BYTE

OCTOBER 1988 A McGRAW-HILL PUBLICATION

REVIEWS
- Dell's System 310
- Sprint
- Amstrad and Epson laptops
- Mac Scanners
- C_Talk
- Turbo Prolog 2.0

PRODUCT FOCUS

20 *Affordable* 80386s

Hypertext

What it is—and isn't
How it works
How you can use it

FIRST IMPRESSIONS

Presentation Manager and LAN Manager for OS/2

New Borland Turbos: Debugger, Pascal, C

PLUS
- Adobe Illustrator
- Apple's CD-ROM Drive
- PC power protection
- Super PC-Kwik/Spooler
- PolyBoost II
- Tickler/2
- Toshiba color printer

10
02662
0 440235 0

$3.50 U.S.A./$4.50 IN CANADA
0360-5280

IN DEPTH

Hypertext

Imagine, if you will, walking into the New York Public Library and picking up a book on Mozart. You begin to read and learn that Mozart was an Austrian composer in the late 1700s. You wonder what else was happening in Austria then, so you go to the card catalog, find a book on Austrian history, go to the stacks, locate the volume (if it's not checked out), and read it before you continue.

In this book, you find a reference to old Salzburg, and you wonder what it looked like. Back to the card catalog, and the stacks, to find a book with images from that time. Finally, you get back to Mozart and read of a piano concerto you've never heard. This time you head for the library's record collection and listening room.

This process continues until you have either satisfied your desire for knowledge on the subject or worn yourself out searching for it, whichever comes first.

Now imagine sitting at your computer and bringing up a hypertext system on music. You begin to read about Mozart. When you wonder about Austrian history, you simply highlight the text and request more information with a mouse click or a few keystrokes. To find images of old Salzburg, you use the same process. And to hear the piano concerto? The same.

Sounds a lot simpler, doesn't it? The only restriction to this seemingly endless fountain of knowledge is that the author of the hypertext system had to establish the connections for you to follow and provide the additional knowledge for you to retrieve.

In the article "A Grand Vision," Janet Fiderio delves into the mysteries of hypertext: where it came from (Ted Nelson's Xanadu and Douglas Engelbart's NLS), where it is now (Guide and HyperCard for microcomputers), and where it's going (CD-ROM). Janet describes its form and various functions, such as browsing, nodes, and links—aspects that separate hypertext systems from normal databases—as well as the two main directions of recent hypertext research.

One of these directions, using hypertext for great libraries of information, is the thrust of the article "From Text to Hypertext" by Mark Frisse. To organize large volumes of textual material, you must convert and structure quantities of (hopefully) on-line text into hypertext format. Mark deals with this process and its attendant problems.

The other research direction is using hypertext as an aid to problem resolution. In their article "The Right Tool for the Job," Michael L. Begeman and Jeff Conklin describe the gIBIS system's approach to system analysis. This system provides a framework within which to present issues, take positions on those issues, and argue with those positions—a framework for constructive discussion.

Finally, in "Hyper Activity," we provide a variety of resources, including some current hypertext products, various educational institutions involved in hypertext research, and a short, noninclusive reading list.

As the mass of knowledge we all must assimilate in this multifaceted world of ours continues to grow, from Mozart to microcomputers, the future of hypertext systems looks bright indeed.

