UNIX Guide: lessons from ten years' development

P. J. Brown
University of Kent at Canterbury
Computing Laboratory, The University, Canterbury, Kent CT2 7NF, England
pjb@ukc.ac.uk

Abstract

Development of the Guide hypertext system has been progressing at the University of Kent since 1982. The paper looks back over the mistakes and successes of the last ten years, with a view of drawing some lessons for the future development of hypertext. The reader is not assumed to be a Guide user, and the lessons learned apply to hypertext systems in general.

1 Introduction

Development of Guide began, using the UNIX[1] operating system and PERQ hardware, in 1982. The aim was not to build a hypertext system *per se* — I was unaware at the time of the work and ideas of such people as Engelbart and Nelson — but simply to produce a system which aided the reading of documents from computer screens. Documents to be read at computer screens were then largely direct reproductions of paper documents. (They still often are.) It would be a huge coincidence if the ideal form for a graphics screen was the same as that for paper.

This application-oriented aim of Guide has been a healthy influence throughout. It is one of the reasons Guide stands out as different from the mass of hypertext systems. The differences are, of course, sometimes positive and sometimes negative, but the more there are different species to cross-fertilise, the better it is for the overall development of a subject.

[1]UNIX is a trademark of Bell Laboratories

2 OWL involvement

This paper is based on the development of Guide at the University of Kent, but the most significant event in Guide's development was the involvement of OWL (Office Workstations Ltd) from 1985 onwards. OWL developed the original ideas in order to produce Mac and PC products. These have had wide market penetration and this has led to a continuous flow of new ideas.

Work at Kent has continued in parallel with OWL's work. The focus at Kent has always remained as UNIX workstations; in the rest of this paper we will refer to this implementation as *Guide-u*.

3 The same or different?

A difficult choice for any software developer is:

(a) to base the software on ideas that are already familiar to the reader, in order that the software will be readily accepted.

(b) to base the software on the ideas that are the natural ones for the application.

The book metaphor in hypertext systems is a prime example of case (a). It has been widely adopted by hypertext systems designed to cater for a lot of casual use, such as the excellent system at the London National Gallery[5]. For longer term usage paper-based metaphors such as books are almost certainly losers, but for short term usage they are winners.

The software developer faces an exactly similar choice at a lower level, that of the style of user interface. The safe choice is to follow the house style of a particular platform. The house style is particularly strong on the Mac, but until recently has not been strong on UNIX. The radical choice is to change the accepted user interface style to fit the perceived needs of hypertext.

Two key elements of the user interface style are the menu and the scroll-bar. Both Guide-u and OWL's Guide have been cautious and have adopted a conventional scroll-bar in order to fit in with the accepted style. It is not a disaster, but it is certainly not the right mechanism. A scroll-bar is designed for a linear document that is largely static. The Guide model of hypertext has a scroll that contains *replace-buttons*: when the user selects a replace-button it is replaced by the text (and/or graphics) folded behind it. For instance the following Guide document contains a replace-button (shown in bold-face) called *Example*:

.. useful feature.
Example
Moreover, ..

If this replace-button was selected the text document might appear as

.. useful feature.
<A 10-line example>
Moreover, ..

Generally, selecting a replace-button causes the scroll, as the reader sees it, to grow in size; indeed most Guide hyperdocuments initially appear as a small scroll, and the reader selectively expands it. The effect is that the scroll-bar behaves erratically: a point that was half-way through the scroll as displayed may, if a big replacement is unfolded in the latter part of the hyperdocument, suddenly appear to be only one quarter the way through the scroll. Instances such as this play havoc with the scroll-bar; nevertheless there has been no serious candidate to replace it.

Though cautious and unimaginative with its scroll-bar, Guide-u's approach to its menu is one of its more radical and more successful features. Guide-u has complete flexibility: any Guide document can be selected to act as the menu. (Here is not the place to go into elaborate detail, but a key point is that Guide buttons can have actions, such as Quit, attached to them. These figure prominently in documents used as menus. Indeed the standard menu consists only of a sequence of such actions, but user-defined menus often use a subset of the standard actions, interspersed with explanatory text, pictures, logos, and other buttons.) This has two consequences:

- each application can have its own menu designed for the task in hand. *All serious Guide-u applications have used their own menu rather than relying on the standard all-purpose one.*
- Guide-u is self-compatible: the menu behaves in essentially the same way as the document that the user is reading. The downside is that the Guide-u menu is *not* compatible with styles such as Motif.

