ELECTRONIC PUBLISHING, VOL . 2(2), 91–100 (DECEMBER 1989)

Do we need maps to navigate round hypertext documents?

P.J. BROWN

Computing Laboratory
The University
Canterbury
Kent, CT2 7NF, UK

SUMMARY

In many hypertext systems users are provided with a map of the underlying directed graph of their hypertext document. Arguably this is like filling a program with goto statements and then placating the readers of the program by providing a map of all the gotos. In this paper we present an alternative approach which goes some way — but not the whole way — towards providing a hypertext user interface that distances the reader from the underlying directed graph.

KEY WORDS Hypertext Navigation Map Guide

INTRODUCTION

The object of hypertext is to represent a body of information in a form that captures all the inherent interlinks in the information. Readers can then peruse the information, following the links of their choice. The aim is that by these means the reader will more quickly be able to gain an understanding of the information, and extract the parts he wants.

There is debate about the extent to which information is inherently like an interlinked mass of spaghetti. However, a good insight has been provided by Brooks [1] who says that information *is* like spaghetti but the duty of a good author is to cut links so that, to a beginner, the information appears to have a simple hierarchical structure — the backbone — with a *few* cross-reference links that cut across the hierarchy. Only more advanced readers need be presented with a full gamut of cross-reference links. This view holds, incidentally, independent of whether the material is presented on paper or via hypertext.

If this view is accepted, a hypertext system must provide special support for hierarchical links, but should also support cross-reference links — as indeed most systems do. Furthermore if there are to be documents for advanced readers which contain thousands of cross-reference links, there should be mechanisms to tame the resultant complexity. On top of this, there is a requirement that a hypertext system must be economical in the number of facilities that readers need to master: if the system is to be used by casual readers, it is no good expecting them to read a 50-page user manual before they start.

REPRESENTATION OF HYPERTEXT

We shall start by considering the representation of information. A body of information

0894–3982/89/020091–10$05.00
Received 14 October 1988
Revised 17 November 1988

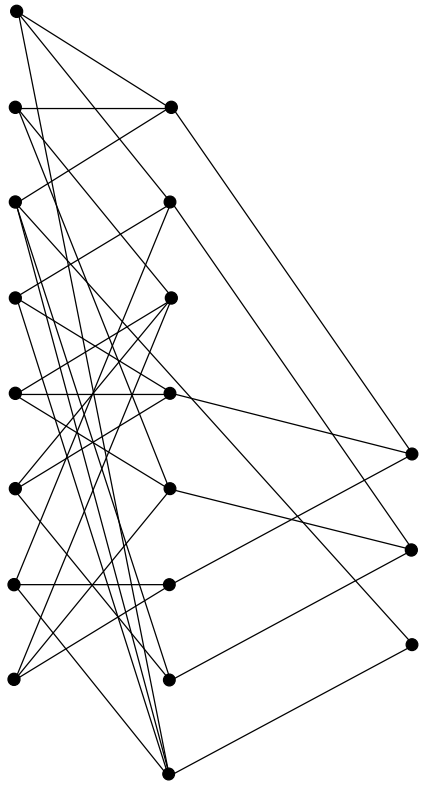

Figure 1. Spaghetti links

stored in a form suitable for hypertext processing is called a *hyperdocument*. There are at the moment no accepted standards for hyperdocuments, and each hypertext system has its own representation. Nevertheless the underlying principles are common in that the hyperdocument normally consists of a directed graph [2]. At the nodes of the graph are fragments of text, and the arcs represent links between the fragments. (Identical principles apply in hypermedia systems, where fragments at the nodes need not be textual. In this paper, however, we will talk in terms of text in order to simplify the discussion.) Even if authors make efforts to simplify the hyperdocument by cutting links, its structure soon becomes a challenge for the human mind. Figure 1 shows an extremely simple case: a set of eight leaf nodes with three alternative hierarchies, yet the picture is still a bit hard to understand. Conklin's [2] survey shows a directed graph representing a real document and its complexity is mind-boggling.

We thus have the well-known problem of 'getting lost in hyperspace'. This can trouble both authors and readers, but we shall start by examining the problem from the reader's viewpoint.