—Jane Morrill Tazelaar
Senior Technical Editor, In Depth

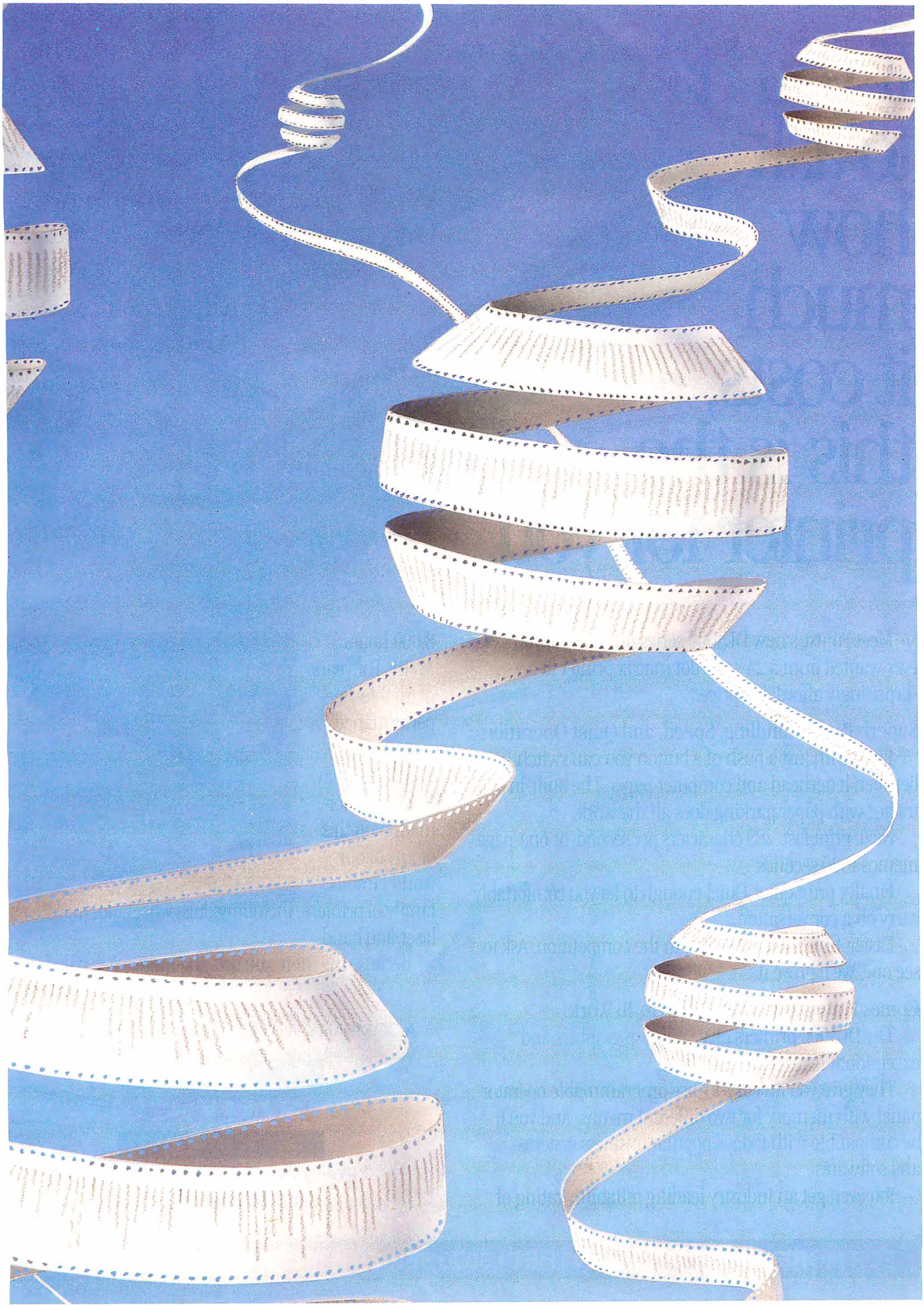


ILLUSTRATION: ROBERT TINNEY © 1988

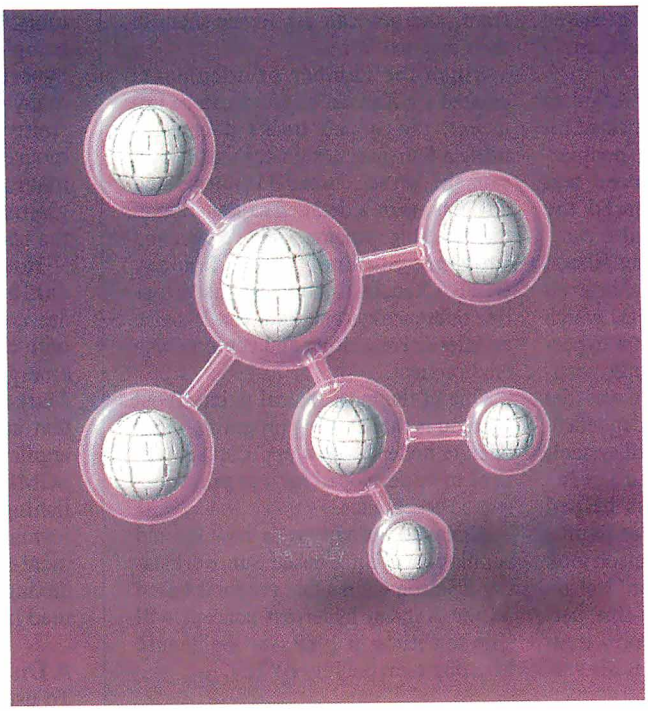
A Grand Vision

Hypertext mimics the brain's ability to access information quickly and intuitively by reference

Janet Fiderio

For 1945, the vision was a grand one: an on-line text and retrieval system that contained not only post-war scientific literature but also sketches, photographs, and personal notes. The machine, called a memex, would let you browse and make associative links between any two points in the library. You could then record and traverse them at will.

Vannevar Bush, President Roosevelt's science advisor and overseer of all wartime research, including the Manhattan Project, envisioned, yet never created, the mechanism. It became the foundation for all hypertext systems. (See the text box "The Pioneer Spirit" on page 238.) Now, 43 years later, hypertext applications are finding their way out of the research laboratories and into the market.



What Is Hypertext?

Hypertext, at its most basic level, is a DBMS that lets you connect screens of information using associative links. At its most sophisticated level, hypertext is a software environment for collaborative work, communication, and knowledge acquisition. Hypertext products mimic the brain's ability to store and retrieve information by referential links for quick and intuitive access.

Current hypertext programs don't use typical database record and file structures; their databases usually consist of screen-size workspaces called *nodes*. You can fill these computer index cards with text, graphics, images, and audio and video data. Most hypertext implementations link nodes in either a hierarchical or nonhierarchical fashion; some support both structures.

Early designers envisioned hypertext either as an environment for interconnected writing and literature storage or as a sophisticated, multipurpose research environment that encouraged cooperative thinking on shared projects. Product development now proceeds on several fronts. Universities, including Brown, Carnegie-Mellon, and the University of North Carolina at Chapel Hill, are experimenting with hypertext systems as multiuser teaching, library-reference, and writing environments. Commercial hypertext applications—like on-line reference manuals and documentation, public information systems, authoring systems, cooperative work systems, and personal organization tools—are either available or in development.

Hypertext programs, and the free-flowing databases that are their trademark, have been adapted for electronic publishing, project management, systems analysis, software development, and CAD. You can also find software

continued

ILLUSTRATION: ROBERT TINNEY © 1988

The Pioneer Spirit

Vannevar Bush designed a simple machine by today's standards. It used microfilm and photocells to store its data. But Bush, who was President Roosevelt's science advisor, dreamed up an information organization and retrieval scheme bold enough to influence two hypertext pioneers 20 years after the fact.

The first researcher influenced by Bush's concepts of associative links and browsing was Douglas Engelbart. His research at the Stanford Research Institute in the early 1960s centered around using computers to augment human intellect. At that time, he began developing the On-Line System, or NLS, now called Augment and used internally for several projects at McDonnell-Douglas.

Augment is an on-line work environment. In its original form, it served as a storage receptacle for memos, research notes, and documentation; as a communication network, since on-line conferencing was possible; and as a shared work space where researchers could plan and design projects.

Still running on a DEC 20, Augment stores information in a sophisticated hierarchical structure allowing nonhierarchical branching. Since speed was important, Engelbart invented the mouse as an input device. He also came up with the concept of viewing filters. Via filters, you can view a shortened version of the statement or file, which lets you move quickly through a hypertext database, scanning for only pertinent data. In fact, Engelbart was the first to use an F10 context-dependent Help system, an integrated mail system, multiple windows, and a shared screen.

While these developments helped researchers deal with complex multidimensional problems, Ted Nelson took Bush's concept a step further. Nelson envisions hypertext as an on-line network holding the world's literary treasures under one roof. Xanadu is his version of the publishing utility of the future. It is, perhaps, the most well-known hypertext system. It was Ted Nelson, in fact, who originally coined the word *hypertext* over 23 years ago to mean nonsequential writing.

As designed, Xanadu will be an ever-expanding publishing environment that millions of people could use to create, interact, and interconnect with linked electronic documents and other forms of hypermedia, such as movies, audio, and graphics. It's designed to run in parallel on many networks of servers. On a basic level, a storage manager lets you create links between like topics and then keeps track of the origins, variations, and interconnections of the text.

Xanadu documents consist of native bytes, the original document and inclusions, information originally found in other documents, and hidden pointers. Links are attached to bytes. You can ask the system to tell you where bytes came from, and you can ask to see them in their original form.

Since the number of documents created via Xanadu's system can be immense, the system tracks documents using a four-part designator that can locate the server, user, document, and contents. (For a detailed explanation of the tracking scheme, see "Managing Immense Storage" in the January BYTE.) Xanadu is more than just an on-line reference system, however. It's also an interactive writing and conferencing environment.

One of the most radical points about Xanadu is that existing programs won't operate under it. New applications will need to be developed for it to gain widespread acceptance.

A Xanadu prototype is now up and running on a Unix-based Sun workstation. Nelson claims that products based on the Xanadu hypertext concept will ship sometime in 1989 (see the item in Microbytes in the July BYTE).

(e.g., outline processors, teleconferencing systems, and windowing products) that borrow some, but not all, hypertext techniques.

In Many Flavors

Hypertext systems come in many flavors and support varying tasks. Typical hypertext software consists of a text editor, graphics editor, database, and browsing tool for three-dimensional viewing. (The *browser* is usually a graphic that you use to become oriented within a database filled with many nodes.) Bit-mapped displays, a mouse, windows, icons, and pull-down menus are all standard hypertext tools.

The various systems have one underlying database, and so far there's no DBMS standard. Current products use everything from home-grown to relational databases. Some products let you distribute the database across a number of networked file servers to create a collaborative hypertext environment.

When you use a hypertext application that you didn't help to author, you really see only the front-end of the program—the user interface. The machinations of the back-end, the database, are hidden. Depending on the application, some systems feature highly developed front-ends, like those in CAI systems, or complex back-ends, like those in research and cooperative work environments.

As a system user, you have access to a number of indexing capabilities. You can create inverted files of words, phrases, or keywords in context and perform word or Boolean searches. Some programs let you create hierarchical indexes, like tables of contents, while others let you create content-based indexes, like thesauri. Some systems let you create both.

