Document Examiner: Delivery Interface for Hypertext Documents

Janet H. Walker

Symbolics Inc.
11 Cambridge Center
Cambridge, MA 02142

ABSTRACT

This paper describes the user interface strategy of Document Examiner, a delivery interface for commercial hypertext documents. Unlike many hypertext interfaces, Document Examiner does not adopt the directed graph as its fundamental user-visible navigation model. Instead it offers context evaluation and content-based searching capabilities that are based on consideration of the strategies that people use in interacting with paper documents.

INTRODUCTION

Hypertext documents are linked modules of information, created, distributed, and accessed electronically[1, 2]. This technology has tremendous potential for making more information more available to more people. The challenge at this time is to make the information actually accessible to people, not just potentially accessible. That is, the major challenge lies in defining the user interfaces for the software that will deliver hypertext documents.

Interfaces for most research-oriented hypertext systems have reflected the organizational structure of the underlying hypertext document. That is, the information base was organized as a network; the user interface was organized as a network browser[3, 4]. A solely network-based interface is not, however, a necessary characteristic of a hypertext document. It might not even be a desirable characteristic, once the time comes for hypertext to leave the shelter of academic research environments.

This paper describes the user interface strategy used in Document Examiner, an end-user interface for commercial hypertext documents[5]. Document Examiner is part of Symbolics Genera[6], the software development environment (operating system) for Symbolics computers.

The examples in this paper show Document Examiner being used to deliver the Symbolics software product documentation. In printed form, this documentation consists of around 8000 pages. It contains all forms of documentation, from initial tutorial material to reference material on software internals. This documentation was prepared by technical writers as part of the Symbolics software product.

In order to set the context, I would like to outline some of the factors for categorizing hypertext systems and contrast our system with two typical, widely known hypertext systems, Xanadu[7] and NoteCards[4]:

- What does it hold? We deliver documentation that is part of a larger software product. Xanadu was designed to manage a library of prepublished material, Notecards to organize a set of working notes.
- How is it organized? Our documentation is highly structured. The material in library-like systems has few inherent interrelationships. The material in Notecards is highly interrelated but typically not highly structured.
- Who writes it? Our document is prepared by a small group of cooperating writers. The documents in Xanadu were to be submitted by authors and the annotations by readers; in Notecards, each user serves as both writer and reader (although recent work[8] has explored using Notecards for collaborative work.
- How much does it change? Our document is under constant maintenance, with revised versions published with every software release. It has more changes than a library but is more static or controlled than the information in Notecards.
- How big is it? Our documentation corresponds to 8000 printed pages with about 10,000 nodes and 23,000 links. This is small by "global library" standards but large in comparison to personal notes.
- Where does it fit in? This work has commercial rather than academic roots and production rather than research goals.

Many current hypertext systems are research vehicles developed in academic environments where it is feasible to have individuals assuming both writer and reader roles interchangeably. In fact, some hypertext systems offer a common interface for reading and writing. In the product development world, however, it is more conventional and manageable to separate these roles: the writers are product developers and the readers are customers. We have separate interfaces for reading and writing our documentation. This paper addresses the readers' interface only, leaving description of the writers' issues to other papers[9,10,11].

DOCUMENT STRUCTURE

This section describes the documentation database for which Document Examiner is the interface.

The documentation is organized as a database of modules. The writers determine the nature of the modularity, depending on the information needs of the subject they are documenting. Modules can be any size at all and can contain any kind of subject matter. The decisions are made by the writer and are not subject to any kind of enforcement. Writers choose the module boundaries according to their understanding of how readers will need to access the material.

Modularity

In our implementation, we refer to the modules as *records*. Each record in the document database has a unique identifier, assigned at the time of its creation. These identifiers are used internally by the system to track the location of records. Readers and writers specify a

record by its name and type. Names are just that: any words sufficient to name the topic uniquely (within its type). Because our application is software documentation, the types are things like "function", "variable", or "section".

Internally, each record is composed of *fields*, which embody various kinds of information about the record:

- Content fields for the document (full description, one-line description and so on).
- Accessory information (keywords, the record type and so on).
- Audit information (the version number and the publication status of the record).
- Database information (the server location of the record, its outward links).

Figure 1 diagrams the kind of information found in a typical record. The content and accessory fields are maintained by the writers; the others are maintained by the editor that the writers use and by other supporting software.

Field name	*Field contents*
Name:	DOC:|CONVERSATION COMMANDS|
Version-number:	1
Disk-location:	(#P"Q:>rel-7>sys>doc>conv>conv1.sab.18" 6328 7780)
Source-file:	"SYS:DOC;CONV;CONV1.SAR.36"
Contents:	#<RECORD-FIELD CONTENTS>
Children:	((INCLUDE #<RECORD-GROUP DOC:|APPEND CONVERSATION COMMAND|> #<RECORD-GROUP DOC:|DELETE CONVERSATION COMMAND|> #<RECORD-GROUP DOC:|WRITE CONVERSATION COMMAND|>))
Tokens:	(("Converse" "commands"))
Keywords:	#<RECORD-FIELD KEYWORDS>
Oneliner:	#<RECORD-FIELD ONELINER>
Source-topic:	#<RECORD-FIELD SOURCE-TOPIC>
Source-type:	SUBSECTION
Flags:	Available, Modified, Filled, Installed
Modification-history:	((1 "jwalker" 2760810574))

Figure 1: A representation of a record data structure, showing some of its fields and their contents.

Relationships

One of the fundamental characteristics of a hypertext document is modularity. Another such characteristic is the ability to indicate the relationships between the modules. The hypertext literature often describes these relationships as "links"[1].

In our database, the writers use links between records to establish the overall structure of the documents in the database (see Figure 2). The links are directional, *from* one record *to* another. Links can link whole records or link a point in text to a whole record. In practice, in the current documentation, all of links are from a point to a whole record.

