Ever discovered after weeks of working on a software problem that your colleague down the hall solved it months ago? What you probably needed was a system like this.

A Browsing Approach to Documentation

Yvan Leclerc, Steven W. Zucker, and Denis Leclerc, McGill University

> There are those who would argue that the OS/360 six-foot shelf of manuals represents verbal diarrhea, that the very voluminosity of manuals represents a new kind of incomprehensibility. And there is some truth in that.
>
> Frederick P. Brooks, Jr.[1]

As Brooks implies above, the sheer volume of information required for research facilities to function makes that self-same information both difficult to retrieve and difficult to comprehend. His response to this Catch-22 situation (in the case of the IBM 360) was to compile a carefully organized set of manuals through which specific information such as the details of a programming language or the parameters of a subroutine could be easily found.

While the time-honored technique of manually searching through extensive indexes does adequately allow for the retrieval of such information, it does not lend itself to the simple and leisurely discovery of that information. In other words, it is difficult to browse through the information in order to discover just what is or is not available. Computerized documentation systems such as the VAX/VMS HELP facility[2] offer more flexibility in their information retrieval capabilities than hard-copy manuals, but they are not particularly well-suited to browsing either. The purpose of this article is to describe one experiment in the design of a documentation system that provides mechanisms for both needs—retrieval and comprehension.

The decision to design and implement a documentation system with browsing capabilities was made when we noticed that many members of our laboratory* tended to "reinvent the wheel" by needlessly duplicating the efforts of others in the laboratory. Even though these previous efforts were documented in various ways, most people were unaware that a particular topic (e.g., fast Fourier transforms or pseudocoloring of images) had even been explored by another member of the laboratory. Even fewer knew precisely what that other member had done. Also, whenever it *was* known that a particular topic had been investigated by a particular person, the original investigator spent considerable time explaining exactly where the documentation for the topic was to be found and in furnishing general information about the topic.

We hoped that providing a documentation system thai allowed people to both discover and quickly retrieve information about the resources of the laboratory (resources such as software and hardware documentation, technical reports, etc.) would alleviate the above problems. (It is also our experience that these information dissemination problems are not unique to our laboratory. In fact, they are more often the rule than the exception.)

System design

As stated above, the primary goal of our *browsing system* was to provide a means for browsing through and/or quickly retrieving information about the laboratory's resources. Our needs, however, dictated that the implementation, and especially the maintenance, of the system require a minimum of manpower and on-line storage.

Accessibility. Since executable programs are an important part of the resources of the laboratory, we felt that, ideally, the documentation system should be accessible from within other programs and that other programs should be accessible from within the documentation system. For example, the former might be useful when editing a program (e.g., to find the parameters of an external subroutine) and the latter would allow a person to discover what programs are available without leaving the environment of the documentation system. Both capabilities are provided in our current implementation.

*The McGill University Computer Vision and Graphics Laboratory.

0018-9162/82/0600-0046$00.75 © 1982 IEEE

Information organization. A capability for browsing requires both an appropriate presentation of individual items of information and an organizational context to facilitate the browsing. The first need was met by naming each item of information explicitly so that it could be retrieved immediately once its name was located. In other words, each item of information, be it an executable program or some form of text, was assigned one or more keywords. The second need was met by making each entry short—short enough to allow a large number of them to fit on one page of a normal display terminal—and these keywords were then embedded in a structure whose category names were also keywords.

We chose a multiple-parent hierarchy (that is, a directed acyclic graph with a single root node) as the structural organization of the keywords. We did this because of the simplicity and flexibility of this type of organization. A strict hierarchy was considered too inflexible because it forces each keyword entry to be placed in a specific category. This often leads to difficulties both in searching and in categorizing information. At the same time, an arbitrary network of keywords, although quite flexible, is difficult for a person to grasp and follow. The compromise of a hierarchy with multiple parents allows items of information to be found through multiple paths, yet it is simple enough that traversal of the graph is natural.

Thus, for example, a general-purpose image-displaying program can be categorized under *both* IMAGE PROCESSING and GRAPHICS using a multiple-parent hierarchy, rather than under just one or the other as would be necessary with a strict hierarchy. As a result, the program can be found more easily, especially if the person searching for the program is not really sure where it belongs.

Separation of organization and information. The hierarchy of keywords described above is kept in a separate network file, with the information associated with each keyword pointed to by file names. The network file basically contains a doubly linked list of variable length records containing the keyword name, the associated file name, and pointers to the fathers and sons of the keyword. The simplicity of this structure is possible primarily because the text information is kept separate from the organization.