If you write applications or use a system that doesn't delineate between author and user, you have access to hypertext's editing, linking, and development tools. You can author both simple and complex applications, depending on the hypertext system you use. In addition, many products let you invoke programs from your application at the touch of a mouse. These programs can be short and macro-like or large conventional programs that you would normally run from the operating system.

Not surprisingly, the only thing standard about hypertext systems is that there are no standards. It's a new technology with creative new implementations. One emerging standard, the Standard General Markup Language (SGML), lets hypertext authors create links across various applications. Although you usually hear SGML described as an electronic-publishing standard indicating type sizes and formats, it also features useful document-structure cross-referencing and indexing commands. Most text editors can read links created with SGML.

A Discrete Affair

To use a hypertext system, you must get used to parsing your information into small discrete units, or nodes, which consist of a single concept or idea. In theory, nodes are both semantically and syntactically discrete. The information contained in a node can usually be displayed on one computer screen. In situations where you need more space, some programs let you create longer nodes that scroll up from the bottom of the screen.

Nodes can come in two varieties:

typed and untyped. An untyped node is a box for information. It has no label or descriptor, so you can fill it with anything. A typed node is labeled, and the descriptor helps you determine the style of information contained in the node. Types help you to classify nodes or define specialized operations. They are also helpful when you're browsing through a database looking for a particular area of interest.

One system that uses typed nodes is gIBIS, the Graphical Issue-Based Information System from MCC (Microelectronics and Computer Technology Corp.). It's a prototype designed for systems analysis of complex problems. It lets you create three basic types of nodes: *issue* nodes, describing an issue you wish to discuss with your work group; *position* nodes, describing an assertion that resolves an issue; and *argument* nodes, containing your objection or support for a position node. Organizing nodes in this manner helps gIBIS users navigate easily through a complex hypertext network. (For more details, see the article "The Right Tool for the Job" on page 255.)

You can also combine nodes to form *composite* nodes. These are composed of related subnodes that can be handled as a single object or broken out into individual elements. You can create icons to reflect the contents of a composite node for easy access. You can also rearrange subnodes if needed.

Depending on the hypertext product you use, nodes can be displayed on the screen one at a time, as in Apple's HyperCard, or in groups, as in NoteCards from Xerox Palo Alto Research Center (PARC), a system designed for idea processing (see figure 1).

The Missing Link

In general, links are used to connect the nodes. A hypertext link is like an electronic footnote, an endnote, or a parenthetical phrase. That is, just as footnotes and parenthetical phrases direct readers of printed material to related points or further topics for research, hypertext links connect you to associated text or ancillary information.

Links, therefore, are the mode of transportation in a hypertext network. You follow them to move about between various nodes. You can usually embed them in text and then edit and review them to ensure that they are valid. You can also create, delete, or change link attributes.

Links must have two qualities: Your computer must be able to trace or follow them, and they must be able to transport you quickly from one node to another. Usually one or two keystrokes or the tap of a mouse button is all you need to transport you from one node to the next. The total time required to traverse a link is small, usually only a second or less.

While it's normally up to you to create links between nodes, some products can create links automatically; this ability may be useful for systems that need to cross-reference large text databases. Systems such as NoteCards also let you "type" links. A typed link specifies a particular relationship between two nodes, one that you define.

Links can do more than just connect two nodes, however. Depending on the hypertext system, links can connect annotations to a document (including notes and comments, like electronic Post-its) and provide organizational information, such as where the text fits in a table of contents or where it originated. Therefore, links can help define the node's relationship to other nodes within the database. Links may also clarify the contents of charts and graphics by connecting the graphics to explanatory information like longer descriptions.

Links usually originate at a single point, like a sentence, called a link *reference*. Their destination, called a link *referent*, is usually a node, a chunk or region of text.

Points and Buttons

A *point* is a single character, token, or icon that "points out" a link in a document. It's usually identified by either the name of the destination node, the link, or an arbitrary string, and by whether it's a source or destination point.

HyperCard and Guide (from OWL International) refer to points as *buttons*. Buttons can trigger the display of additional information, traverse a link, or activate a program. They can be represented by text or icons, or, as in HyperCard's case, they can even be invisible. (For more information on these two popular microcomputer hypertext systems, see the text box "What about Micros?" on page 242.)

A Bird's-Eye View

Hypertext systems are designed to let you browse through or quickly peruse associated nodes. While this feature is important, it can also be a problem, because in large hypertext databases, you can forget how or why you got to where you are. To alleviate this problem of disorientation,

continued

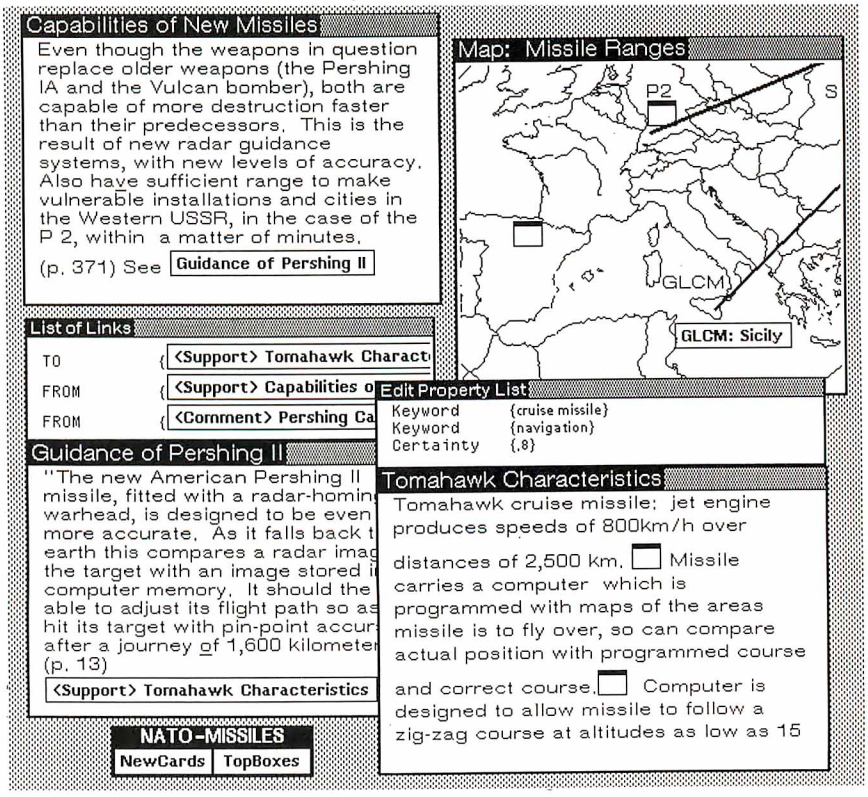


Figure 1: *NoteCards' display of multiple nodes. Note that the lower two nodes further clarify the topic introduced in the upper left node. (Graphics courtesy of Frank Halasz from MCC.)*

many systems provide a tool called a *graphical browser*.

The graphical browser is a node that contains a structural diagram of a network of nodes. Browsers usually supply a global or "zoom lens" map of the network. You can use the browser to orient yourself or to move directly to an area you're interested in by selecting that point on the screen with a mouse (see figure 2). While not all systems provide a graphical browser, most attempt to provide some type of overview system that helps you stay oriented in the network and visualize how information is linked. In large hypertext environments consisting of hundreds of nodes, browsing tools are especially important because it's so easy to get lost.