The first point to be made is that, although getting lost is often claimed to be a great problem, the evidence is largely circumstantial and conflicting. In some smallish applications it is not a major problem at all, as, for example, in Shneiderman's report [3] of the use of a hypertext encyclopedia in a museum; he cited unusually careful and

skilled authorship as a prime reason for suppressing the 'getting lost' problem. It is certainly true that the general standard of hypertext authorship is, because of inexperience, dreadful. Doubtless when, in ten year's time, we look back on current hyperdocuments we shall be horrified at the poor grasp that authors had of the medium. (A similar phenomenon applies, for example, to user manuals for software: the standard of presentation ten years ago was much worse than it is today — even though the art of writing such manuals has been evolving for forty years.) As time goes by, hypertext authors will learn good presentation techniques, using various cues to help the reader know where he is. These new techniques will be different from the techniques used by authors of paper documents, since the medium is so different. The reason for the currently conflicting reports on the seriousness of the 'getting lost' problem is doubtless due to variability in authorship skills.

Although improvement in authorship techniques is the prime way to tackle the 'getting lost' problem, it will not solve the problem. What it will do is raise thresholds: readers should have no trouble navigating round small to medium hyperdocuments, and the 'getting lost' problem will only be paramount in largish documents. Thus we need to continue to provide extra navigation aids for readers.

CURRENT METHODS

The current wisdom is that readers should be presented with a map of the hyperdocument. Usually this map is a representation of (part of) the directed graph that underlies the hyperdocument. Halasz [4] even claims that such a map is a requirement.

We are uncertain that this is wisdom. To understand why, it is best to take an analogy. In the sixties programs were full of goto statements, and it was found that such programs were hard to read. However, people did not attack the problem by creating tools which, when fed a program, created a map which showed where all the gotos went. (Perhaps fortunately, graphics terminals were extremely rare in those days so such an approach had severe practical difficulties. There were, nevertheless, some attempts to generate flowcharts automatically from programs.) Instead the problem was attacked more directly, and gotos were largely superceded by better mechanisms. In fact the goto was still present in programs in the sense that the underlying hardware that executed programs was based on gotos, but readers of programs — and indeed writers of programs — no longer had to be aware of this.

The purpose of this paper is to explore a step towards achieving a similar advance for readers of hyperdocuments. We must emphasize that it aims to be a step rather than a complete solution and that it is aimed at readers rather than authors of hyperdocuments — though every author is, of course, also a reader. The essential tenet of the approach is that a reader of a hyperdocument should not be made aware of the underlying structure of the document any more than the user of, say, a database or information retrieval system need be fully aware of its underlying structures. This same view has been expressed by Oren [5] of Apple Computer, who points out that 'getting lost in hyperspace' can be a bogus problem in that it relates to the creations of a piece of software rather than the real nature of the information presented.

The approach has been implemented within the Guide hypertext system [6] so we shall now give a brief description of Guide.

THE GUIDE HYPERTEXT SYSTEM

Guide is a hypertext system that has been the subject of research and development at the University of Kent at Canterbury since 1982. In 1984 Office Workstations Ltd (OWL) became interested and they have since produced successful implementations on the Macintosh and PC. The University has continued to develop Guide, and its rôle has been to try out new ideas whereas OWL's has been to produce successful products for the marketplace — though the market driven approach has itself certainly led to new ideas. The University's work has been based on UNIX[1] workstations, and here we will use the term *UNIX Guide* to mean the University's implementation rather than OWL's. The fact that it runs on UNIX rather than some other operating system is not, however, relevant to this paper.

Most hypertext systems are based on pages (often called cards) of information, i.e. each node of the directed graph is treated as a page. When the reader wants to go from one page to another, either the current page is overwritten by the new one or, on systems that support multiple windows, the new page may come up in a separate window. Guide takes a different approach from the majority. In essence the user sees a single scroll. Within this scroll are *replace-buttons*, and when the user selects a replace-button it is replaced *in situ* by the text associated with the button. To be exact, a *region* around the button is replaced: the region is normally just the button itself, but it could be a question (an *enquiry* in Guide's parlance) in which the button is embedded — or even a 'page' if authors prefer that style. For readers unfamiliar with Guide, Figure 2 shows a Guide

