like you're writing a letter' when you are in fact doing something completely different. Finally, the embodiment of the paper clip, though intended to be 'cute', was perceived by many frequent users as simply irritating.

Microsoft Office Assistant may have retired but he provides an important lesson in online user support. Help must be under the user's control and offers of help should be unobtrusive – it should not be up to the user to switch off the assistant or to close it down. The replacement Microsoft XP features such as *smart tags* have learned this lesson. Smart tags provide rapid access to actions associated with a

particular task (for instance formatting a table) much as Office Assistant did. But smart tags are indicated by a small icon, which appears near the object of interest, then disappears when the user performs another action. It is up to the user to select the smart tag if it is of interest at that point, but the icon is small and unobtrusive enough to be ignored if not required. It is interesting that the tags are offered through a functional iconic representation – it seems that assistants with ‘attitude’ are out!

11.4 ADAPTIVE HELP SYSTEMS

In any large or complex computer system, users will be familiar with a subset of the available functionality, demonstrating expertise in some applications and having no experience with others, even to the point of being unaware of their existence. In addition, different users will have different needs and levels of understanding. Adaptive help systems attempt to address these problems by adapting the help that they provide to the individual user who is making the request and by actively suggesting alternative courses of action of which the user may not be aware.

Adaptive help is a special case of a general class of interactive systems, known as intelligent systems. These include domain-specific expert systems, intelligent tutoring systems and general adaptive interfaces. We will concentrate in this discussion on adaptive help systems, since they are most relevant here, and incorporate aspects of the others, but it should be noted that many of the techniques we will look at can be applied in these other systems. Since these represent a significant class of interactive system, we will cover the techniques in some detail.

Adaptive help systems operate by monitoring the activity of the user and constructing a model of him. This may include a model of his experience, preferences, mistakes, or a combination of some or all of these. Using this knowledge, together with knowledge of the domain in which the user is working, and, sometimes, general advisory or tutorial strategies, the adaptive help system will present help relevant to the user’s current task and suited to his experience. That at least is the theory. In practice it is not as simple as it sounds. First, the knowledge requirements of such a system are considerable, and data on interaction are particularly difficult to interpret. Secondly, there is the issue of control and initiative within the interaction. Should the help system take an active role, removing some control from the user, and will the adaptivity confuse the user, if he perceives it as ‘shifting ground’? Thirdly, what exactly should be adapted and what will be the result of the adaptivity? Finally, what is the scope of the modeling and adaptivity: does it extend beyond the application level and, if so, how does it deal with the variation in expertise of a single user across an entire system? We will consider some of the developments and solutions, concentrating, in particular, on the knowledge requirements.

DESIGN FOCUS



It's good to talk — help from real people

Although well-designed documentation and intelligent help systems are important, there is, perhaps, nothing like talking to a real person! As we noted in Section 11.3.5, expert users are often more likely to consult documentation, as there is a skill in knowing how it is organized. Studies of UNIX help found that novice users preferred not to look at *man* pages, instead they asked local experts who did look at the pages. This pattern of local experts has been found across all types of systems in many environments.

In web-based systems there are many ways of linking more static documentation with dynamic interpersonal support. There may be a link from the electronic to the personal: technical support sites often have some sort of email query form for when you cannot find what you want online. There may also be inter-personal to electronic links: a telephone support desk may email you a URL to guide you to suitable information.

Other users of a product can also be a source of help, and many product sites, especially open-source products, include some way for users to submit their tips or advice. For example, the online manual pages for the PHP web scripting language have for each function a series of user tips that you can add to yourself. There are important issues of community culture here as the tips are generally useful and do not include extraneous comments.

User community and manufacturers' technical support may be integrated. For example, Sun's Java developer support site includes a community area with online user forums and live chat with Sun staff, authors and other experts.

See www.php.net/ and <http://developer.java.sun.com/developer/community/>

see also pg 400

11.4.1 Knowledge representation: user modeling

Every interactive system that is built incorporates some model of the user for whom it is intended. In many systems this model is the designer's view of the user and is implicit within the design. The designer has in mind a 'typical' user, and builds the interface accordingly. If the designer has done her homework this model can be quite effective. However, it does assume that all users are essentially the same and have the same requirements.