A browser can also help you decide on your next action. For example, Symbolics' Document Examiner, an on-line hypertext documentation system for Symbolics' Lisp machines, uses its browser, the Show Overview command, to help you quickly locate information. The command displays a tree-structured view of related nodes called *records*. By repeatedly using this command, you can get a feel for the context of the surrounding subject area and familiarize yourself with an area of interest.

A Variety of Tools

Depending on the particular product you use, commands and features may vary. Some hypertext products use a *path* to help you find your way through a network. Paths are default routes through a database; they guide or direct you through an ordered list of nodes.

When you follow a path, you are really letting the original author guide you to the next logical node, which relieves you of navigational duties. An example of a system that used paths is Textnet, created by Xerox PARC researcher Randall Trigg. It was designed as a multiuser literary-exchange system for the scientific community.

A viewing filter is another interesting hypertext tool. Basically, a filter does exactly what you'd think it would—it suppresses detail. By filtering the lower level of a node's contents, you can scan quickly through a network for the information you need.

Where the Products Are

The availability of windowing products and low-cost workstations with high-resolution graphics and storage options like CD-ROM have made the development of hypertext products more attractive. Fifteen or more systems are now used in universities and in research centers such as MCC and Xerox PARC, and commercial products are in development.

Hypertext systems vary significantly, depending on the applications and users they address. They are designed for either single-user or multiuser applications and are most commonly run on workstations, although more and more microcomputer applications are becoming available. HyperCard and Guide are perhaps the best known of the microcomputer products.

Typically, you'll find four types of hypertext systems: problem-resolution systems, on-line browsing systems, library or literary-exchange systems, and multipurpose systems. Depending on the type of system, the tools available may differ.

Systems designed primarily for problem resolution and network creation feature tools that help you define and analyze data through structured types of links and nodes. These systems help you organize elements in unstructured problems and feature commands that let you create and modify internal links between concepts quickly.

Most importantly, the tools can usually suppress details through viewing filters similar to those in Douglas Engelbart's On-Line System (NLS). (NLS is now used as a prototype for several collaborative-work projects at McDonnell-Douglas under the name Augment.) Such products might be used for systems analysis, idea processing, or authoring new applications. Augment and gIBIS are examples of systems designed to be problem-resolution work environments.

Just Browsing

Hypertext systems created primarily for browsing, such as CAI programs or on-line reference manuals, have fewer user tools for editing or link creation. These systems feature clear, understandable screen displays for presenting information and easy-to-operate browsing commands for perusing it. For example, the Document Examiner features a clean, book-like user interface and heuristic on-line string and keyword searches. You use these features to browse through the documentation, sometimes viewing information in several levels of detail.

Like many other browsing systems, the Document Examiner won't let you modify a reference manual, but you can keep a chronological record of recent searches or information of interest using the Bookmarks Pane. You can save personalized bookmarks and use a mouse to call bookmarks for fast retrieval. And

continued

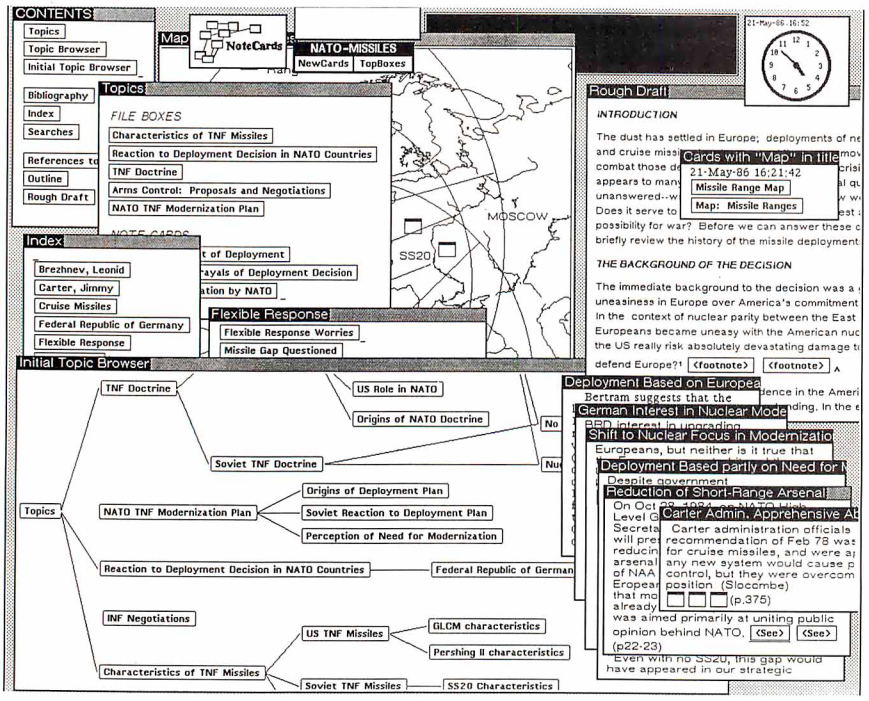


Figure 2: *NoteCards' displays can hold a significant amount of information. In this screen shot, a graphical browser (lower left) takes up much of the display space; a contents node, a topics node, and an index node are also displayed; and pages of relevant information are shown at the lower right. (Graphics courtesy of Frank Halasz from MCC.)*

What about Micros?

Want to test the hypertext waters and see what all the hoopla is about? Well, if you don't have a workstation handy, two of your primary options for microcomputer hypertext environments are OWL International's Guide and Apple's HyperCard.

Guide

Guide, which runs on both the Macintosh systems and the IBM PC AT and compatibles running Microsoft Windows, is a general-purpose hypertext tool (see "Guide" by William Hershey in the October 1987 BYTE). A few of the applications you can develop with it include on-line documentation, storyboards, E-mail, and CAI courseware.

Guide lets you create dynamic layered documents. The "guideline" network is organized in both a hierarchical and a nonhierarchical manner. To move about hierarchically, you use replacement buttons, which follow embedded menus. You can also use note buttons to bring up complementary information, such as a definition of a word or phrase. An inquiry button, which reveals the other buttons at your disposal, is also available.

To follow nonhierarchical links, you use the reference button, which will jump you to a new document or a different section of the document you're in.

Guide 2.0 uses an internal script language to let you execute external programs from your Guide document. You can also access and control videodisk players and modems via the serial port.

Last, but not least, a version called CD-Guide lets you create CD-ROM applications. OWL also markets a developer's toolkit so software developers can use Guide as a frame for an on-line help system.

HyperCard

HyperCard, available for the Mac II, the Mac Plus, and the Mac SE, is a personal organization tool and a simple database manager (see "HyperCard" by Gregg Williams in the December 1987 BYTE). It is also a commercial software developer's tool and is in use in some corporations as a front-end to the mainframe database.

This system uses screen-size cards (or window-size cards on the Mac II) organized into topic-related stacks to create simple databases. One card is displayed at a time. Touching your mouse cursor to a button on a card executes a script written in HyperTalk, HyperCard's programming language.

You can browse through already-created stacks (stackware), paint and type, author new cards and stacks, and write and edit HyperTalk scripts. (It's fairly easy to write scripts with HyperTalk because of its English-like syntax.)

HyperCard applications have been developed in many areas. Much stackware is available in the public domain.

Editor's note: *An assortment of public domain stacks can be found on BIX in the "stackware" area of the listings conference. See page 3 for more details.*

you can display a piece of documentation in an editor window while you write program code in another part of the screen.