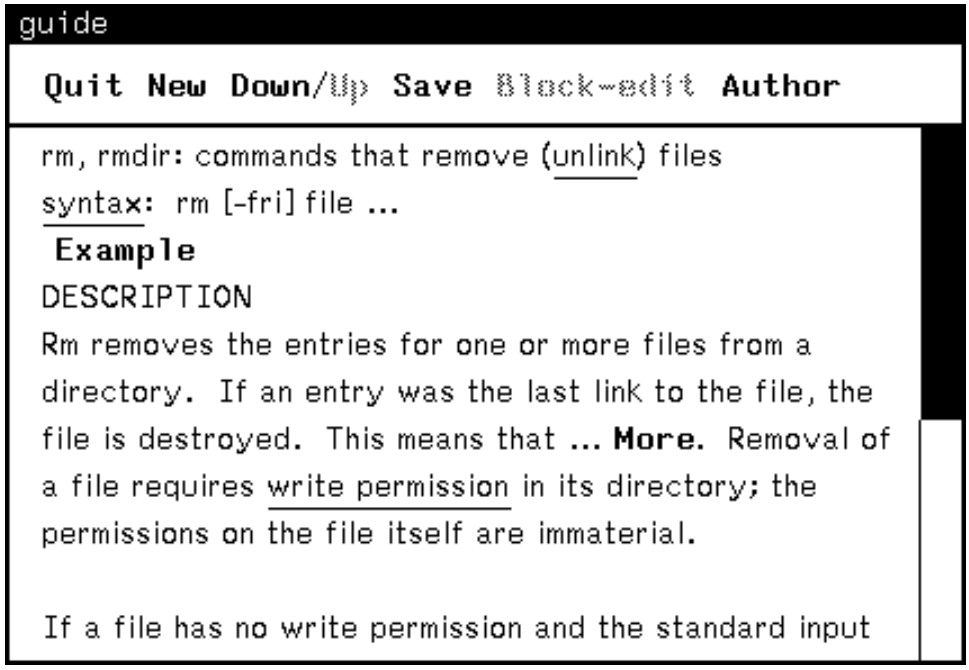

Figure 2. Guide viewing a document

[1] UNIX is a trademark of Bell Laboratories.

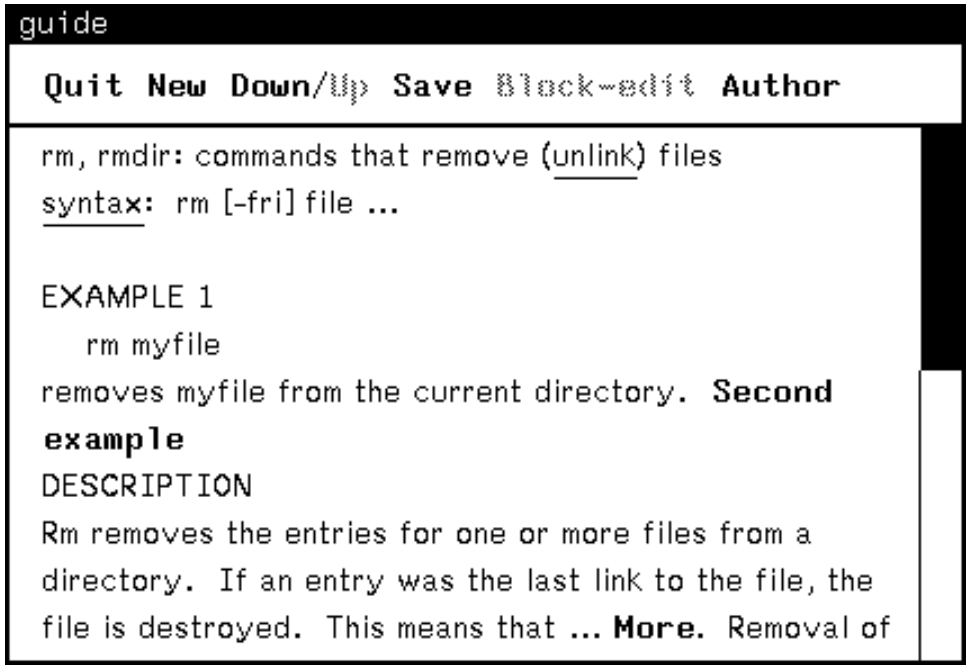

Figure 3. The result of selecting the **Example** *button in Figure 2*

scroll, and Figure 3 shows the same scroll after the replace-button **Example** has been selected. Replace-buttons are shown in bold to distinguish them from ordinary text.

This mechanism is therefore a step away from the goto: material is inserted in context rather than in a separate place. The replace-button is, however, only one of the linkage mechanisms in Guide. There are also mechanisms for out-of-line replacement (i.e. the new material comes up in a new window or sub-view) and for performing a goto (e.g. by refocusing the current scroll at a new position).

There are thus mechanisms both for hierarchical links, that are implemented by *in situ* replacement, and cross-reference links, that are implemented by bringing up a new window or refocusing the current one.