There are other important advantages in storing the information separately from the hierarchy:

(1) Once a keyword has been entered for a particular item of information, the information can be updated by the person responsible without the intervention of a "librarian."

(2) Information need not be modified in any way to be incorporated into the browsing system.

(3) The information can be created independently of the system without the need for special editors, thereby reducing the implementation cost.

(4) Since information is not stored explicitly, the structure is relatively small and can be quickly and efficiently traversed by the system, allowing quick traversal of the graph by the user.

Human interface. One constraint on our design of the human interface to the browsing system was the requirement that it be usable from a standard video terminal with minimal graphic capabilities. Inspired by the standard Lisp pretty-printed form of displaying lists, we chose the indented form of displaying a hierarchy illustrated in Figure 1. The desire for quick traversal of the graph led to single keystroke commands, which are listed at the bottom of the screen as a reminder to the user.

Figure 2 illustrates the initial display of the browsing system. The design of the human interface was also inspired, in part, by an application of the Zog system[3,4] to browsing through books and technical reports.[5] In fact, a version of the browsing system could have been implemented in Zog, but this conflicted with our requirements for simplicity of implementation and efficiency of computation. Interested readers can find further details of the issues of a document retrieval language in Gebhart and Stellmacher.[6] Note that only one level of the hierarchy is initially visible in Figure 2, although ellipses indicate that further levels exist. This is to simplify the user's initial view of the system, but the display can be changed to view up to four levels of the hierarchy simultaneously. If the displayed hierarchy does not fit

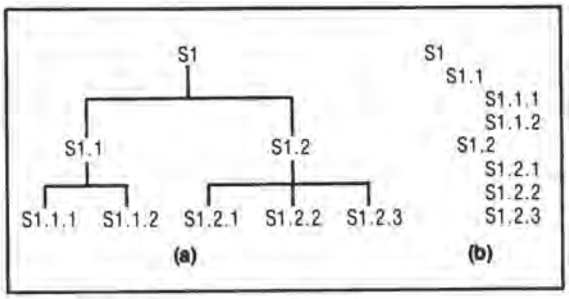
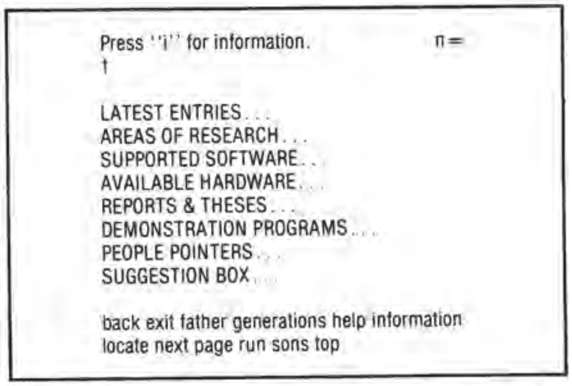
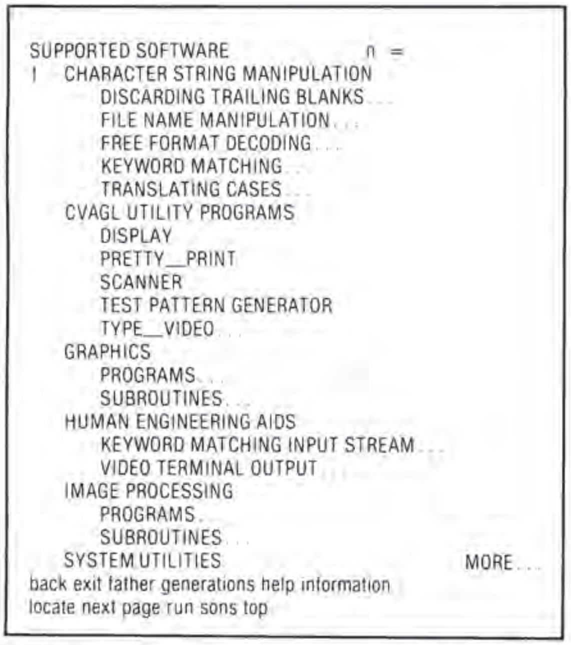


Figure 1. (a) A small hierarchy. (b) The same hierarchy displayed in the indented form used in the browsing system.