Menus are particularly important in scroll-based systems such as Guide. In a card-based system, many of the functions of the menu can be taken over by buttons that appear in each card (e.g. go-to next card, go-to some base point). In a scroll-based system, where such buttons could move around the screen or even off the screen as a result of scrolling, such an approach is undesirable, and thus the true menu is more important.

In summary, if you want your system to be used for real applications, you can probably only afford to be radical in one element of a user interface. Guide-u's radical approach to the menu has been a success (though with some downside), but was there a bigger wasted opportunity with the scroll-bar?

4 Using what is there

As well as fitting in with existing metaphors and an existing user interface style, a hypertext system will be under pressure to fit with existing tools, too. To survive, any hypertext system needs to interact with and to use other tools[4]. A choice facing the designer is which of these alternatives to select:

(a) to use what is there already, i.e. to use the existing command languages, editors for various media, spelling checkers, printing programs, searching tools, etc.

(b) to provide a framework by which others can plug extra tools, specially written to certain conventions, into the hypertext system.

An example of case (b) is Quill[8], which is a novel document editing system. Quill consists of a number of co-operating editors, co-ordinated by the Quill 'shell', which maintains a consistent user-interface across the editors. Each editor needs to be specially written to fit in with Quill.

Clearly approach (b) is more attractive for a researcher keen to escape from the dreary current world into a brave new world. However it requires

writing enough supporting tools to make the system viable. Then, with luck, the system will take off, and users all over the world will add supporting tools to it.

For a small project, the developer or researcher needs to grit his teeth and adopt approach (a): fitting in with the current imperfect world and making use of it. Hypertext is an applied topic, and it is vital for researchers that their new systems find real authors who will write hyperdocuments that find real use. There are many research-oriented hypertext systems whose only use has been a few hyperdocuments produced at the site where the system was developed — in numerous research papers these same examples are quoted again and again.

Guide-u therefore adopted approach (a).

5 Influence of UNIX

The philosophy of UNIX is that you do not build huge monolithic tools, but instead rely on collections of smaller tools, all good at their own thing. Although this philosophy may have weakened over the years, it remains a reason for the success of UNIX.

An early mistake of Guide-u was to encode source files in an opaque form, with lots of embedded binary codes. The result was that other UNIX tools could not work with Guide files. The mistake was rectified by going to a textual mark-up. The chosen format was similar to *troff* requests. For example a button is encoded as

.Bu <*attributes of button*>
Button-name
.bU

(An enduring mistake is that pictures, which can be embedded in Guide-u documents, *do* still contain binary codes. In Guide-u's mark-up, these pictures have been made to look like *troff* comments, so that they do not confuse any utility (e.g. *spell*) which uses *deroff* to strip out *troff* mark-up. The embedded binary codes can, however, confuse utilities such as the *vi* editor and the mail system.)

This mark-up has been a success of Guide. One can, of course, argue whether an SGML style would be better than the *troff* style, but the role of the *troff* style as something of a standard for UNIX tipped the balance. An example of its value is that if one wanted to analyse the style of a Guide-u document by counting the number of occurrences of each hypertext feature, it could be done with a short UNIX shell-script involving commands such as *grep*, *sort* and *uniq*.

A similar example is a script to perform a systematic edit such as prefixing some fixed symbol to all button-names of a certain type.

Embedded mark-up such as Guide's has its limitations, and the current fashion is to keep the structuring information separate from the material to which it applies. Nevertheless it is not something I feel keen to change, as, in conjunction with the current array of UNIX tools, it offers simplicity and power.

6 WIMP interface

When you start building a hypertext system, you imagine you will spend your time creating hypertext functionality. In fact you spend most of the time working on aspects of the WIMP interface. In a recent talk, Bob Anderson of Xerox EuroPARC said that the user-interface proportion of a modern application program was 85% and rising. Though others make more modest claims than this, the consensus figure is still over 50%.

It is particularly important in a hypertext system to encourage readers to explore, but if they are to do this they must have a natural and painless way of going backwards, and undoing the effect of anything that turned out to be a false step. They must also be able to abort operations halfway through, with no harm resulting.

As anyone who has implemented such systems will know, this requires lots of planning, careful implementation, and, if the user is to be properly informed about what is going on (with subtle graphic effects, etc.), lots of lines of implementation code.

A mistake in the design of Guide-u, which still shows scars, is that Guide tried to cater for two interface styles at once. There were hugely more 'glass teletype' screens than graphics screens in the world in 1982, and many remain today. Guide-u tried to cater for both: the idea was to have a display module that could be written either for a character-array screen with function-key input, or a graphics screen with a mouse. You cannot do both well. The designer has to make the tough decision to go for a particular configuration and make sure his software is right for this.