IN-LINE CROSS-REFERENCE

A strong hierarchical backbone certainly helps the reader to orient himself. Indeed Akscyn *et al.* [7] believe that this on its own makes the need for a map marginal. Inserting hierarchical material *in situ*, we believe, helps further. UNIX Guide, however, has taken one extra step down the path distancing the reader from the underlying directed graph: cross-referenced material can also be inserted *in situ*.

The approach is best illustrated by considering one of Guide's current applications. This is an application where it is used to display information on hardware faults, and to allow the reader, by selecting the replace-buttons relevant to his current problem, quickly to arrive at the section of the hyperdocument that diagnoses the fault. Initially the reader is presented with an enquiry containing a list of possible problems, e.g. 'Will not load', 'Printing problem', 'Hot smell', each represented by a replace-button. He selects one of

these and this typically leads to a further enquiry with several embedded buttons for selecting the reply; for example

Is the screen **Blank** or **Flickering** or **OK**?

The user continues answering questions until he reaches a point in the hyperdocument that suggests the cause of the fault. (There is, incidentally, no element of artificial intelligence: it is just walking a directed graph, in this case a non-cyclic one.)

In practice, of course, different paths through the document often converge. If, say, cables are faulty this may manifest itself to the user as 'Will not load', 'Blank screen', 'Drive will not start', etc. Whichever of these replace-buttons the user selects, he will eventually come to the part of the hyperdocument that deals with cabling problems.

In the old paper-based system on which this hyperdocument is based, the cabling problems in fact came under loading problems, and the other manifestations led to cross-references to this part of the manual.

The simplistic approach, therefore, is to make the hyperdocument ape the original paper document: i.e. to make the link to cabling from the 'Will not load' dialogue a hierarchical link and to make any other link to cabling a cross-reference link. This, however, makes no sense from the reader's viewpoint, since he sees all the paths to cabling as equally hierarchical. UNIX Guide provides a mechanism that supports the reader's natural view. The mechanism is a special sort of replace-button called a *usage-button*; this appears to the user just like an ordinary replace-button, but in fact gets its material by following a cross-refence link in the hyperdocument. The exact mechanism is that the author designates the material on cable faults to be a *definition*, and the usage-buttons make use of this definition. When the usage-button is selected the definition is copied and this acts as the replacement of the button.

MULTIPLE COPIES

We now move on to a more esoteric but nevertheless important point. If two usage-buttons share the same definition it may happen that the user has expanded both. This is unlikely, perhaps, in our example, where the user has a well-defined goal of following a path until the fault is diagnosed, but in applications where the user is exploring information it can, and does, happen. This brings up a design choice: are multiple copies independent of each other or are they treated as several views on the same information? There is no easy answer: the first approach is better for user-inserted bookmarks, since the user will not want lots of duplicate bookmarks, and the first approach is also normally better for marginal notes; for most kinds of editing, on the other hand, the second approach is preferable. The first approach is easier to implement unless an infrastructure that supports multiple views of information. e.g. the Andrew Toolkit [8] is available. UNIX Guide has chosen the first approach; it thus unequivocally maintains its hierarchical model to the user.

Users of UNIX Guide can edit their scroll and save the resulting hyperdocument, and thus the inherent disadvantages of multiple copies need to be faced. UNIX Guide's rule is that there is one master (the definition) and any other copies are transient copies of the definition (made at a certain moment and not subsequently changed if the definition changes). If a user tries to edit a transient copy he is given a warning message, since the results of the editing will not be saved. (To be exact, Guide offers two kinds of saving:

one saves the document with all its structure; the other saves the current instantiation. Transient copies are only lost on the structural save.) There are also mechanisms — currently somewhat imperfect — that help the user find where the definition is.

This approach seems to work well on those few occasions when the problem of duplicated information arises.

A DIFFERENT EXAMPLE

The purpose of usage-buttons is to reinforce the user's hierarchical view of the way he explores a hyperdocument and to disguise the underlying leaps round a directed graph. It turns out, however, that usage-buttons have other merits in that they open out new applications not normally associated with hypertext systems.