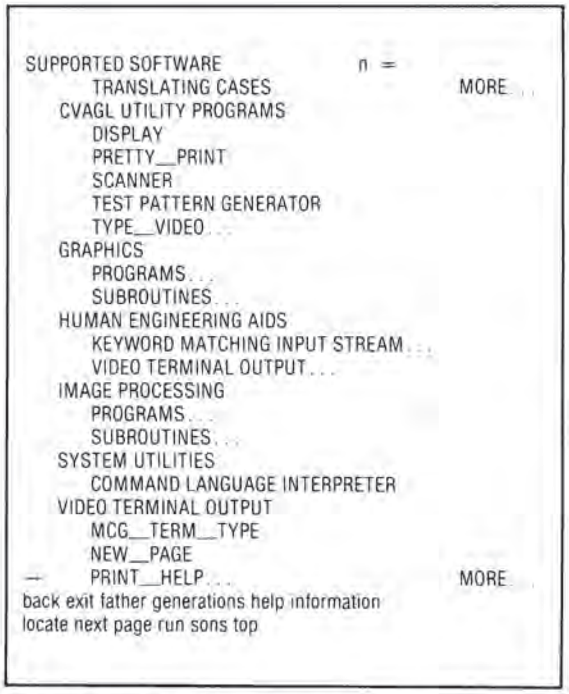
```
Press ''i'' for information.          n=
↑

LATEST ENTRIES . . .
AREAS OF RESEARCH . . .
SUPPORTED SOFTWARE . . .
AVAILABLE HARDWARE . . .
REPORTS & THESES . . .
DEMONSTRATION PROGRAMS . . . .
PEOPLE POINTERS . . .
SUGGESTION BOX . . .

back exit father generations help information
locate next page run sons top
```

Figure 2. The initial display of the browsing system. The ellipses following the keywords indicate that the keyword has sons. Attached to each of these keywords is an arbitrary text file, which can be retrieved by moving the pointer (currently pointing to the topmost keyword) to the appropriate keyword and then pressing "i." The set of available commands are listed at the bottom of the screen.



```
SUPPORTED SOFTWARE                          n =
|   CHARACTER STRING MANIPULATION
        DISCARDING TRAILING BLANKS . . .
        FILE NAME MANIPULATION . . .
        FREE FORMAT DECODING . . .
        KEYWORD MATCHING . . .
        TRANSLATING CASES . . .
    CVAGL UTILITY PROGRAMS
        DISPLAY
        PRETTY__PRINT
        SCANNER
        TEST PATTERN GENERATOR
        TYPE__VIDEO . . .
    GRAPHICS
        PROGRAMS . . .
        SUBROUTINES . . .
    HUMAN ENGINEERING AIDS
        KEYWORD MATCHING INPUT STREAM . . .
        VIDEO TERMINAL OUTPUT . . .
    IMAGE PROCESSING
        PROGRAMS . . .
        SUBROUTINES . . .
    SYSTEM UTILITIES                                    MORE . . .
back exit father generations help information
locate next page run sons top
```

Figure 3. Two levels of the keyword "SUPPORTED SOFTWARE." The "MORE . . ." indicates that the two levels could not fit completely on the screen. This display was obtained by using the "generations" command preceded by the number "2." (Optional numeric arguments, as for the "generations" command, appear after the "n =" of the top line of the display.)

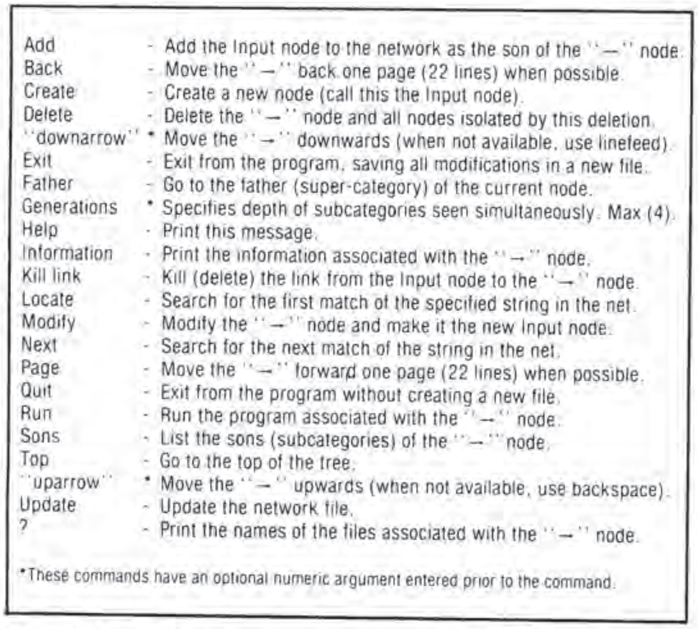


```
SUPPORTED SOFTWARE                          n =
        TRANSLATING CASES                               MORE . . .
    CVAGL UTILITY PROGRAMS
        DISPLAY
        PRETTY__PRINT
        SCANNER
        TEST PATTERN GENERATOR
        TYPE__VIDEO . . .
    GRAPHICS
        PROGRAMS . . .
        SUBROUTINES . . .
    HUMAN ENGINEERING AIDS
        KEYWORD MATCHING INPUT STREAM . . .
        VIDEO TERMINAL OUTPUT . . .
    IMAGE PROCESSING
        PROGRAMS . . .
        SUBROUTINES . . .
    SYSTEM UTILITIES
        COMMAND LANGUAGE INTERPRETER
    VIDEO TERMINAL OUTPUT
        MCG__TERM__TYPE
        NEW__PAGE
→       PRINT__HELP . . .                               MORE . . .
back exit father generations help information
locate next page run sons top
```