An example of a scar from Guide-u's attempt to have its cake and eat it, is that there is no facility for dragging the mouse to select a sequence of

characters (e.g. as the subject of a cut or copy): this was felt unsuitable for a glass-teletype terminal. The philosophy of no multiple selection has become engrained in Guide-u's design and is now hard to change. (OWL Guide avoided this trap.)

7 Communication and pipes

UNIX's simple primitive for communicating between tools is the pipe. A pipe allows a one-way communication of a stream of bytes between one tool and another.

There is no equivalent of a pipe to allow two-way communication between tools in a highly interactive environment. Guide-u has gained hugely from working with other UNIX tools but all the successful examples have involved communication that is not highly interactive.

Guide-u allows UNIX shell-scripts to be attached to buttons: when the button is selected, the script is run and the output is piped back into Guide and becomes the replacement of the button. To the user the button appears like any other, and the user can be unaware that a program has run. This facility has been successfully used to provide active documents, e.g. a button might run a report-generating program which, say, returned a summary of items currently in stock.

The shell-script facility, at a stroke, added infinite programmability to Guide-u. In addition Guide-u has its own small programming language, like HyperTalk in HyperCard, or Logiix in OWL Guide, but much weaker. This language allows most of the actions that the user might do (selecting a button, using the menu, typing text) to be reproduced. For example a 'program' in this language might say 'Find the string XX and then Type the string YY': such a program could be attached as an action to be done when a button was selected. The language is so small that it does not merit a name, but, for convenience, we will call it *P* here. Shell-scripts initiated from Guide can use a utility called *callguide* which passes Guide a P program to execute. Thus a shell-script might take the form

```
if XXX
    callguide with program P1
else
    callguide with program P2
```

The mistake in the design of Guide-u was to assume that the existence of P, plus the infinite programmability provided by shell-scripts, was a short cut to making Guide-u fully programmable. The problems are two-fold.

The first and lesser problem is notational. Intermixing P programs with bits of shell-scripts leads to poor readability since two separate notations are closely inter-twined. At the lowest level there are often several levels of nested quotes, which are hard to get right.

The larger problem concerns communication. If one looks at the condition XXX in the above example, the author might well want this condition to be 'if the Guide-u user has selected button BB'. In other words, the shell-script needs to know intimately about Guide-u's current state. The problem is that it does not: Guide-u provides a small amount of information through environment variables and the like, but it only provides a tiny proportion of the information that may be needed. It is simply not possible for two independently written programs, in this case Guide-u and a shell-script written by a user, to examine each other's internal states.

To summarise, it is moderately easy to accept existing tools as helpful friends, but it is desperately difficult to make them intimate colleagues.

8 Constraining authorship

Certain user interface toolkits are described as 'policy-free', meaning that they do not impose a style on the user-interface. I believe that hypertext systems should be policy-free in that they do not impose a style of authorship. We are a long way from finding out what the best hypertext authorship styles are, and any system that restricts authors is holding back development.

Guide set out from the start to be policy-free, but has not always realised its aim successfully. A success has been the enquiry (now called a *group* in OWL Guide). This allows the author to specify an arbitrary *source region*, i.e. a region surrounding a replace-button; when the button is selected, the source region is replaced by the button's replacement. At one extreme the author can design enquiries so that the region is always the whole screen: in this case Guide behaves like a page-based system such as HyperCard. At the other extreme, the author who does not use enquiries at all will have a pure scroll model — what is replaced when a button is selected is just the button-name. There is plenty of scope in between these extremes: one of the most successful Guide-

u applications, the LOCATOR project[6], uses enquiries to present information so that the first part of the screen gives a historical trail and the lower part, an enquiry, gives the current information.

The enquiry is a good example of a construct that aids a policy-free style. Other constructs in Guide-u have unfortunately allowed policy to creep in. For example the *glossary-button* in Guide-u allows the user to select a button that places information in a separate sub-window called the *glossary*. This is reasonable, but what was wrong was to attach a policy that said the glossary was always put in alphabetic order. Thus if the user selects three glossary-buttons in sequence the three items are placed in the glossary in alphabetical order. Generally the screen is not big enough to see the whole glossary, and the effect to the user is that the glossary leaps about in an apparently random way. Readers — and even authors — often have to be told after the event the alphabetical-order rule, as an explanation of what has been happening.

Another failure was a tendency to bundle features together in packages rather than to provide the underlying primitives. For example two aspects of buttons are;

(a) where the replacement of a button is to go (e.g. in-line or in the glossary, or perhaps in a separate window).

(b) how the replacement is generated (e.g. using a fixed piece of text, using a separate file, using a definition from the library, using the output from a shell-script — here a success of Guide-u is that it insulates the reader from the details of how material is stored, or indeed whether it is stored at all).