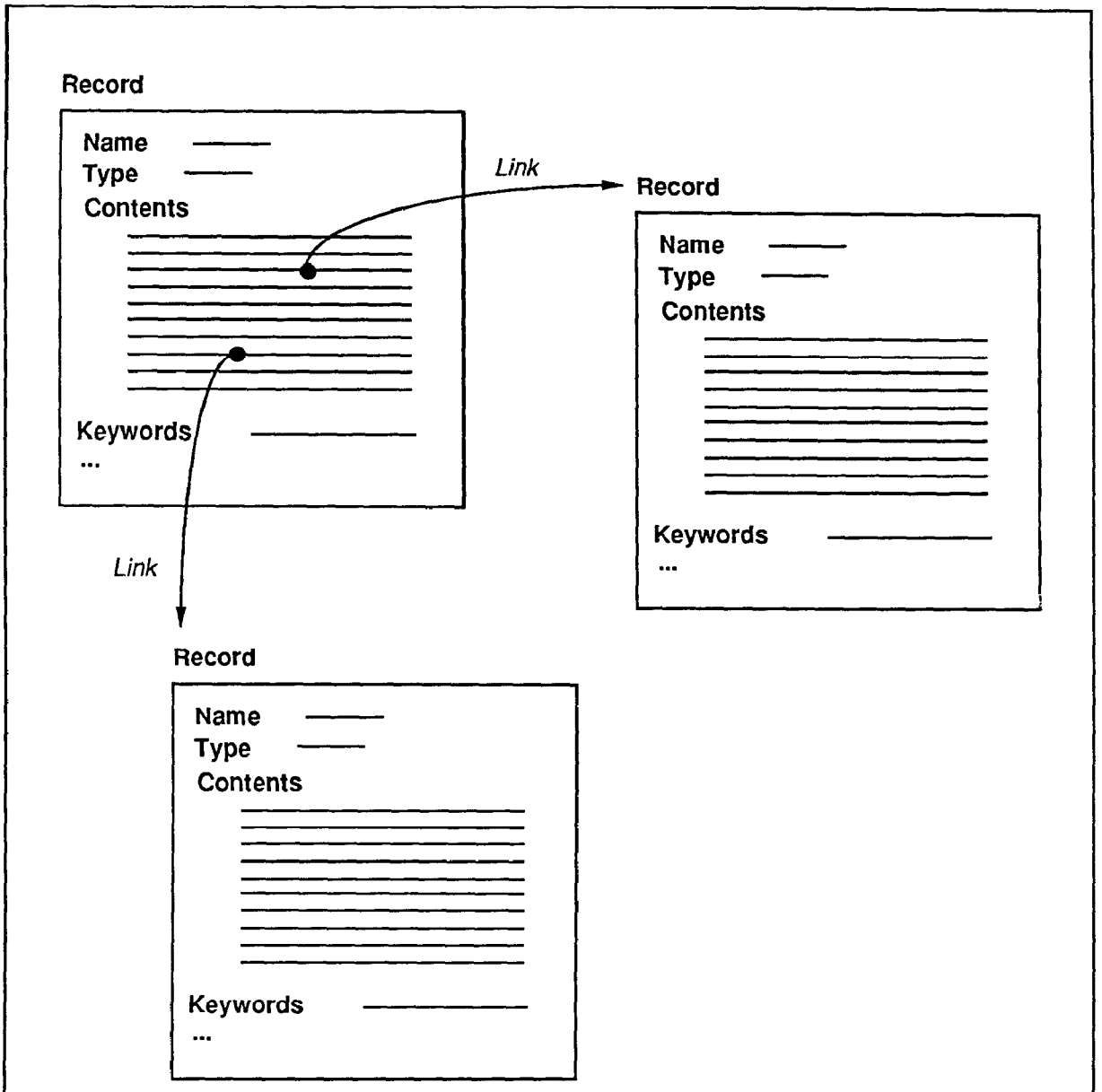

Figure 2: Links from a point in one record to another record.

In a documentation application, it is necessary to impose some structure on the information rather than providing simply a large "flat" namespace of interrelated modules. You can think of a conventional document, containing chapters, sections, subsections, and so on, as being a predefined path through a hypertext structure. The writers use links from within the textual content of a record in order to impose structure on the modules and hence create the document structure.

A taxonomy of different kinds of links has been proposed by Trigg[12]. At this time, we have only one kind of link and we support several different kinds of views for it:

- Inclusion. An inclusion link specifies that the content fields of the record referred to are to be included at that location when a reader is reading the document.

- Precis. A precis link specifies that the title and oneliner fields of the record are to be included at the location of the link.
- Crossref. The result of a crossreference link is to insert a conventional crossreference at the location of the link, for example, "See the section Combatting Gnats."
- Implicit. As writers create the material, they can enclose the names of some topics in implicit name links.

Conventional document structures are built using these different record views. One record can "call" any other record using a link. Figure 3 shows the structure imposed by inclusion and crossreference links on a set of records.