The Document Examiner is particularly interesting because it's integrated into the software-development environment that it supports. It lets you scroll through the entire Symbolics software-development product documentation. The hypertext implementation of this documentation has an estimated 11,000 nodes and 23,000 links.

Another system designed for structured browsing by a large user base is ZOG, developed in 1972 at Carnegie-Mellon University and installed in 1982 as a computer-assisted information-management system on the USS *Carl Vinson*, a nuclear-powered aircraft carrier. Its applications included an on-line policy manual and an on-line maintenance manual with a videodisk attachment.

The commercial version of ZOG is KMS (Knowledge Management System) from Knowledge Systems. ZOG/KMS uses *frames* instead of nodes; frames are connected by two kinds of links, hierarchical and cross-referential. To help you navigate through a network, frames are formatted with a name, a title, a body, tree items linked to lower-level links, and special and command items. For simplicity and speed, ZOG and KMS use neither overlapping windows nor a graphical browser. The developers focused, instead, on fast text-search capabilities, multiuser support, and a minimal system-response time.

CD-ROM is particularly well suited as a database for hypertext browsing systems. One such system, being beta tested by Boeing for Knowledgeset Corp., is an on-line maintenance manual for the Boeing 757.

On-Line Libraries

Systems envisioned to support mammoth on-line libraries, documents, and document creation and critiquing, such as Ted Nelson's Xanadu, are the third major application group of current hypertext systems. These systems will probably feature complex, multiple structured back-ends, or databases, that can store everything from collaborative notes and research to E-mail, documents, and whole libraries.

Unfortunately, these systems will be difficult to implement because of the complexity and size of the task. Before such systems can become a reality, we must develop a standard user interface and a central storage system. In addition, we must be able to maintain the network such that all links are legitimate, copyrights and royalty issues are addressed, and the systems are fast enough to meet the needs of the general public.

Significant research in this area has been completed. Ted Nelson and his colleagues have worked on Xanadu for years. Randall Trigg's system, Textnet, let users store archival documents, making time-consuming research unnecessary. The system allowed collaborative writing and the critiquing of new documents. It featured two types of nodes—those containing text and those containing tables of contents of other nodes. The system defines over 80 types of links.

While many of the designs for the original on-line library systems had to be implemented from scratch, researchers at Bellcore are working on a hypertext-like front-end to connect existing on-line encyclopedias, libraries, and wire services. When completed, Telesophy, as the design is called, will let you access CD-ROM, recorded speech, and image archives. (Putting existing text onto a hypertext system is a challenge in itself. See the article "From Text to Hypertext" on page 247.)

Several well-known hypertext products function as general-purpose systems. You can customize these products to fit your application or simply to experiment with hypertext itself. NoteCards, HyperCard, and Guide are three such systems.

In the Driver's Seat

Regardless of the application, to use a hypertext system correctly, you must real-

continued

Some hypertext systems give you control when you may need guidance.

ize that you are in the driver's seat. Hypertext products won't think for you. They have no artificial intelligence. They might help you clarify and manage your thoughts or speed you through your research, but you are in control. Your value judgments determine what to include in the database, what type of links to create, and how to organize topics.

If you use an on-line hypertext documentation system, you decide which nodes to access and which links to follow. If you follow obscure paths, you may find it hard to locate information. Likewise, if your associative powers are weak and you create meaningless links, you may well end up with a worthless database. To put it simply, branching documents, like hypertext, require greater attention from both the system's users and its authors.

The Problem with Hypertext

Hypertext is an immature technology with many problems yet to resolve. Perhaps the most difficult part of creating a hypertext system is not building the user interface but creating sound underlying data models that can be maintained.

Since hypertext systems need to be maintained, systems designers should watch for uncontrolled linkages, which will become maintenance problems. Just as large software programs with many patches can turn into "spaghetti" code, so a hypertext system can turn into a morass of meaningless, obscure connections and references. Hypertext systems, therefore, must let you edit and delete links and nodes easily.

Another problem for some users is that some hypertext systems give you control when, in fact, you may need guidance. You may, for example, get lost following obscure links before you have a firm grip on the basics of the subject area you're trying to research.

When you're reading printed text, a good author will guide you through a network of interrelated, relevant points. With hypertext, you guide yourself and make your own associations—at the risk of taking the wrong turn and getting lost. Even experienced hypertext users can get lost in large hypertext networks. While graphical browsers may help, the lack of visual and spacial cues can still be disorienting. One of the valuable attributes of printed copy is that it has such cues.

Another issue is the difficulty of breaking a thought or a segment of information into a node. Themes in a document or thought can be very tightly interwoven, so much so that breaking the information into discrete nodes would be detrimental. Therefore, not all literature is suited for a hypertext literary system.

On a similar note, even though information may have discrete components, you may not be at the level where you perceive these units when you are constructing a hypertext application. In such cases, you might break information into nodes prematurely and at a later time realize that your logic was skewed. Then you would need to edit, rearrange, or retitle the information.

Unfortunately, such changes are not well-supported by all hypertext systems at this time. Virtual structures—nodes, links, or composites—would be useful in this situation. They would change dynamically when you add or delete nodes and links, depending on their descriptions. Virtual structures are similar to relational database views.

One last concern is that many hypertext systems are really only suited for new application development. Converting existing applications to hypertext is a difficult task because the file structures are so different.

Tremendous Potential

Augmenting human intellect with the help of hypertext is a grand vision indeed, one worth exploring. Hypertext applications, including interconnected writing, on-line libraries, and collaborative work environments, have tremendous potential. Current products are just the forerunners of more sophisticated applications, and we will probably see many hypertext features in mainstream software packages.

But hypertext is still in its infancy; implementation and design problems and standards issues must be resolved. Just as it takes a writer time to shape and mold a good short story, it takes time for the structure of new concepts to gel and for practical applications to emerge. But the concepts that underlie hypertext, whether they go by that name or another, will be with us for a long time to come. ■

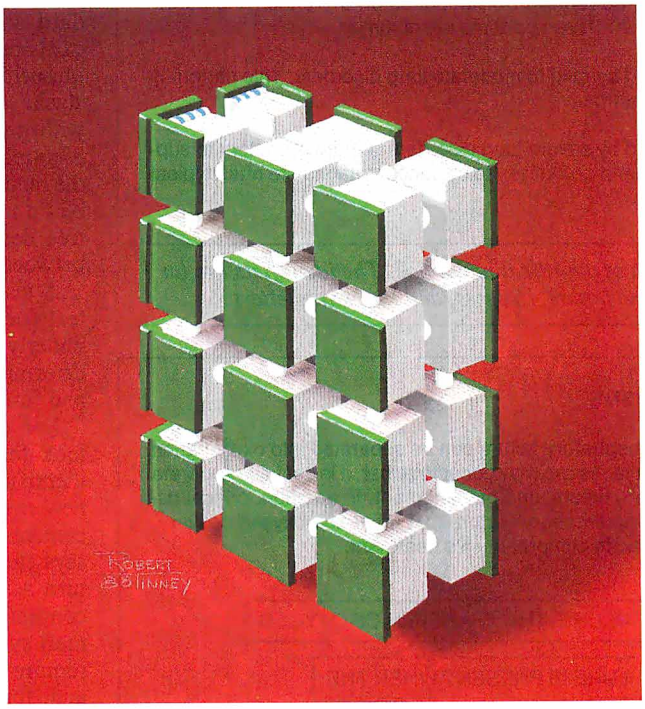
Janet Fiderio is a BYTE technical editor. She can be reached on BIX as "jfiderio."