Other systems allow the user to provide a model of himself around which the system will be configured. Simple examples of this are browser or email preferences that can adjust certain parameters of the system to the requirements of the user. Such systems are called *adaptable*, since the user is able to adapt his own environment to suit his preferences. This increases the flexibility of the system but places the onus of the customization on the user. The result of this is that users have access only to the default system when they most need flexibility: that is, when they first start out. It is only later that they have the know-how to construct the necessary model.

The third approach to providing the system with a model of the user, and the one used in adaptive help systems, is to have the system construct and maintain a model of the user based on data gleaned from monitoring the user's interaction. This model can then be consulted when required. This automatic approach to user modeling also has the problem of the setup time required, during which time the user has a default system, but the onus to build the model is taken away from the user. Various suggestions have been made as to how to deal with the setup time, including getting the user to choose an initial default model, and building a model based on pre-use activity, such as game playing. The former is problematic in that again it makes the user decide on a model at a time when he may not have sufficient experience to do so effectively. The latter may not produce a model that is transferable to the actual domain. The most common approach is still to provide a basic start-up default model and concentrate on rapidly updating this for the actual user. The default model may be based on experimental or observational results gleaned in evaluation.

So how are user models constructed and maintained? There are a number of approaches. Some quantify user experience or classify users into stereotypes; some compare the user's behavior with some norm; others maintain a catalog of known errors and compare user actions to these.

Quantification

This is one of the most simple approaches to user modeling. The system recognizes a number of levels of expertise, which it will respond to in different ways. The user is placed on one of these levels, and moves between them, based on a quantitative measure of his expertise at that time. Different activities are given weightings, and the user is scored according to the weightings of the activities he takes part in. If the score exceeds a certain threshold, the user is moved to a different expertise level and the system adapts accordingly.

This approach is simple and measures the user at a coarse level of granularity. However, it is effective for simple adaptivity. For example, this method was used by Mason to adapt the presentation of command prompts to the user's level of experience [228]. The system used a set of rules, which dictated when a user's level of expertise changed. For example,

Move from Level 1 to Level 2

If

- the system has been used more than twice (0.25)
- commands x and y have been used effectively (0.20)
- help has not been accessed this session (0.25)
- the system has been used in the last 5 days

Such a model can only give a rough approximation of the user's expertise, but at the same time requires little analysis to extract the required information from the system logs. This approach can be used effectively in adaptive tutorials.

Stereotypes

Another approach to automatic user modeling is to work with stereotypes. Rather than attempting to build a truly individual model of the user, the system classifies the user as a member of a known category of users or stereotype. Stereotypes are based on user characteristics and may be simple, such as making a distinction between novice and expert users, or more complex, for example building a compound stereotype based on more than one piece of information. There are several ways of building stereotypes. One is to use information such as command use and errors to categorize different types of user and then to use rules to identify the stereotype to which the user belongs. An alternative method is to use a machine learning approach, such as neural networks, to learn examples of different types of user behavior (from actual logs) and then to classify users according to their closeness to the examples previously learned. Stereotypes are useful in that they represent the user at the level of granularity at which most adaptive help systems work, and do not attempt to produce a sophisticated model, which will not be fully utilized. After all, if the only information that is available about the user at any time is how he is interacting with the system, it is not possible to infer very much about the user himself. However, what can be inferred may be exactly what is required to provide the necessary level of help.

Overlay models

One of the most common techniques used is the overlay model. Here an idealized model, often of an expert user, is constructed and the individual user's behavior compared with it. The resulting user profile may represent either the commonality between the two models or the differences. An advantage of this style of modeling is that it allows a certain degree of diagnostic activity on the part of the system. Not only is the system aware of what the user is doing, but it also has a representation of optimal behavior. This provides a benchmark against which to measure the user's performance, and, if the user does not take the optimal course of action, gives an indication of the type of help or hint that is required.

A similar approach is used in error-based models where the system holds a record of known user errors and the user's actual behavior is compared with these. If this behavior matches an error in the catalog, then remedial action can be taken. Potential errors may be matched when partially executed and help given to enable the user to avoid the error, or recover more quickly. These types of modeling are also useful in intelligent tutoring systems where diagnostic information is required in order to decide how to proceed with the tutorial.