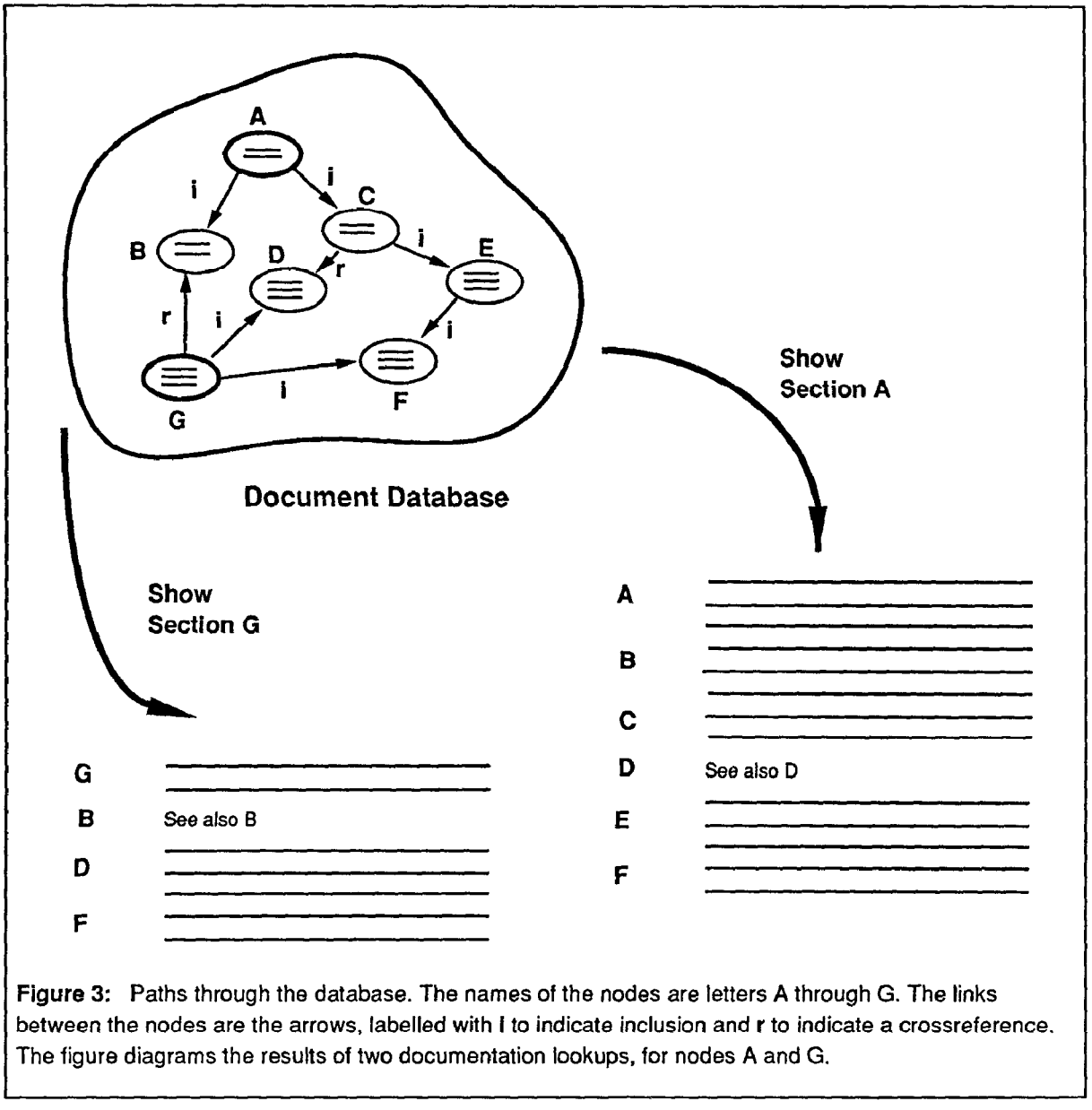

Figure 3: Paths through the database. The names of the nodes are letters A through G. The links between the nodes are the arrows, labelled with **i** to indicate inclusion and **r** to indicate a crossreference. The figure diagrams the results of two documentation lookups, for nodes A and G.

Versions

Some hypertext systems incorporate concepts that are usually addressed under the topic of version control or configuration control in software development systems[13]. This is primarily a document structure or management issue as opposed to an interface issue. For configuring the document, our document database uses the general system configuration tools[14] available with Symbolics Genera. In addition, writers can view either the published version of a record or the version(s) they are currently working on.

INTERFACE REQUIREMENTS FOR HYPERTEXT DOCUMENT DELIVERY

Hypertext documents, hypertext delivery software, and hypertext authoring software are all distinct, separable problems. In most cases so far, the people building the delivery interface are also the people creating the underlying information structure that it is delivering. Hence it is natural, but not necessary, for these three components to become intertwined in design. As the concepts in this field mature, this situation will change; standards for information structures will emerge, companies will emerge to prepare documents in hypertext form and other companies will develop delivery interfaces to serve different customer bases. (Apple's HyperCard product is a preview of the future in information delivery.)

In the near term, the problem facing designers of hypertext delivery interfaces is exactly that posed by Jeff Conklin in his description of using hypertext[1]:

> "The writer is no longer making all the decisions about the flows of the text. The reader can and must constantly decide which links to pursue....reading hypertext ... tends to present the user with a large number of choices about which links to follow and which to leave alone. These choices engender a certain overhead of metalevel decision making..."

This is a description of a very high level of cognitive overhead, much higher than that experienced by people reading conventional documents. I think we should view this description as the challenge of hypertext interface design rather than as its solution.

If hypertext documents are to replace paper documents, they must both retain the advantages of paper delivery and provide the advantages of electronic delivery.

What does a paper manual provide?

In spite of its often-derided rigidity and linearity of structure, paper has had many incidental good qualities as a delivery medium. In designing a replacement for paper, one needs to consider these qualities and to devise electronic analogs for them.

What do people do with paper manuals? How do they use them?

- Look up things they know they want. They use the index to locate the relevant pages. For something they refer to often or something very important that they want to be able to find again, they put in bookmarks.
- Try to find out what they want. They sometimes use the index or table of contents to find anything that might be related to what they want. They then travel in ever-

widening circles around that area of the book, hoping to stumble on the relevant material.

- Try to find out the general nature of what is available. They use the table of contents to see the overall structure or generally flip through pages looking at headings or pictures.
- Annotate the material. They use a highlighter to emphasize relevant portions. They make notes in margins with ideas, crossreferences, caveats, clarifications, or examples.
- Take "snapshots" for use elsewhere. They sometimes copy pages for either remote use or very fast reference use.

People also mention the reassuring tangible, physical nature of paper:

- Take it on the bus. Books are portable; you can read them anywhere.
- Leave it open beside the keyboard.
- Find vaguely remembered information by position.
- See at a glance "where" you are (by fractional position within the book). The size, feel, and design of a book all give information about its likely relevance to any particular information-finding problem.

In addition, paper is a "low overhead" medium for readers. If they want, they can simply read the material in the order that it was supplied by the author, with some degree of confidence that the result was designed to be comprehensible that way. Strategies for using paper documents are highly overlearned skills for most adult computer users.

Replacements for the good qualities of paper need to be more than imitations that try to carry the surface features of paper into the electronic world. Instead, they should be functional analogies that provide the same kinds of benefits with an entirely different implementation.

What can an electronic manual provide?

An online manual can provide benefits that are unimaginable with paper delivery.