From Text to Hypertext

Traditional tools like outline processors already incorporate many of hypertext's lessons

Mark Frisse

One reason hypertext is attracting so much attention these days is that more and more people are communicating via electronic media. Equipped with a modem and a microcomputer, you can spend a good bit of time reading electronic bulletin boards, composing electronic mail messages, contributing to group databases, and preparing large documents for printing and distribution.



However, communicating through these channels means that you often have to integrate fragments of text into your personal computer archives. You can do this by placing text items as separate entries into a file system, which works as long as you don't have too many interdependent files to manage. Hypertext programs provide another alternative for information storage.

But hypertext can be used for more than creating simple databases of 3 by 5 cards. You can also use these programs to convert on-line versions of a printed document, such as a book, into hierarchical hypertext skeletons amenable to complex user-specified interactions. The examples included in this article come from converting part of a manuscript for a medical textbook into an experimental hypertext handbook using Xerox's NoteCards. You can follow them to learn some of the tips you'll need to turn text into hypertext.

Setting Up the Cards

Hypertext lets you rearrange text. In the early systems described by Theodor Nelson and Douglas Engelbart, the basic unit of text is a single character. New documents are created by linking characters from different documents. For instance, character strings from *Romeo and Juliet* could be merged with those from *Julius Caesar* to create a new play entitled *Caesar and Juliet*.

While this approach is exciting, it isn't widely adopted in hypertext design. A second approach, which is advocated by developers of systems such as Xerox's NoteCards and Apple's HyperCard, specifies a nondecomposable data structure, often called a card. You define cards both in terms of the types of data structures they support (e.g., text, bit-mapped graphics, and video) and the operations that can be performed on the data structures (e.g., text insertion and deletion).

To map a flat-text file onto a set of hypertext cards, you first decide how much text you want to place in each card and then create a program that will perform this transfer with minimal intervention. If your text file consists of a series of E-mail items, addresses, or telephone conversation summaries, you can easily fit

continued

each item from the text file into a separate card. If an essay or other lengthy text file is not divided into logical sections, you usually must place an arbitrary number of paragraphs onto a card. You can write sentences into scrolling text fields, but this approach can detract from the power of the card metaphor.

But the organization of some flat-text files can't be characterized as either a series of discrete, unrelated paragraphs or a single lengthy text file. The medical-handbook text files used in these examples are hierarchically structured documents. A specific set of identifiers—either Roman numerals, uppercase letters, integers, or lowercase letters—pinpoints each level of the hierarchy (see figure 1). Because the text between the identifiers consists of only a few sentences or paragraphs each, it's appropriate to use these identifiers as card delimiters (see figure 2). In text files with many sentences between delimiters, you might have to obtain a line or paragraph count to decide when to start a new card.

II. Pathophysiologic mechanisms. Respiratory failure can be separated into oxygenation failure and ventilation failure. While the two may occur together, it is useful to separate them to understand their pathophysiology and management. In addition, critical tissue hypoxia may result from nonpulmonary factors that influence oxygen delivery, and these must also be considered in comprehensive treatment.

A. Oxygenation failure. The transfer of oxygen from alveolar air to pulmonary capillary blood is affected by the partial pressure of oxygen in the alveolus (PAO2), the diffusion of oxygen across the alveolar-capillary membrane, and the matching of alveolar ventilation to capillary perfusion. The five mechanisms that may lead to a low arterial oxygen tension (PaO2) are low inspired oxygen tension, alveolar hypoventilation, diffusion impairment, mismatch of ventilation to perfusion, and right-to-left shunt. The goal of oxygen therapy is to relieve critical hypoxemia. Although clinical criteria are important, serial ABGs are crucial to plan and evaluate treatment.

1. **Response to oxygen administration** depends on the underlying pathophysiology (see sec. II.A). Three patterns are common.

   a. **Hypoxemia** caused by mild to moderate lung disorders. This pattern is typical of flu and asthma.

   b. **Hypoxemia** caused by severe lung disorders is more refractory to supplemental oxygen, and potentially toxic concentrations are often typical of severe disorders.

Figure 1: *A representative section of a medical handbook. The identifiers are (in order) Roman numerals, uppercase letters, integers, and lowercase letters.*

S1.II. PHYSIOLOGIC MECHANISMS

II. Physiologic mechanisms. Respiratory failure can be separated into oxygenation failure and ventilation failure. While the two may occur together, it is useful to separate them to understand their pathophysiology and management. In addition, critical tissue hypoxia may result from nonpulmonary factors that influence oxygen delivery, and these must also be considered in comprehensive treatment.

S1.II.A. OXYGENATION FAILURE. THE TRANSFER OF OXYGEN

S1.II.B. VENTILATION FAILURE IS PRESENT WHEN THE

S1.II.C. OXYGEN DELIVERY AND TISSUE OXYGENATION

S1.II.D. INSPIRATORY MUSCLE FATIGUE OCCURS WHEN THE

Figure 2: *A hypertext card from the same section of the medical handbook. The card contains only the text delimited by the identifier II and the identifier* A. *The titles of other child cards denoted by uppercase* B, C, *and* D *are visible on the hypertext card but not on the page displayed in figure 1.*

Making the Connections

Hypertext's power resides in the links that weave isolated cards into a unique graph-structured fabric. When you transform a flat-text file into hypertext, you must recognize two distinct classes of links. The first, structural links, enforces the mapping between the conventional document and the hypertext skeleton. These links must be generated by programs, text parsers, that convert conventional text into hypertext. The second class, user-defined links, lets you create new, nonsequential paths through text.

Each collection of cards in a hypertext document has some underlying order maintained by structural links (see figure 3). The order of cards in a random-file hypertext database (e.g., recipes, phone numbers, and so on) can be arbitrary and quickly modified through standard sorting techniques. The order of cards representing a lengthy essay is sequential and usually static. E-mail stacks can be ordered by topic, and, within a topic, by date of receipt. In a hierarchically structured document like the medical handbook, card order resembles a tree. In this case, the limbs of the tree declare the structural relationships between cards.

A text-parsing program converts flat-text files into hypertext documents. The underlying structure of the flat-text file determines the text parser's complexity and function.

For example, database parsers can look for new record identifiers. These identifiers signal the system to create a new card and to copy the new record's contents to the new card. Parsers for essays can simply allocate to each card in turn as many paragraphs of text as will conveniently fit on each card. Parsers for mail systems, on the other hand, might require a list of acceptable identifiers and, for each topic, a pointer to the last card filed under the identifier. Parsers for hierarchical documents can use an ordered list of identifiers to create a hierarchical hypertext consisting only of cards and structural links.

For each possible set of identifiers, there exists only a limited number of "acceptable" tokens that lead to creation of a new card and appropriate links. For example, if you were placing text from a section labeled "V.C.2.a." into a card,

the acceptable identifiers would be any member of the set VI, D, 3, b (see figure 4). If the identifier read was D, the current subtree "C.2.a." would be popped, a new card of level "VI.D." would be instantiated, and a structural link between card "VI." and card "VI.D." would be created. Defining card identifiers appropriately is critical for simple, rapid parsing of hierarchical documents.

Although most flat-text file parsers employ ad hoc grammars, the parsing process will be simplified if document-definition language standards become more widespread. One of these, the Standard Generalized Markup Language (SGML), appears particularly promising. SGML emphasizes document structure over document appearance. For example, the standard might identify a string as a "section heading," but it would not make any statements about the section heading's font or size.