11.4.2 Knowledge representation: domain and task modeling

All adaptive help systems must have some knowledge of the system itself, in order to provide relevant and appropriate advice. This knowledge may include command use, common errors and common tasks. However, some help systems also attempt

to build a model of the user's current task or plan. The motivation behind this is that the user is engaged in a particular problem-solving task and requires help at that level. Generic help, even adapted to the expertise and preference of the user, is not enough.

One common approach to this problem is to represent user tasks in terms of the command sequences that are required to execute them. As the user works, the commands used are compared with the stored task sequences and matched sequences are recovered. If the user's command sequence does not match a recognized task, help is offered. This approach was used in the PRIAM system [85].

Although an attractive idea, task recognition is problematic. In large domains it is unlikely that every possible method for reaching every possible user goal could be represented. Users may reasonably approach a task in a non-standard way, and inferring the user's intention from command usage is not a trivial problem. As we saw in Chapter 9, system logs do not always contain sufficient information for a human expert to ascertain what the user was trying to do. The problem is far greater for a computer.

Assistants and agents use task recognition at a basic level to monitor user behavior and provide hints and macros when a familiar or repeated sequence is noticed (see Section 11.3.6).

11.4.3 Knowledge representation: modeling advisory strategy

A third area of knowledge representation, which is sometimes included in adaptive help, is modeling advisory or tutorial strategies. Providing a help system with this type of information allows it not only to select appropriate advice for the user but also to use an appropriate method of giving advice.

As we have already seen, people require different types of help depending on their knowledge and circumstances. These include reminders, task-specific help and tutorial help. There is evidence to indicate that human experts follow different strategies when advising colleagues [293]. These include inferring the intention of the person seeking help and advising at that level or providing a number of solutions to the person's problem. Alternatively they may attempt to place the problem in a context and provide a 'sample solution' in that context.

Few adaptive help systems have attempted to model advisory strategy, and those that do provide a limited choice. Ideally, it would be useful if the help system had access to a number of alternative strategies and was able to choose an appropriate style of guidance in each case. However, this is very ambitious – too little is known about what makes a guidance strategy appropriate in which contexts. However, it is important that designers of adaptive help systems give some thought to advisory strategies, if only to make an informed choice about the strategy that is to be used.

The EuroHelp adaptive help system adopts a model of teacher-pupil, in which the system is envisaged as a teacher watching the user (pupil) work and offering advice and suggestions in an 'over-the-shoulder' fashion [126, 44]. In this case, instruction

may be high to begin with but will become less obtrusive as the user finds his feet. The user is able to question the system at any point and responses are given in terms of the current context.

This mixed-initiative dialog is also used in the Activist/Passivist help system, which will accept requests from the user and actively offer suggestions and hints, particularly about areas of functionality that it infers the user is unfamiliar with [133].

11.4.4 Techniques for knowledge representation

All of the modeling approaches described rely heavily on techniques for knowledge representation from artificial intelligence. This is a whole subject in its own right and there is only room to outline the methods here (although some of the techniques are based on theories of memory and problem solving as discussed in Chapter 1). The interested reader is also referred to the text on artificial intelligence in the recommended reading list.

There are four main groups of techniques used in knowledge representation for adaptive help systems: rule based, frame based, network based and example based. Note that these general techniques are often combined to produce hybrid systems.

Rule-based techniques

Knowledge is represented as a set of rules and facts, which are interpreted using some inference mechanism. Predicate logic provides a powerful mechanism for representing declarative information, while production rules represent procedural information. Rule-based techniques can be used in relatively large domains and can represent actions to perform as well as knowledge for inference. A user model implemented using rule-based methods may include rules of the form

```
IF
  command is EDIT file I
AND
  last command is COMPILE file I
THEN
  task is DEBUG
  action is describe automatic debugger
```

Frame-based techniques

Frame-based systems are used to represent commonly occurring situations and default knowledge. A frame is a structure that contains labeled slots, representing related features. Each slot can be assigned a value or alternatively be given a default value. User input is matched against the frame values and a successful match may cause some action to be taken. They are useful in small domains. In user modeling the frame may represent the current profile of the user:

User

Expertise level: novice
Command: EDIT file I
Last command: COMPILE file I
Errors this session: 6
Action: describe automatic debugger

Network-based techniques

Networks represent knowledge about the user and system in terms of relationships between facts. One of the most common examples is the semantic network. The network is a hierarchy and children can inherit properties associated with their parents. This makes it a relatively efficient representation scheme and is useful for linking information clearly. Networks can also be used to link frame-based representations.