One example — a diversion from the main theme of this paper but perhaps of some interest in itself — is the exploration of a BNF definition of a language. For example the BNF definition of Pascal could be transliterated into corresponding Guide definitions. Given this, an author can take any Pascal language construct — here we will assume <while-statement> — and provide a usage-button corresponding to it. On selecting the **<while-statement>** usage-button, the user would see a copy of the corresponding definition, which is

while **<expression>** do **<statement>**

The user can instantiate the above by selecting either of the two replace-buttons within it. If, for instance, he selects the first, the above will be replaced by

while **<simple-expression>** | **<relational-expression>** do **<statement>**

Each replace-button in the above expansions is a usage-button which causes a copy of the corresponding definition to be inserted when the button is selected. Where there is an alternative, such as <simple-expression> or <relational-expression>, this is represented by a Guide enquiry. Guide allows recursion: indeed, selecting **<statement>** above would lead back to **<while-statement>** as one of the alternatives.

The whole arrangement, which is built entirely in terms of Guide's usage-buttons, definitions and enquiries, can be used by readers to get a feeling for a BNF definition of a language by investigating the various instantiations of each language construct. (The approach depends, incidentally, on Guide's property that multiple copies are independent of each other: most fully instantiated examples will contain many independent occurrences of such constructions as <expression>.)

UNDOING

In any application of hypertext, readers will need to go backwards as well as forwards. In particular they will want to undo (i.e. fold back) previous replace-button selections in order to go back to an earlier state and then, perhaps, explore other paths.

Guide's use of *in situ* replacement makes it possible to provide a particularly simple user-interface for this operation. The rule for the user is simply: 'if you want to get rid of anything, simply point at it and click the *undo* button on the mouse'. When this is done

the replacement disappears and the original replace-button (or the enquiry in which it is embedded) reappears in its place. There is no need for maps or other paraphernalia, such as the history-trail mentioned below.

OTHER TYPES OF BUTTON

We emphasized at the start that our approach was only a step in tackling the 'getting lost' problem. *In situ* replacement is suitable for many applications and works well, but it is not suitable for everything. As we have explained, Guide also supports buttons that generate out-of-line material (which is displayed in a separate sub-window — an ephemeral one in the case of note-buttons in OWL's Guide), and buttons that cause a goto. As examples the underlined terms in Figure 3, syntax and write permission, are buttons; if one of these is selected the corresponding definition — an explanation of what the term means — comes up in a sub-window. Authors see a need for these facilities and make quite extensive use of them. Extra mechanisms are therefore still needed to help readers know where they are, and retrace their steps. OWL's Guide, for example, provides a 'history trail' icon that allows recent steps to be retraced; UNIX Guide provides a pop-up menu on an undo operation, and this allows the user to see the current hierarchy — each item on the menu is the name of a button used to move down the hierarchy — and to go back to a previous level. Figure 4 shows an example of one such

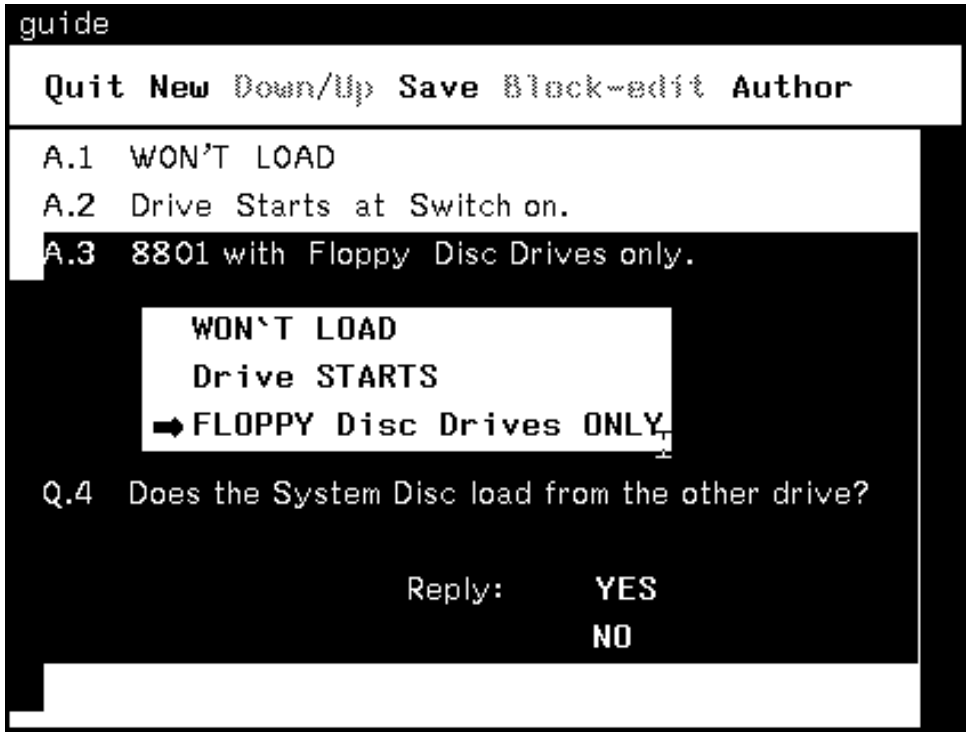

Figure 4. Pop-up menu giving current position in the hierarchy

pop-up menu. These mechanisms are, of course, something else for the user to learn, and although simpler than full-blown maps, are steps in that direction.