SGML also lets individuals or groups define logical structures for new document types. This flexibility increases the likelihood for standards in specialized and highly technical publishing niches. Finally, the interest shown in SGML by the federal government has encouraged the development of a number of tools for authoring, revision, and document parsing. Conceivably, these tools can be used to simplify the conversion of text already in electronic form into personal or community hypertext.

Finding the Right Card

Creating a hypertext from a flat-text file is rather simple. You exploit document identifiers to parse the file and create the new data structure. Developing the software that lets you access appropriate portions of a hypertext document is much more difficult. Most hypertexts let you access cards through several methods.

First, the initial cards in many hypertext documents contain a brief table of contents. This method provides you with an overview of the overall organization of the hypertext document. Browsers are another way you can find a card. They are useful when you want to peruse small lists or card networks. String pattern-matching is the third card-access method. It's useful when you think that the search will retrieve a small number of useful cards and very few, if any, useless cards.

Unfortunately, there are many situations where these methods are inadequate. The table of contents method fails when a card can be filed under any one of several categories, which requires you to

continued

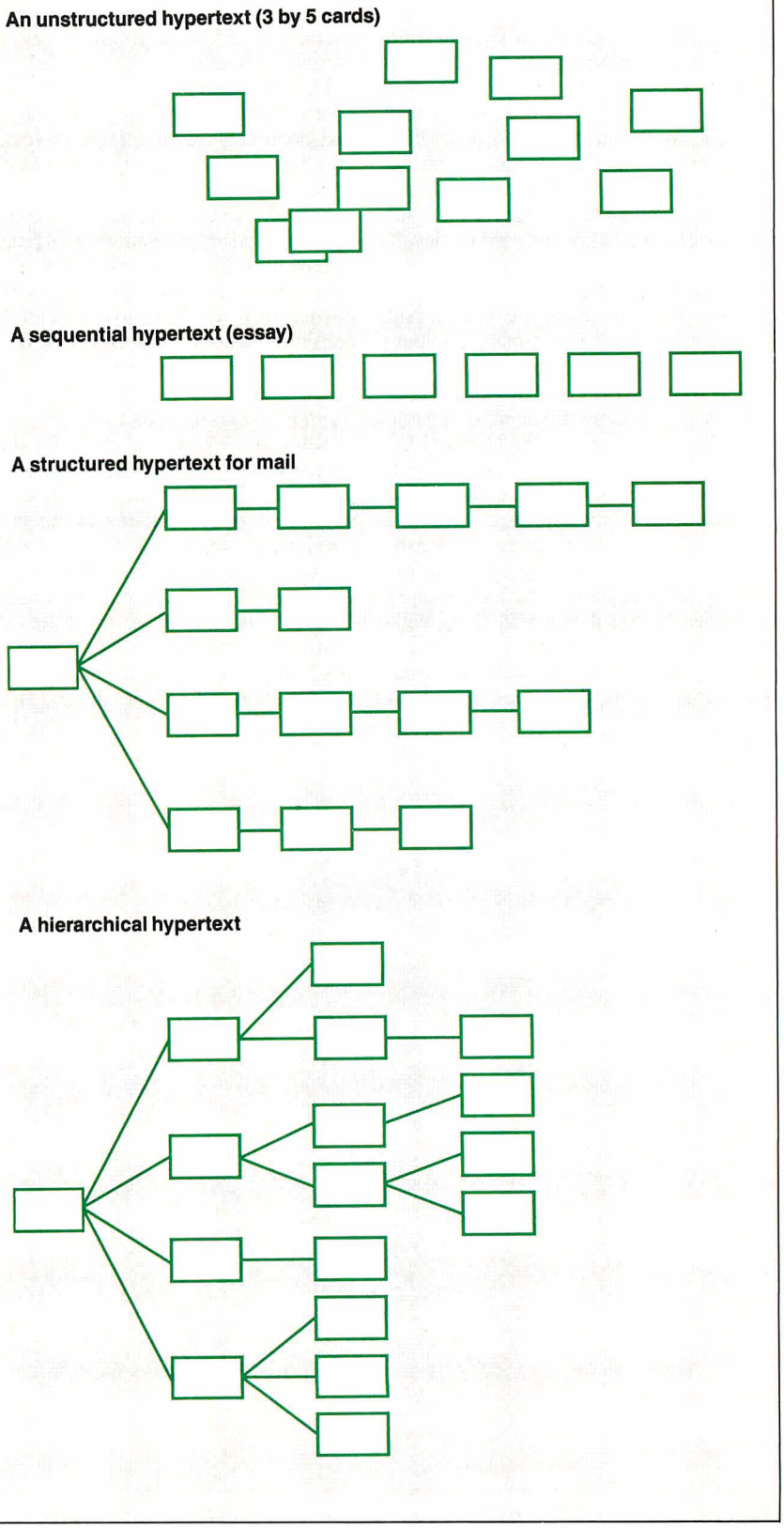


Figure 3: *Hypertext file-card arrangements. Some applications, like unsorted recipes or telephone logs, require only a random ordering. Long essays generally require a sequential ordering. Mail files resemble trees with a strong sequential component. Complex documents in outline format require hierarchical data structures.*

search many categories to find desired information. Browsing can be laborious if there are many cards to peruse, and pattern matching fails if the desired card uses a synonym (poor recall) or if there are many unwanted cards containing the same search string (poor precision). Therefore, you need alternative methods for both card indexing and card retrieval.

Hypertext Indexing

You can, however, exploit two powerful document-indexing techniques in hypertext. The first, indexing using a controlled vocabulary, classifies each document by one or more members of a finite set of descriptor terms. The National Library of Medicine's Medical Subject Headings (MeSH) system is one of the best examples of this approach.

This 15,000-term hierarchical vocabulary is used to classify most of the world's medical literature. Its principal advantage is that it is a widely agreed-upon vocabulary implemented by experts in the field of medical classification. This ensures that properly trained users, and effective programs, retrieve equivalent queries.

Unfortunately, there are two major shortcomings to indexing small hypertext documents with controlled-vocabulary terms that are developed for larger documents. First, you must create the vocabulary so that each card is classified by at least one term, and you must have a relatively uniform distribution of classification terms among all hypertext cards. Both criteria are difficult to achieve. Second, the contents of the cards using the index terms must be classified manually, a prohibitive expense for most hypertext authors.

You can use a second powerful document-indexing technique, classification with an uncontrolled vocabulary, when a structured body of index terms is not available or when cost factors prohibit the controlled-vocabulary method. The uncontrolled-vocabulary method creates inverted indexes by eliminating stop words (e.g., *the*, *are*, *a*), removing suffixes (e.g., -s, -ing, -ed), and retaining word roots as indexes into the text file.