The compile example could be expanded within a semantic network:

CC is an instance of COMPILE
COMPILE is a command
COMPILE is related to DEBUG
COMPILE is related to EDIT
Automatic debugger facilitates DEBUG

Example-based techniques

Example-based techniques represent knowledge implicitly within a decision structure of a classification system. This may be a decision tree, in the case of an inductive learning approach such as ID3 [298], or links in a network in the case of neural networks. The decision structure is constructed automatically based on examples presented to the classifier. The classifiers effectively detect recurrent features within the examples and are able to use these to classify other input. An example may be a trace of user activity:

EDIT file I
COMPILE file I

This would be trained as an example of a particular task, for example DEBUG.

11.4.5 Problems with knowledge representation and modeling

Knowledge representation is the central issue in adaptive help systems, but it is not without its problems. Knowledge is often difficult to elicit, particularly if a domain expert is not available. This is particularly true of knowledge of user behavior, owing to its variability. It is especially difficult to ensure completeness and correctness of the knowledge base in these circumstances. Even if knowledge is available, the amount of knowledge required is substantial, making adaptive help an expensive option.

A second problem is interpreting the information appropriately. Although the knowledge base can be provided with detailed knowledge of the expected contexts and the domain in advance, during the interaction the only information that is available is the system log of the user's actions. As we saw in Chapter 9, interpreting system logs is very difficult because it is stripped of much context and there is no access to the user's intention or goal (except by inference). However, this data is not arbitrary and does contain recurrent patterns of activity, which can be used with care to infer task sequences and the like. However, it should be realized that these represent approximations only.

11.4.6 Other issues

Other issues that should be considered in designing an adaptive help system are initiative, effect and scope.

Initiative A major issue in adaptive help system design is that of initiative and control: should the user retain complete control over the system; should the system direct the interaction; or should a mixed dialog be supported? System activity can be intrusive to the user, particularly if badly handled. No user wants to be constantly told he is not performing a task in the most efficient manner! However, we know that there are normally large sections of system functionality of which the user is simply not aware. Without some form of system activity this problem will not be addressed. The solution seems to be to encourage mixed initiative in the interaction. The user should be able to question the system at any time, and the system can offer hints to the user. However, the latter should be offered sensitively and the user always allowed to continue as before if he wishes.

Effect Another issue that the designer should consider is the effect of the modeling and adaptivity: what exactly is going to be adapted and what information is needed to do this? All too often modeling systems use vast resources producing a detailed profile of the user, the bulk of which is never used. Modeling, whether of the user, the domain or strategies, should be directed towards the requirements of the help system. For example, if it is simply to offer different help to novices and experts, the system does not need details of task execution. Such considerations may reduce the overheads of adaptive help systems and make them more viable.

Scope Finally, the designer must consider the scope of the help: is it to be offered at an application level or system wide? The latter may be the ideal but is much more complex. If users are to be modeled at a system level, the model should take into account the levels of activity in which they are engaged and be able to distinguish actions at an application level. In many systems it would also have to cope with interleaving of activities and concurrent execution. Each of these makes the modeling activity more complex.

11.5 DESIGNING USER SUPPORT SYSTEMS

There are many ways of providing user support and it is up to the designer to decide which is most appropriate for any given system. However, there are a number of things which the designer should take into account. First, the design of user support should not be seen as an 'add-on' to system design. Ideally, the help system should be designed integrally with the rest of the system. If this is done, the help system will be relevant and consistent with the rest of the system. The same modeling and analytic techniques (for example, task analysis, see Chapter 15) used to design the system can guide the design of support material as well. Secondly, the designer should consider the content of the help and the context in which it will be used before the technology that will be required. Obviously, available technology is an important issue. However, concentrating on the task and the user will help to clarify the type of help required within the constraints of technical resources. Viewing the process in reverse may prevent the designer seeing beyond the technology she is familiar with. Bearing in mind the expected user requirements, the designer of help also needs to make decisions about how the help will be presented to the user and how this will be affected by implementation issues.