- *Full indexing.* We can analyze the contents of electronic documents in order to provide much more complete indexing than is feasible for almost any paper document. In addition to indexing, brute-force full-text searching is also an option for locating material.
- *Quick following for cross-references.* When the text of a document instructs its reader to "See section 9.3", a document delivery interface can let the user follow that instruction directly.
- *Back referencing.* Software that can analyze the structure of a document knows which other topics have links to a particular topic.
- *History.* An online delivery interface can keep track of what the reader has already seen.

As online information bases become more extensive, helping users manage volume, context, and history will emerge as the most important practical problems with interfaces.

DOCUMENT EXAMINER INTERFACE DESIGN

Document Examiner was designed to preserve beneficial aspects of paper manuals while adding the power and flexibility of content-based operations.

Document Examiner is a window-based utility that is integrated with the rest of Symbolics software environment. Figure 4 shows a screen display from Document Examiner.

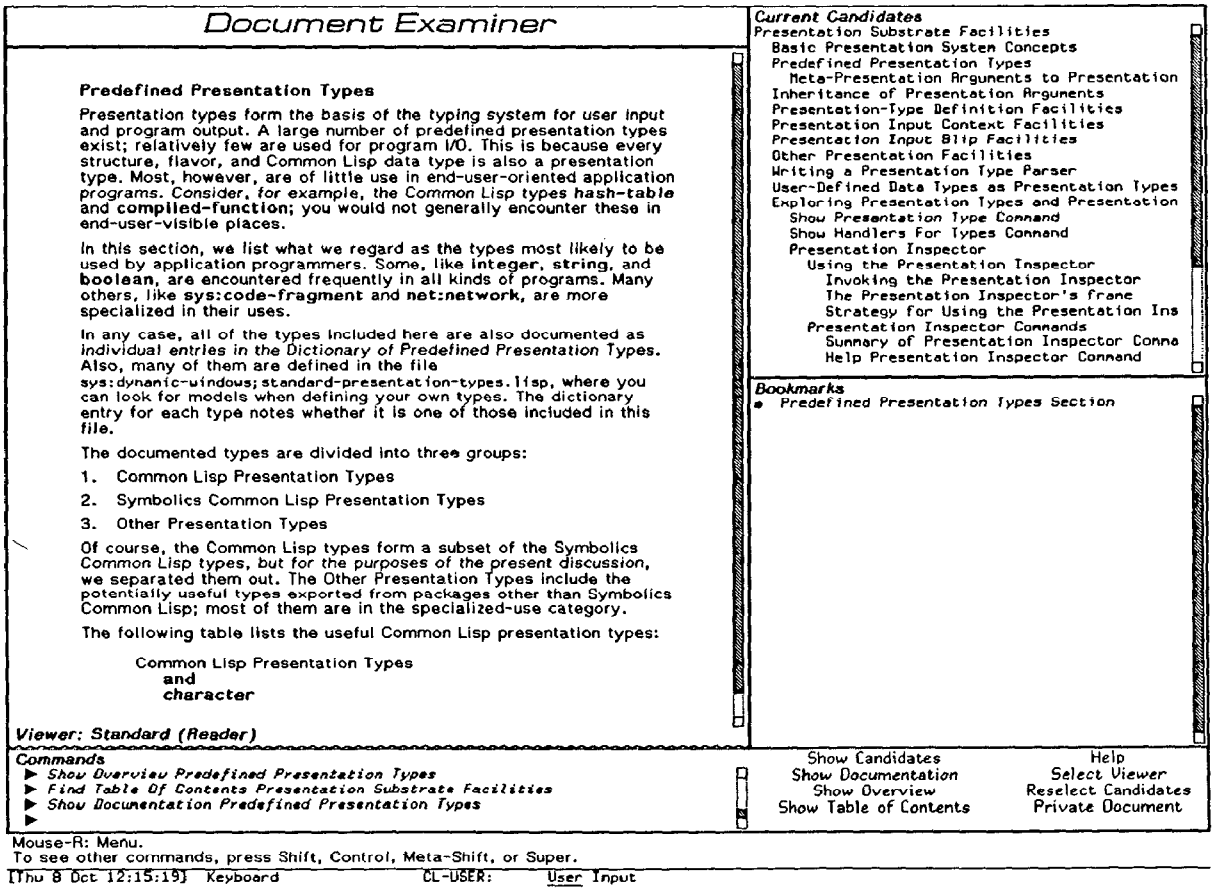

Figure 4: Document Examiner screen display. The viewer contains the first screenful of a section, whose bookmark is in the bookmarks pane. The candidates pane contains the table of contents for the document that this section appears in. Several recent commands are visible in the command pane.

The most fundamental decision in the interface was to make the material that a person was reading look essentially as it would in a paper book. The reason for doing this was "ease of use". We saw no reason to have the underlying information structure be reflected in the user interface model unless that structure was a good model for interacting with information. My experience in trying to help users with a tree-structured information interface (the INFO subsystem in EMACS) led me to believe that a book-like interface would be more palatable for many people.

The rest of this section describes the ways in which Document Examiner addresses the requirements of people using it to read documentation.

General description and terminology

Document Examiner is an application that runs in its own window. The window is divided into panes (subwindows) used for different aspects of managing the user's interaction with the document:

- *Viewer.* The majority of the screen area is used for showing a topic. It gives people the feeling of reading a section from a book.
- *Command pane.* The bottom area of the screen contains a fixed command menu and a command interactor area where the user can type commands. Most commands are available in either mouse or keyboard forms.
- *Candidates pane.* "Candidates" is the term used for a set of record names that have been retrieved in answer to some user query. Candidates are *mouse-sensitive.* (that is, clicking a mouse button while the mouse cursor is positioned over the name invokes a command.)
- *Bookmarks pane.* This area of the screen maintains a chronological record of the topics that a user has read in the accompanying viewer. The bookmarks are mouse-sensitive.
- *Overview window.* "Peephole" context for a topic. In a temporary display, the overview shows both a graph of the inclusion links for a topic and all its outward links.