Guide-u provides various button-types that cover certain combinations of (a) and (b); some combinations are not available. The problem arose because of piecemeal additions of various button-types to cater for perceived authorship needs, rather than a proper analysis of underlying primitives.

The moral is that if you really do not know what authors need, stick to primitive operations, and if you get something wrong do not try to patch your way out of it.

9 Logical structure

We have argued for a set of primitive operations below the policy level. Such primitives need to be complemented by a high-level logical approach that captures the nature of a document in a manner *above* and independent of the policy for hypertext presentation. This logical structure can be mapped into a style of hypertext presentation by a table or similar mechanism. For each logical object a table entry specifies, in terms of the primitive operations, how to display it. For example the logical object could be 'refinement of detail', and this could be represented by a hypertext button.

The same considerations apply to document preparation in general. It is, for example, the accepted wisdom that documents for a word-processor or formatter should be marked up in terms of their logical structure rather than their appearance.

The problem is that, although standards and models for documents (SGML, HyTime, Dexter, etc.) apply at the logical level, most systems that are in use in the field allow or even encourage authors to think in terms of appearance rather than logical structure.

Guide-u is still partly guilty here. On the plus side, authors *cannot* specify documents at the level of fonts, character sizes, colours, etc. Instead they are constrained to use logical objects[2]. For example, the author says 'this is a book title', not 'this is italic text'; a table entry for 'book title' might specify that tables are to be represented in, say, italic text, and in a certain colour if it is available.

On the negative side, there are no logical abstractions above the 'button' level. For example the Guide author creates hypertext structure at the level of, for example, a replace-button. In creating a replace-button the author says 'this is a button to be replaced in-line by XYZ'. Logically the button may represent, for example, a refinement of detail, a division of a management structure or a report (generated by running a program). This information is never captured. The loss is huge. For example the lost information would be invaluable to:

- any author who was maintaining a document written two years earlier by another author. To do a decent maintenance job, you need to understand the logical structure: the 'why' rather than the 'how'.
- anyone charged with the task of producing a program to take a hyperdocument and produce a paper document that captured the hyperdocument at its lowest level of detail.

- readers. Although, in most instances, it is better if the logical structure only manifests itself to the reader through the appearance of the document (e.g. a reader should recognise a heading without being told 'This is a heading'), there are cases where a direct awareness of the document's structure helps the reader's understanding. In some hypertext systems, for example, readers are made aware of different types of links.

- a user who wanted to perform a structural search (see later discussion of the Find command) to show all instances of buttons that represented reports.

10 No go-tos

Primitives do not have to be at the lowest level, even though they need to be policy-free. A link in a hyperdocument is a go-to, and go-tos are harmful[3]. Guide-u has always attempted to find higher-level primitives than the go-to. It has never had direct links embedded in its source file. Instead objects are linked together by matching names, in the same way as the name of a variable in a programming language links to a declaration of the variable[1]. (KMS and the Symbolics system are other hypertext systems that use name matching.) To take an example, assume that the hyperdocument is a reference work on birds and that the information on habitat for both the willow warbler and the chiff-chaff is the same. The way that this information is shared is that the Guide author would create the habitat information as a *definition* and give it a name, say 'willow warbler habitat'. The chiff-chaff entry would contain a button that uses this definition, i.e. the button's replacement would be specified as the definition called 'willow warbler habitat'. When the user selects this button the definition is copied so that it forms the replacement of the button (a 'come from' rather than a 'go to'). This gives the user an illusion of a hierarchy, rather than a jump to a different place in the hyperdocument. There is reasonable circumstantial evidence that this approach lessens the getting-lost problem.

Initially the no-go-to principle was extended to a ban on any form of Find command. A Find command was thought to be a particularly insidious form of go-to since it cuts across the structuring of the hyperdocument provided by the author. Nevertheless banning a Find command was a big mistake, as it pre-supposed an omniscient author who would provide a hypertext structure that would allow all readers to find exactly what they wanted without resorting to a Find command. Human nature being as it is, I did not admit the mistake and quickly correct it. Instead there has been a slow lessening of the wrong principle until today there is a reasonably powerful Find command (adapted to hypertext use since it covers both content and structure, though it could profitably be fitted into a general mechanism for computed links). OWL, being driven more by market needs than by dogma, have always had a good Find command in their Guide.