11.5.1 Presentation issues

How is help requested?

The first decision the designer must make is how the user will access help. There are a number of choices. Help may be a command, a button, a function which can be switched on or off, or a separate application. A command (usually) requires the user to specify a topic, and therefore assumes some knowledge, but may fit most consistently within the rest of the interface. A help button is readily accessible and does not interfere with existing applications, but may not always provide information specific to the user's needs. However, if the help button is a keyboard or mouse button, it can support context sensitivity, as we saw earlier. The help function is flexible since it can be activated when required and disabled when not. The separate application allows flexibility and multiple help styles but may interfere with the user's current application.

How is help displayed?

The second major decision that the designer must make is how the user will view the help. In a windowed system it may be presented in a new window. In other systems it may use the whole screen or part of the screen. Alternatively, help hints and prompts can be given in pop-up boxes or at the command line level. The presentation style that is appropriate depends largely on the level of help being offered and

the space that it requires. Obviously, opening a manual page line by line is unhelpful, as is taking over the whole screen to give the user a hint. Some active help systems provide visual cues when they have a suggestion to make (for example, an icon may be highlighted) – this gives the user the option of taking the suggestion without forcing him to abandon or interrupt his work. Again this decision should take account of the rest of the design, and aim to provide consistency.

Effective presentation of help

Help screens and documentation should be designed in much the same way as an interface should be designed, taking into account the capabilities and task requirements of the user. No matter what technology is used to provide support, there are some principles for writing and presenting it effectively. Help and tutorial material should be written in clear, familiar language, avoiding jargon as much as possible. If paper manuals and tutorials exist, the terminology should be consistent between these and the online support material. Instructional material requires instructional language, and a help system should tell the user how to use the system rather than simply describing the system. It should not make assumptions about what the user knows in advance. For example, a help message on the use of windows might read

To close the window, click on the box in the top right-hand corner of the window.

rather than

Windows can be closed by clicking on the box in the top right-hand corner of the window.

An exception to this is in documentation where the intention is not only to instruct the user in how to use the system but to record a full description of the system's functionality. However, documentation should be presented so that information is readily accessible, and should present both instructional and descriptive information clearly. The physical layout of documentation can make a difference to its usability. Large blocks of text are difficult to read on screen, for example. This can be alleviated by breaking the documentation into clear logical sections, or by using technology such as hypertext to organize it. A useful style is to provide a summary of the key information prominently, with further information available if required. This can be done either by devising a hierarchical help system where each layer in the hierarchy provides increasing detail, or simply by using layout carefully. An index can be used as a summary of available topics but should be organized to reflect the functional relationships between the subjects rather than their alphabetic ordering. Consistency is also important here – each topic in the documentation should be described using the same format so that the user knows where to look for a particular type of information. Documentation and help may contain definitions, descriptions, examples, details of error messages, options and instructions. These should be clearly recognizable.

11.5.2 Implementation issues

Alongside the presentation issues the designer must make implementation decisions. Some of these may be forced by physical constraints, others by the choices made regarding the user's requirements for help. We have already considered how help may be requested and how it appears to the user. Obviously each of these decisions involves implementation questions: will help be an operating system command, a meta-command or an application? What physical constraints does the machine impose in terms of screen space, memory capacity and speed? Speed is a very important consideration, since an unacceptably slow response time is liable to make the system unusable no matter how well it has been designed. It is better to provide a simple help facility that responds quickly than a sophisticated, intelligent one that takes minutes to provide a solution.

Another issue the designer must decide is how the help data is to be structured: in a single file, a file hierarchy, a database? Again this will depend on the type of help that is required, but any structure should be flexible and extensible – systems are not static and new topics will inevitably need to be added to the help system. The data structure used will, to an extent, determine the type of search or navigation strategy that is provided. Will users be able to browse through the system or only request help on one topic at a time? The user may also want to make a hard copy of part of the help system to study later (this is particularly true of manuals and documentation). Will this facility be provided as part of the support system?