Topic Lookup

The basic lookup command is Show Documentation, which operates on a record name. This is the command that users issue to see documentation for some system feature or document section for which they already know the name (or enough of it to specify the record uniquely). Figure 4 shows a record partially read into a viewer.

Show Documentation is actually a request for an inclusion view of the record. The system retrieves the record from the remote server and begins displaying the fields specified for an inclusion view. As further references are encountered, those records are retrieved and displayed according to the view that the writer specified for them. Structurally speaking, the users are reading the linear structure resulting from tree traversal of a subtree in the document structure.

Users can scroll forward through the topic to its end. Repositioning within a topic is handled with standard system scroll key commands and a mouse-operated scroll bar.

The display is analogous to an editor buffer in which each new record's display is appended to previous displays. The user can reselect previously displayed records (using the names in the bookmarks pane), scroll through earlier text, or use search commands to look for a particular textual string within the display.

As writers create the material, they enclose the names of topics in implicit name links. Figure 4 shows a record that contains this kind of link. Every topic name that is visible

anywhere in the documentation can be an implicit link to another topic. Users can follow these links because they are mouse-sensitive, either directly or as operands for typed-in commands.

Finding topics of interest

The basic search command is Show Candidates, which operates on a set of words. This command is Document Examiner's equivalent to using an index in a paper manual. This is a happy case where a strategy that people are accustomed to from the paper world (using an index) can be implemented far more powerfully online than in the paper world.

Show Candidates uses the word or phrase that the user specifies to search for records that contain those words in their titles or keywords. Figure 5 shows some results from an example query.

Command line:

Show Candidates finding help

Candidates offered:

Finding Out About Zmacs Commands with HELP
Finding Out What a Prefix Command Does
Method for Searching for Appropriate Zmacs Commands
More HELP Commands for Finding Out About Zmacs Commands

Keywords for "Method for Searching for Appropriate Zmacs Commands":

m-X Apropos
Searching for appropriate commands
Finding the right command
Help A Zmacs command

Figure 5: Documentation topics suggested by Show Candidates for the words "finding" and "help".

The user can control several attributes of the search strategy:

- *Kind of matching.* The default search strategy uses simple heuristic matching to identify records of interest. When the words in the query and the words in the keywords have stems[15] in common, then the record is retained for the candidates. For example:

  "Deleting files" *matches* delete-file
  File deletion
  Deleting multiple file versions

  Also available are other modes of searching that involve conventional exact or substring matching on the query words and keywords.

- *Multiple word order.* The default is to accept a record as a candidate if it has the query words in the keyword phrases in any order. Other modes specify that the words have to be adjacent, in the same order, or nearby (in the same keyword phrase) in order for the record to qualify.

- *Word combination.* The default searching uses "logical and" combination for a multiple word query. All of the words in the query have to be present in the keywords for the record to be a candidate.

Searching is based on keywords rather than the full text of the documentation for several reasons. Full-text searching is slower than keyword search by definition because the volume of material is much greater. In addition, the full text is kept on a server machine (for storage efficiency) rather than in the user's local memory; searching would be a performance bottleneck when several users needed to search at once. Furthermore, although we have not tried a fully inverted index, we expect that it would result in many more false alarms without more hits than keyword indexing does.

The candidates resulting from any query are stored in the candidates pane. Figure 6 shows the candidates list that results from a search for "deleting files". Any record in the candidates pane can be operated on with a number of mouse commands, including:

Show Documentation
Show Overview
Show Table of Contents

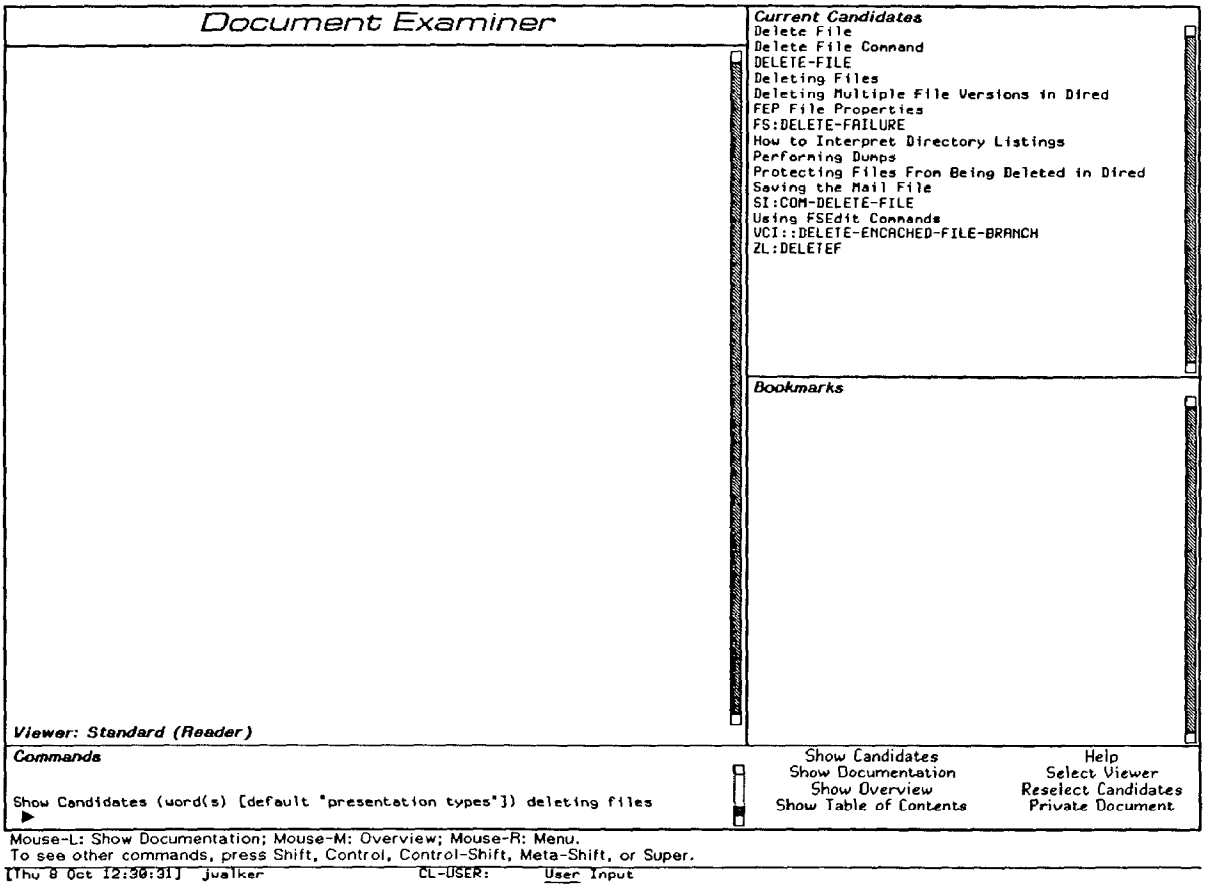

Figure 6: The candidates pane contains the list of candidates resulting from the search for "deleting files".