What the thinking should have been is: *information retrieval shows that readers find it hard to extract the information they need from a mass of documentation, and all existing mechanisms are far from perfect. Therefore the reader should have all the tools that can be reasonably provided, and if these tools are orthogonal to one another, all the better.*

11 Testing and maintenance

As hyperdocuments increasingly go into production use the cost of testing and maintenance will dominate the original authorship cost. Guide-u provides two simple mechanisms to help testing and maintenance:

(a) a mechanism for testing that all required definitions exist (thus, for example, that selecting the button that represents the habitat of the chiff-chaff does not lead to the error: 'definition not found').

(b) a way for authors to insert comments into a document in order to explain what they have done and why they have done it.

This is better than most hypertext systems provide (though the Symbolics Concordia system is one honourable exception), but it is pitifully inadequate: it is like saying that all problems of software engineering will be solved by (a) providing a syntax checker and a linkage checker for programs, and (b) allowing authors to put comments in their programs.

12 Tailoring

One of the benefits often claimed for hypertext is that, unlike paper, a hyperdocument can automat-

ically adapt according to the nature and needs of the reader and the environment. An example is a tutorial where the user states his level of expertise and the tutorial changes accordingly. Another example would be a hyperdocument about office procedures: the hyperdocument could change so that a salesman in subsidiary A saw a different hyperdocument from a director of subsidiary B.

I have never seen these benefits realised in practice except in simple ways. This is because it is hard enough to author a good document in the first place — irrespective of whether it is a paper document or a hyperdocument — and the extra overhead of providing a smooth adaptation over a wide spectrum of users is just too much.

Guide-u contains a lot of tailoring methods to cater for the perceived need. Two of these are:

(1) buttons can be automatically selected, either by the user, or by the system, or according to some Boolean expression (e.g. if file X exists). If such buttons are embedded in enquiries then this can act as a multi-way switch: show text 1 if ...; otherwise show text 2 if ...; otherwise show text 3.

(2) a feature for procedures with arguments.

These, particularly (1), have seen good use, but mainly for limited tailoring, done behind the user's back. They are used, for example, in the *Xtutor* hyperdocument[7], which is a guide for beginners to X windows. This product is portable to different sites and can adapt to cover differences in, say, the use of X windows mouse-buttons or file names. Xtutor wisely does not, however, try to adapt itself radically according to the expertise of the user.

My conclusion in this area is that too much effort has been devoted to tailoring facilities in Guide-u, and a few simple facilities is all that authors can reasonably cope with.

13 Conclusions

This paper has covered wide areas but its messages can be presented, in somewhat simplified form, as the following guidelines to developers of hypertext systems.

Do:

- design source formats and interfaces so that the system will work well with existing tools. More generally, try to work with the world as it is, rather than the world as you would like it to be.
- provide foundations that are above the go-to level.
- allow — even constrain — authors to work at the logical level, independently of the hypertext presentation.
- decide the style of hardware/software platform needed and stick with your choice.
- decide whether the system is for casual use, and thus needs familiar metaphors, or for professional use, and thus probably needs new metaphors.
- expect most of the implementation work to be concerned with the user-interface.

Do not:

- set out to build a hypertext system; instead set out to create a system to help authors and readers of on-line documents.
- impose a style of authorship.
- devote much effort to tailoring mechanisms.
- expect hypertext facilities to be enough to meet all the user's needs to extract information from hyperdocuments.

And, finally, *Do*:

- expect to enjoy the whole undertaking.

14 Acknowledgements

I would like to thank all the many people who have contributed to the development of Guide over the last ten years, especially at OWL and the University of Kent.

References

[1] P. J. Brown, A hypertext system for UNIX, *Computing Systems*, **2**, 1, 1989, pp. 37-53.

[2] P. J. Brown, Using logical objects to control appearance, *EP—odd*, **4**, *2*, 1991, pp. 109-117.

[3] L. De Young, Linking considered harmful, in Rizk, Streitz and André (Eds.), *Hypertext: concepts, systems and applications*, Cambridge University Press, 1990, pp. 238-249.

[4] N. Meyrowitz, The missing link: why we're all doing it wrong, in Bartlett (Ed.), *Text, context and hypertext*, MIT Press, 1989.

[5] B. Rubinstein, *The micro gallery at the National Gallery*, International Conference on hypermedia and interactivity in museums, Pittsburgh, Pa., 1991.

[6] G. W. Rouse, Locator — an application of knowledge engineering to ICL's customer service, *ICL Technical Journal*, **7**, *3*, 1991, pp. 546-553.

[7] W. Strang and G. Tardivel, *Xtutor hyperdocument*, in the public computer archive: *uk.ac.hensa.unix*, 1992.

[8] Y. Wolfsthal, Style control in the Quill document editing system, *Software—Practice and Experience*, **21**, 6, 1991, pp. 625-638.