Furthermore the mechanism of usage-buttons, although, we trust, helpful to readers is of no special advantage to authors over a normal cross-reference mechanism. The author still needs to keep track of how the usage-buttons link with the corresponding definitions.

RESTRICTED VIEWS OF THE HYPERDOCUMENT

Finally, we will mention one other common facet of hypertext systems: the restricted view.

A popular way of simplifying a hyperdocument for certain users is to provide *paths*. A path constrains (or, in less authoritarian systems, advises) the user to make certain choices when travelling through the hyperdocument. This may be done either (a) by not revealing to the user what the other choices are, thus making parts of the hyperdocument totally invisible to him, or (b) by making it clear to the user what path is being taken, and also what other alternatives could be available if the user was not constrained.

In a system like Guide which supports replace-buttons, a facility similar to paths can be achieved by *pre-setting* buttons. A pre-set button is automatically replaced immediately before the user examines the material in which the button is embedded. UNIX Guide supports two kinds of pre-setting: one permanent, one undoable. Permanent pre-setting prevents the reader seeing parts of a hyperdocument, whereas undoable pre-setting sends the reader down a suggested path but still allows adventurous readers to explore other possibilities. Undoable pre-setting therefore helps the reader get a feel for a complete hyperdocument without the danger of getting lost in it. Pre-setting is implemented by attaching a level number to each button: only users whose profile puts them higher than this level will see the button. (One detail: usage-buttons can be pre-set, but to guard against endlessly recursive usage-buttons, Guide gives up gracefully at a certain maximum depth.) The level numbers are in fact a somewhat crude realization of link attributes and a more general attribute mechanism would be a considerable improvement.

An attraction of the use of pre-set buttons to implement paths is that it does not represent a new concept for the reader to learn; indeed the whole purpose of a permanently pre-set button is that the reader is totally unaware that some dictatorial author has removed a choice from his view of the hyperdocument.

CONCLUSIONS

One approach to aiding navigation in hypertext is to provide a map of the complicated structure that may underlie a hyperdocument. An alternative approach is to try to present material to the reader in such a way that he need not be aware of the underlying structures. Even a partial achievement of this goal, as is presented here, may be a valuable advance.

Finally, it is worth emphasizing that maps have other uses than showing the reader where he is at a particular moment. For example, a special map might be designed to give an impression of the overall scope and structure of a hyperdocument, and perhaps even to highlight the areas that the reader has *not* visited; it is an open question whether the ideal map for such an application resembles a traditional navigational map.

ACKNOWLEDGEMENTS

Two anonymous referees provided useful suggestions for improving this paper.

REFERENCES

1. F.P. Brooks Jr., 'Banquet talk: the newly burning bush', at *Hypertext 87*, Chapel Hill, North Carolina (1987).
2. J. Conklin, 'Hypertext: introduction and survey', *IEEE Computer*, **20** (9), 17–41 (1987).
3. B. Shneiderman, 'User interface design for the HyperTIES electronic encyclopedia', in *Hypertext 87*, Chapel Hill, North Carolina, pp. 189–194, (1987).
4. F.G. Halasz, 'Reflections on NoteCards: seven issues for the next generation of hypermedia systems', *Communications of the ACM*, **31** (7), 836–852 (1988).
5. T. Oren, *Personal communication*, 1988.
6. P.J. Brown, 'Interactive documentation', *Software—Practice and Experience*, **16** (3), 291–299 (1986).
7. R. Akscyn, E. Yoder, and D. McCracken, 'The data model is the heart of interface design', in *CHI'88 Proceedings*, Addison-Wesley, Reading, Mass., pp. 115–120, (1988).
8. A.J. Palay, F. Hansen, M. Kazar, M. Sherman, M. Wadlow, T. Neuendorffer, Z. Stern, M. Bader, T. Peters, 'The Andrew Toolkit — an overview', *Proceedings of EUUG Conference*, London, April 1988, pp. 311–314