Examining context and structure

After using an index search command, the user next needs to determine which of the items retrieved are most relevant. Using an index in a paper manual, this can be a very time-consuming task, depending on the quality of the index and the number of references to check. In this arena, an online system can shine.

The Show Overview command displays an overview of any record (see Figure 7). An overview contains contextual information enabling the user to determine whether or not this record is relevant and, if it is relevant, whether it or something related to it is more appropriate.

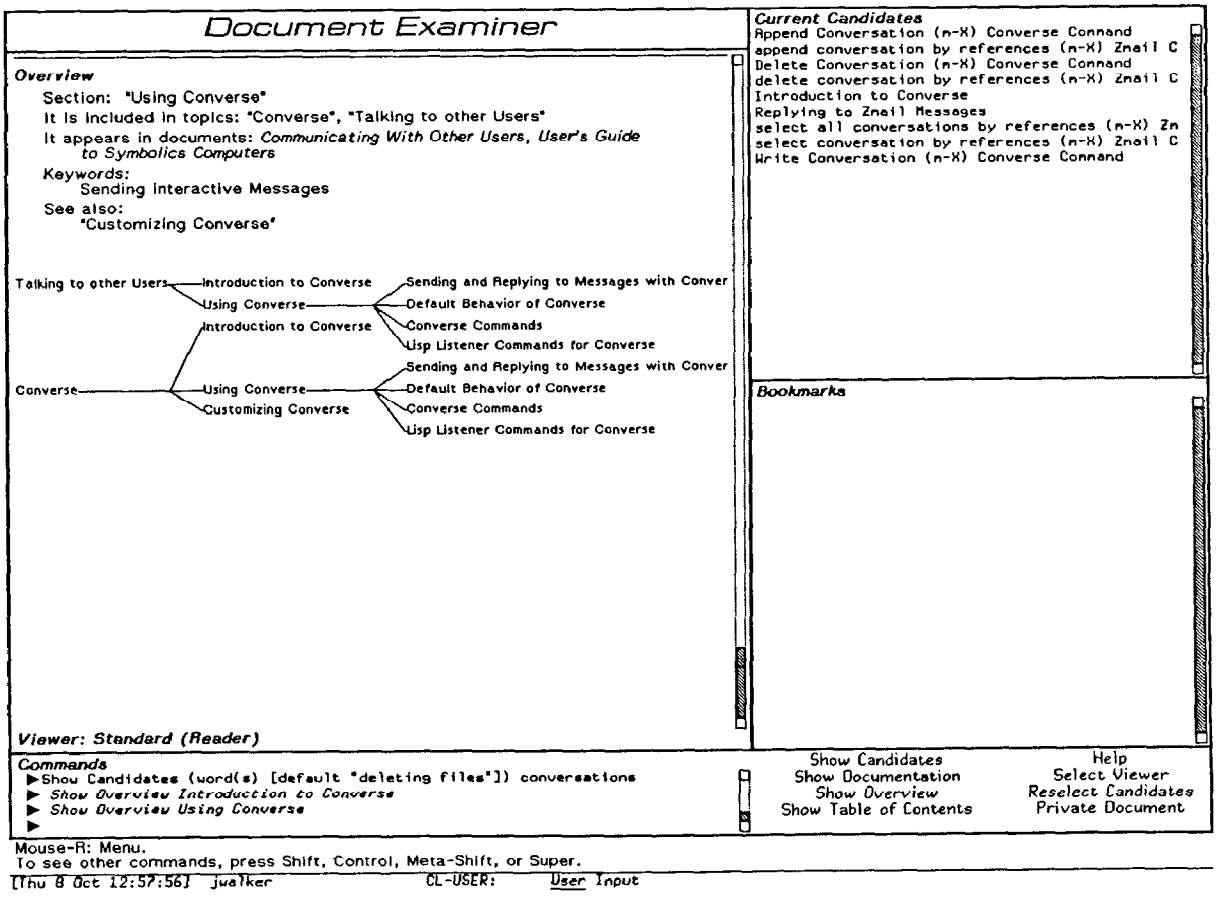

Figure 7: An overview for the section "Using Converse" appears in a temporary window overlaying the Viewer. This section occurs twice in the document set, in two contexts, as shown by the diagram.

One graphic display is shown for each inclusion-type link to the record. In tree structure terms, the graph shows the parent, siblings, and children for the overviewed record. All of the record names on the screen are mouse-sensitive so that the user can explore this set of topics further, perhaps with more overviews, in order to pinpoint the relevant areas of the document. In fact, users employ the overview heavily to explore "the neighborhood" for a record and thus to zero in quickly on the most relevant area to read. This graphic display is primarily a decision-making aid and only secondarily a navigation aid.

This kind of display has significant advantages over either a conventional table of contents or a full display of the graph. It constrains the amount of information that the user has to process while still giving enough relevant information with which to make decisions. (In this sense, it is similar to the powerful "fisheye view" concept[16].)

Users starting out to investigate a new system or new topic area need an equivalent to the paper-based strategy of flipping pages to see what's there. To address this need, Document Examiner can provide a table of contents for the subtree under any record. Figure 4 has a table of contents display in the candidates pane.

The initial screen display (Figure 8) has the names of all the documents in the document set so that the reader can use those as a basis for commands to see either an overview or their table of contents.