Finally, the designer should consider the authors of help material as well as its users. It is likely that, even if the designer writes the initial help texts, these will be extended by other authors at different times. Clear conventions and constraints on the presentation and implementation of help facilitate the addition of new material.

11.6 SUMMARY

This chapter has been concerned with user support in the form of help and documentation. No interactive system of any complexity is so intuitive that the user never requires help. Help should therefore be an integral part of the design. Users require different types of help, depending on the context and circumstances, and the user support facilities should support these. Different styles of help support different requirements and different types of user. We have considered several types of help system, including adaptive user support. It is important to select a support style and design user support with the user in mind, just as the design of the system is user centered. In particular, the presentation of help should take into account usability principles, and the language should be clear and instructional.

EXERCISES



- 11.1 Write a manual page for making a cup of coffee. Assume your user has no experience but will recognize a cup, a kettle, a spoon, etc. Swap your manual with a partner. Does your partner's manual give you sufficient instruction to make the cup of coffee? Discuss improvements with your partner and agree on a final version of the manual.
- 11.2 Find a computer application that you have never used before. Attempt to learn to use it using only the online support. Is there enough information to allow you to use the application effectively? Is the information easy to find? What improvements (if any) would you suggest?
- 11.3 What knowledge is needed to build an adaptive help system? Which do you think is most difficult to provide and why?
- 11.4 Look at as many online support systems as you can. Which do you find most useful and why? Try to assess them using the requirements discussed in Section 11.2.
- 11.5 Using your library facilities and the world wide web, investigate the benefits and limitations of adaptive help systems. What examples of adaptive and adaptable help are available and how useful are they?
- 11.6 What are the four main types of help that users may require? For each type, give an example of a situation in which it would be appropriate.
- 11.7 Which usability principles are especially important in the design of help systems, and why?
- 11.8 Describe some of the different approaches to providing user support systems, with examples.
- 11.9 Applications are often supported by an online version of the paper documentation; in some cases there is no paper documentation at all.

What are the advantages of online documentation? What are the disadvantages, and how can they be overcome?
- 11.10 Discuss the presentation issues involved in the design of effective and relevant help systems.

RECOMMENDED READING

- R. C. Houghton, Online help systems: a conspectus, *Communications of the ACM*, Vol. 27, No. 2, February 1984.
A good review of non-adaptive help systems, most of which is still relevant today.
- C. Turk and J. Kirkman, *Effective Writing*, 2nd edition, E. and F. N. Spon, 1989.
An excellent introduction to technical writing, including writing instructional material.
- E. Rich and K. Knight, *Artificial Intelligence*, 2nd edition, McGraw-Hill, 1991.
A detailed text on artificial intelligence techniques. Readers should select appropriate sections on knowledge representation.
- Clippy's home page: www.microsoft.com/Office/clippy/.
An interesting case study on how help can fail – and how failure can be turned to advantage. Accessed March 2003.

MODELS AND THEORIES

PART

3

In all engineering disciplines, the designer recruits a selection of models to contribute to the design process. If we were building a new office block, for example, then we would use models of air circulation to design the ventilation system, structural models for the fabric and possibly social models for the detailed design of the office layout.

Models are used in other disciplines too. We may analyze the structure of a piece of music and decide that it is a rondo, or say that a poem is in sonnet form. Further, we may deliberately set out to write a sonnet, thus imposing the model upon the creative process. Craft is the art of design within constraint, and models help to formulate the constraints.

The chapters in Part 3 describe a range of models that can be used during the interface design process. Just as in the design of the office block several different types of model are required for different aspects of the building, so in interface design we would expect to use a whole selection of complementary methods.

Chapter 12 considers models with psychological or cognitive origins, where the emphasis is on formulating aspects of user behavior such as goal formation and problem solving. Chapter 13 discusses socio-technical models that attempt to describe the user within a social and organizational context, while Chapter 14 looks at models of collaboration and group interaction. Chapter 15 describes task analysis techniques for determining the relevant actions a user performs in some work domain. Chapter 16 is concerned with dialog description techniques used to specify and analyze the communication between user and system. Chapter 17 describes the use of general mathematical notations used in software engineering to specify and analyze abstract descriptions of interactive systems, and Chapter 18 extends these notations to model rich interactions.