Figure 4. Two levels of the keyword "SUPPORTED SOFTWARE" scrolled down several lines. Notice that the arrow now points to the keyword "PRINT__HELP" and that the system is indicating that more of the hierarchy can be displayed by the two symbols "MORE . . ."

completely on the screen, the user can scroll the display up or down.

Figure 3 illustrates a case in which two levels of a hierarchy are displayed simultaneously, while Figure 4 shows the displayed hierarchy scrolled down several lines. (Figure 4 also illustrates the multiple-parent hierarchy concept; the keyword "VIDEO TERMINAL OUTPUT" is a son of both "SUPPORTED SOFTWARE" and "HUMAN ENGINEERING AIDS.")



```
Add            - Add the Input node to the network as the son of the ''→'' node.
Back           - Move the ''→'' back one page (22 lines) when possible.
Create         - Create a new node (call this the Input node).
Delete         - Delete the ''→'' node and all nodes isolated by this deletion.
''downarrow''  * Move the ''→'' downwards (when not available, use linefeed).
Exit           - Exit from the program, saving all modifications in a new file.
Father         - Go to the father (super-category) of the current node.
Generations    * Specifies depth of subcategories seen simultaneously. Max (4).
Help           - Print this message.
Information    - Print the information associated with the ''→'' node.
Kill link      - Kill (delete) the link from the Input node to the ''→'' node.
Locate         - Search for the first match of the specified string in the net.
Modify         - Modify the ''→'' node and make it the new Input node.
Next           - Search for the next match of the string in the net.
Page           - Move the ''→'' forward one page (22 lines) when possible.
Quit           - Exit from the program without creating a new file.
Run            - Run the program associated with the ''→'' node.
Sons           - List the sons (subcategories) of the ''→'' node.
Top            - Go to the top of the tree.
''uparrow''    * Move the ''→'' upwards (when not available, use backspace).
Update         - Update the network file.
?              - Print the names of the files associated with the ''→'' node.

*These commands have an optional numeric argument entered prior to the command.
```

Figure 5. A summary of the commands available in the browsing editor. Note that all of the commands of the browsing system are also commands of the browsing editor.

The user moves down the hierarchy by moving the pointer (in the top left-hand corner of Figure 2) up or down on the screen to the desired keyword, typing "s" to view its sons. The user can move up the hierarchy one level at a time (retracing his or her steps down the hierarchy), or can move directly back to the top. The user can also locate a specific keyword by entering either a full keyword or its abbreviation. The browsing system then searches for the first matching keyword from the top of the hierarchy (using a depth-first search). If the matched keyword is not the one the user wanted, he or she can force the system to go on to the next match.

Once the user has placed the pointer at an appropriate keyword, the associated information can be displayed on the screen or the associated program run. The user can always return to the browsing system via a control character, giving him or her the freedom to experiment.

Creating and modifying a network. Any user can create and modify a personal *browsing network,* although a password is required to modify the (default) system network. A different program is used for this, one that is a superset of the normal browsing system. In other words, every command in the browsing system is also available in the *browsing editor,* along with extra commands to create, insert, delete, and modify keyword entries.

There are two required steps for creating a new entry. First, a new node is created, specifying the keyword name and the associated text and/or executable file name(s).

Second, the new node is added to the network as the son of one or more other nodes. Once a node has been entered, its keyword name and/or file name can be modified and it can be moved about the network at will by removing and adding links from it to other nodes. Figure 5 lists the commands available for doing this in the browsing editor.