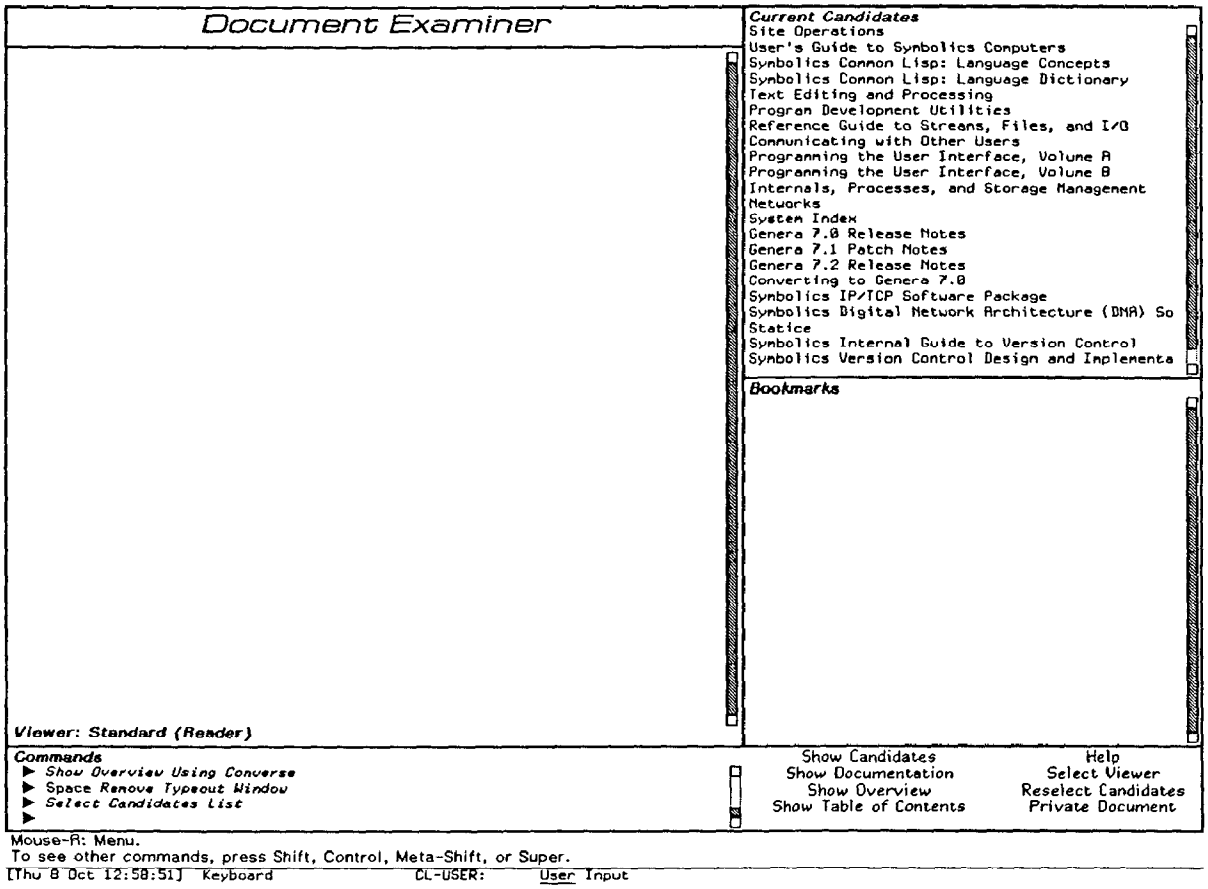

Figure 8: Initial screen configuration of Document Examiner, showing the "top level" directory of documents.

Saving the results of an investigation

Users shouldn't have to remember the history or state of their interaction with a document. Document Examiner addresses the issues of convenience and memory load with both short-term and long-term strategies.

For assistance with an ongoing investigative session, Document Examiner maintains several kinds of context:

- *Input history.* Using a standard system feature in Symbolics Genera, the user can recapture and edit any command entered earlier in a session.
- *Query result history.* In addition to reactivating earlier commands, a user can select the results of earlier commands. For example, the results of earlier queries can be reinstated in the candidates pane, saving some time but more importantly, eliminating the need for users to remember their exact queries.
- *Lookup history.* When a user asks to see a record, Document Examiner creates a bookmark for it. The set of bookmarks in the bookmarks pane constitutes a chronological record of a user's interactive session. The bookmarks are active, of course, so that a user can reselect a topic by clicking the mouse over the topic's bookmark.
- *Reading context.* Users follow crossreference links freely, suspending reading one topic in order to look at another. Document Examiner saves the user's reading position within a topic so that when they reselect that topic, it is positioned as they left it.
- *Preserving lookup history.* When substantial effort has gone into finding a set of relevant topics, it is useful to be able to save the results of this effort for a future session. The user can save a set of bookmarks in a file called a "private document". The set of topics represented by the bookmarks can then be read in automatically in a subsequent session. This is our approach to the need addressed by Bush's "associative trails" in Memex[17].

EVALUATION

Document Examiner attempted to provide users with familiar and functional strategies for finding and using information in a large document set. How well did it succeed?

- *Look up things you know you want.* Show Documentation displays exactly and only the topic that the user requests.
- *Try to find out what you want.* Show Candidates functions like a powerful index. Show Overview displays local context and acts as a decision-making aid for whether to read a topic.
- *Try to find out the general nature of what is available.* Show Overview and Show Table of Contents display local or complete structural information for a document.
- *Annotate the material.* We have not yet attempted to address this issue.
- *Take snapshots.* Several commands serve to hardcopy a record or collection of records. Save Private Document lets the user save a collection of bookmarks for future use. Users can copy areas of the viewer (for example, code fragments) to editor buffers for further manipulation.
- *Tangible aspects of paper.* Document Examiner has a full-screen window whose parts are stable and always visible. Although this by no means models the physical attributes of paper, subjectively it has some similar reassuring properties.

Document Examiner was first shipped as part of Symbolics software product in April of 1985. In our experience, it is both usable and used. In a year-long usage survey, we found users at all levels of experience used Document Examiner about equally often. Both groups of users looked up large "conceptual" topics as well as short reference ones; Show Overview and Show Candidates commands were used heavily for locating material to read[18].