The approach we used in building the system network was to proceed from the abstract to the concrete. The top level of the hierarchy is the most abstract, representing various points of view that a user might have when entering the system. For example, a person interested in seeing the type of research carried on in the laboratory might start searching through the "AREAS OF RESEARCH" keyword; a visitor to the laboratory might first search through the "DEMONSTRATION PROGRAMS" keyword to get a glimpse of the current work; while a veteran user might be more inclined to start with the "LATEST ENTRIES" keyword to see what has recently been added to the network.

As a user moves down the hierarchy, the keywords become increasingly more specific, with the lowest-level keywords typically pointing to programs, subroutine sources, technical reports, etc. The multiple-parent feature of the system was used extensively to allow keywords to be found through a variety of paths, easing the discovery of information by the user.

Conclusions

The browsing system has been implemented as described in this article. To date, it has been used primarily, though not exclusively, to document software developed in our laboratory. This has been done in conjunction with a prior commitment to the use of a standard header page preceding every piece of software created in our labratory. Thus, for the most part, the browsing system is pointing to files starting with a standard header page, which makes for a consistent view of the available software.

The system has been very helpful in alleviating needless duplication of effort by providing a means for discovering, in a simple manner, what other people have done. And, perhaps more importantly, the development and maintenance costs were quite reasonable (two to three man-months for designing and implementing the system, with another man-month for organizing the information in the network), demonstrating the feasibility of implementing useful documentation systems at a reasonable cost. When these costs are compared with the time that our expert users no longer have to expend to simply disseminate information, the investment seems eminently worthwhile. In short, the system is working as planned.

Acknowledgments

The authors are grateful to John Mohammed, David Kashtan, Harold Hubschman, and Peter Sander for their help in designing the browsing system, and for their critical comments and helpful suggestions concerning this article.

This research was supported in part by NSERC grant number A4470 and in part by the Quebec Department of Education.

References

1. F. P. Brooks, Jr., *The Mythical Man-Month—Essays on Software Engineering*, Addison-Wesley, Reading, Mass., 1975, p. 134.
2. "VAX/VMS Command Language User's Guide," AA – D023B-TE, Digital Equipment Corporation, Maynard, Mass., 1980.
3. G. Robertson, A. Newell, and K. Ramakrishna, "ZOG: A Man-Machine Communication Philosophy," Carnegie-Mellon University Technical Report, Carnegie-Mellon University, Pittsburgh, Pa., Aug. 5, 1977.
4. G. Robertson, D. McCracken, and A. Newell, "The ZOG Approach to Man-Machine Communication," Carnegie-Mellon University Technical Report, Carnegie-Mellon University, Pittsburgh, Pa., Oct. 23, 1979.
5. M. S. Fox and A. J. Palay, "The BROWSE System, Part I: An Introduction," Computer Science Department, Carnegie-Mellon University Technical Report, Carnegie-Mellon University, Pittsburgh, Pa., 1979.
6. F. Gebhart and I. Stellmacher, "Design Criteria for Documentation Retrieval Languages," *J. Am. Soc. Information Science*, Vol. 29, No. 4, July 1978, pp. 191-199.

Yvan G. Leclerc is currently working toward a PhD in the field of computer vision at McGill University. His research interests include computer vision, the characterization of local vs. global computations, and cooperative processing. Other academic interests include the general study of human intelligence (from both the psychological and artificial intelligence points of view), and the human engineering of computer systems.

Leclerc received the M. Eng. and B. Eng. degrees in electrical engineering from McGill University in 1980 and 1977, respectively. He is currently a student member of the IEEE and ACM.

Steven W. Zucker is currently an associate professor in the Department of Electrical Engineering, McGill University, Montreal, P.Q., Canada and is codirector of the Computer Vision and Graphics Laboratory there. His research interests include computer vision, human perception, and artificial intelligence. He received the BS degree in electrical engineering from Carnegie-Mellon University in 1969, and the MS and PhD degrees in biomedical engineering from Drexel University, Philadelphia, in 1972 and 1975, respectively.

From 1974 to 1976 he was a research associate at the Picture Processing Laboratory, Computer Science Center, University of Maryland, College Park.

Zucker is a member of Sigma Xi, ACM, and the IEEE.

Denis Leclerc is currently in the master's degree program in computer science at McGill University, working in the programming languages area. He is also working part-time at Bell Northern Research on a text-to-speech conversion system. His research interests are programming languages, compiler optimization, and natural language understanding. He received the BSc degree in mathematics and computer science from McGill University in 1980.