New employees of Symbolics are introduced to Document Examiner as part of their early experience with the machine. We have software engineers who know little about the organization of the paper manuals as they do most of their reading using the online form of the manual. In fact, a recent survey of the engineering staff found about half of the 24 people who answered either did not have a paper document set or had not removed the shrink wrap from their books (five months after receiving them).

Several people expressed strong preference for online lookup over paper. One person mentioned using paper occasionally when they didn't understand something reading it online (but commented that "the documentation was as impenetrable on paper as it was [online].") A few people remained opposed to online information delivery in principle, independent of the interface. For these people, the subjective value of the tangibility of paper outweighs all current benefits of electronic delivery.

The major complaints concerning Document Examiner, from both customers and inhouse users concern performance. Many commands, including overviews, large tables of contents, long lists retrieved by index searching, and remote lookup of long topics, take more than 10 seconds to complete. This amount of delay is unacceptable to everybody, including the implementors. The fact that people do continue to use this facility heavily in spite of the delays is probably a testimony to the usefulness of the online features over paper.

ISSUES FOR FURTHER WORK

We have identified a number of areas in our implementation that need further investigation:

- *Locators.* As the documentation being delivered by this kind of interface becomes larger, the index searching capabilities become correspondingly more important. Some of the work now underway in information science in automatic indexing is relevant to hypertext document delivery (for example[19]).

- *Annotation.* As in other hypertext implementations, users do need the capability to make notes "on" our documentation. We have approached this problem cautiously, however, since the design issues include helping users maintain their notes across different releases of the system documentation.

- *Context.* Readers often have some need to constrain the set of topics under consideration in searching tasks. Several kinds of constraints:

    - *Structural.* Consider only one particular document (that is, the records in a particular subtree).

    - *Content.* Consider only records that have anything to do with some general topic area (for example, only records related to I/O).

  This issue has been addressed by Intermedia with the concept of webs[20].

- *Naming.* Our topics are designated externally by topic/type identifiers. This naming strategy requires that topic names be unique within their type. At present, "section" is the record type used for all conceptual material in documents. As a result, the writers often feel that their freedom to name things appropriately is hampered by the implementation.

CONCLUSION

Document Examiner meets its goals of delivering information from a large, complex document set to users. As an interface to information, it is flexible and powerful. By building the interface around the information-finding knowledge and strategies that people bring from their experience with paper documents, it is simple to operate.

ACKNOWLEDGMENTS

Many people have contributed to the design, implementation, and refinement of this system. Particular thanks are due to Richard L. Bryan for implementation prowess.

REFERENCES

1. Conklin, J., "Hypertext: An Introduction and Survey", *IEEE Computer*, Vol. 20, No. 9, September 1987, pp. 17-41.

2. Engelbart, D. E., "Authorship Provisions in AUGMENT", *Intellectual Leverage: The Driving Technologies*, IEEE Spring Compcon84, 1984, pp. 465-472.

3. Robertson, G., McCracken, D. & Newell, A., "The ZOG Approach to Man-Machine Communication", *International Journal of Man-Machine Studies*, Vol. 14, 1981, pp. 461-488.

4. Halasz, F. G., Moran, T. P., & Trigg, R. H., "NoteCards in a Nutshell", *Proc. CHI+GI '87 Human Factors in Computing Systems and Graphics Interface*, SIGCHI Bulletin, April 1987, pp. 45-52.

5. Walker, J. H., "Symbolics Document Examiner", SIGGRAPH Video Review, Vol. 19.

6. Walker, J. H., Moon, D. A., Weinreb, D. L., & McMahon, M., "Symbolics Genera Programming Environment", *IEEE Computer*, Vol. 20, 1987, In press

7. Nelson, T. H., "Literary machines", Published privately by the author, 1981.

8. Trigg, R. H., Suchman, L. A., & Halasz, F. G., "Supporting collaboration in NoteCards", *Proceedings of the Conference on Computer-Supported Cooperative Work*, 1986, pp. 153-162.

9. Walker, J. H., "Symbolics Sage: A Documentation Support System", *Intellectual Leverage: The Driving Technologies*, IEEE Spring Compcon84, 1984, pp. 478-483.

10. Walker, J. H., "Supporting Document Development with Concordia", *IEEE Computer*, In press.

11. Walker, J. H., & Bryan, R. L., "An editor for structured technical documents", Paper accepted for Protext IV conference.

12. Trigg, R. H. & Weiser, M., "TEXTNET: A Network-Based Approach to Text

Handling", *ACM Transactions on Office Information Systems*, Vol. 4, No. 1, 1986, pp. 1-23.

13. Delisle, N. & Schwartz, M., "Contexts--A partitioning concept for hypertext", *Proceedings of the Conference on Computer-Supported Cooperative Work*, 1986, pp. 147-152.

14. Symbolics Inc., *Volume 4. Program Development Utilities,* Release 7.0 ed., 11 Cambridge Center, Cambridge, MA 02142, 1986.

15. Salton, G., *The SMART retrieval system--Experiments in automatic document processing,* McGraw-Hill, New York, 1968.

16. Furnas, G. W., "Generalized Fisheye Views", *Proc. CHI '86 Human Factors in Computing Systems*, SIGCHI Bulletin, April 1986, pp. 16-23.

17. Bush, V., "As we may think", *Atlantic Monthly*, Vol. July, No. 176, 1945, pp. 101-108.

18. Young, E., & Walker, J. H., "A case study of using a manual online", Paper in preparation for CHI '88.

19. Fagan, J. L., "Automatic phrase indexing for document retrieval: An examination of syntactic and non-syntactic methods", *Proceedings of the Tenth Annual International ACMSIGIR Conference on Research & Development in Information Retrieval*, ACM SIGIR, 1987, pp. 91-101.

20. Yankelovich, N., Meyrowitz, N, & van Dam, A., "Reading and Writing the Electronic Book", *IEEE Computer*, Vol. 18, No. 10, October, 1985, pp. 15-30.