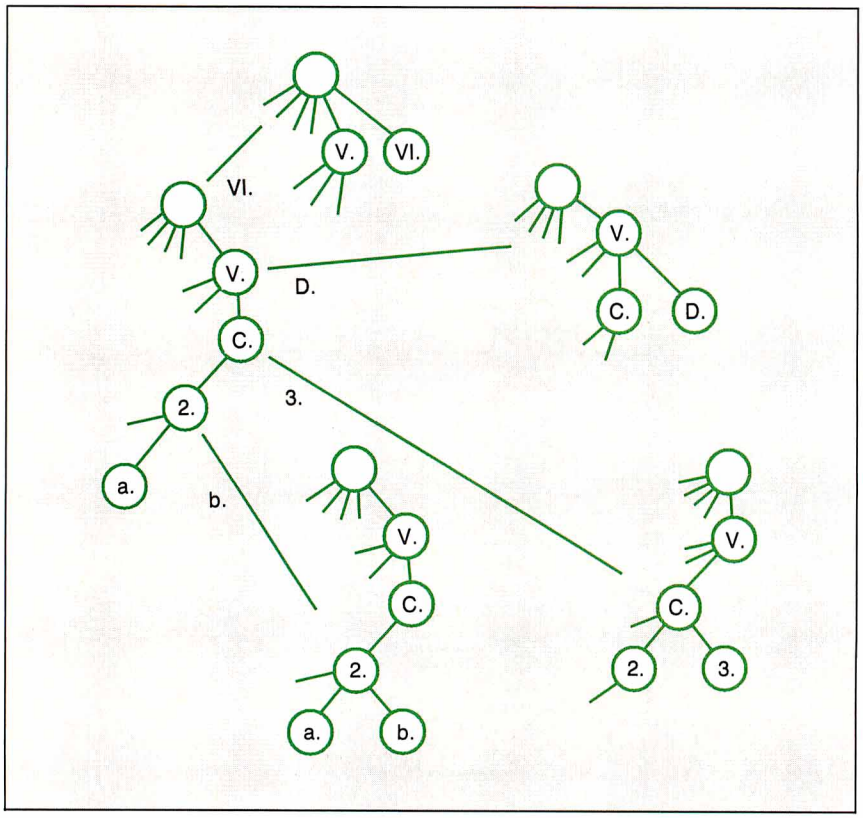


Figure 4: *The characters preceding each section of the handbook serve both as delimiters to assigning text to a card and as identifiers to control the parsing and creation of links between cards. The leftmost tree is a part of the subtree present when the active card is V.C.2.a. The arrows depict each possible new card identifier. The rightmost trees depict the trees that would result from the detection of each legal identifier.*

For example, the sentence "The lungs are inflated" creates the index terms "lung" and "inflate." In general, the index file will be about one half the size of the text file. Proponents of this approach argue that, for most domains, it is as effective to retrieve information this way as it is via controlled vocabularies. Moreover, the software needed to create these indexes is readily available. But for many applications, the space required by the indexes and the problems that occur because of misspellings and synonyms offset the benefits of indexing with an uncontrolled vocabulary.

Making the Best Match

Adding information-retrieval approaches commonly used for larger documents could make hypertext systems more powerful and responsive. How you implement these approaches depends on the structure of the underlying hypertext. In unstructured, highly modular hypertext (e.g., unrelated cards with significant amounts of free text on each card), the hypertext is really just a "folder" containing many tiny documents. In these settings, you don't need to enhance traditional free-text document-retrieval techniques. If, on the other hand, you have created a highly structured hypertext, you must exploit retrieval techniques.

In the hypertext medical handbook prototype, the power of uncontrolled-vocabulary indexing techniques that measure card content was combined with heuristic card-weight propagation functions that reflect card context. This was done so that the user could identify the "best" set of cards for browsing about a requested topic. The system doesn't try to identify a card with "the answer" to the query.

For example, if a hierarchically structured hypertext contains several potentially useful sibling cards whose parent doesn't contain text relevant to the query, you could design the system so it presents you with a sequential list of the sibling cards (see figure 5). As an alternative, it could just show you the parent card and provide a note on the card saying several of the children appear to contain useful information (see figure 6).

The good point about the second option—seeing the parent card—is that it conveys additional information concerning the context in which the various child cards are stored. This means you can better judge their relevance.

There are three basic steps to implementing this approach. First, define a utility function that calculates a card's intrinsic "utility" based on the query

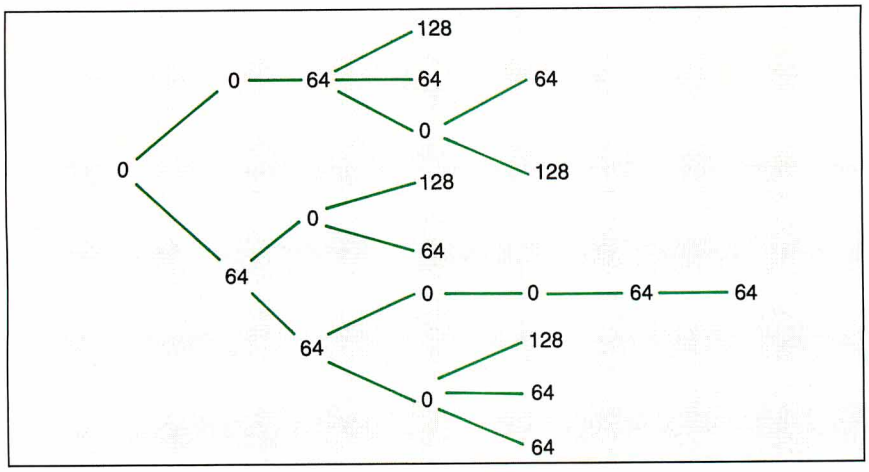


Figure 5: *This graph displays only the intrinsic card utilities resulting from a query. Cards with positive values contain one or more terms in common with the query terms. Cards with a value of 0 do not contain any of the query terms. The graph displays only structural links between cards.*

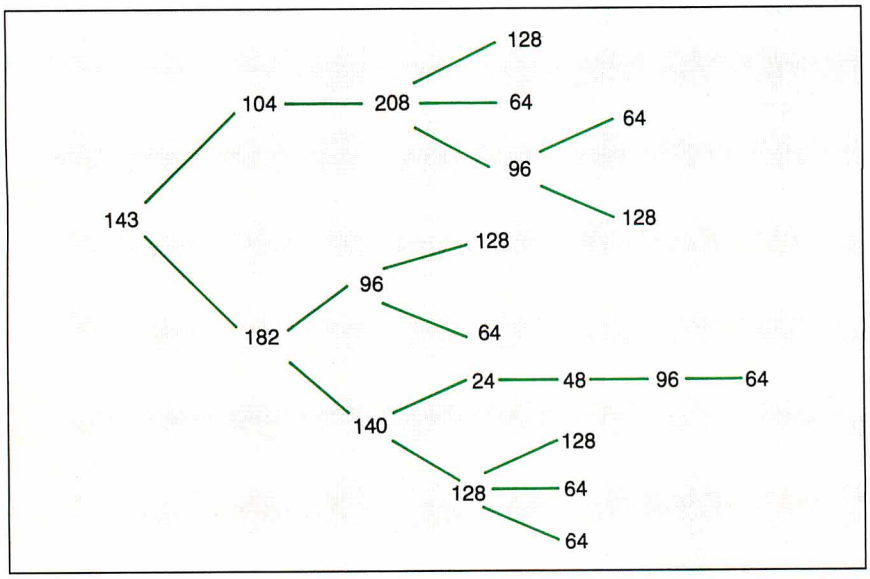


Figure 6: *Here, a graph displays the total card weights, which are the sum of the intrinsic card utilities based on card content and the propagated weights based on card context. Even in this simulation, the propagation of weights has a marked impact on the ranking of cards and suggests that the lower subtree might prove to be a good candidate for browsing.*

terms and card contents. Then, identify a method that propagates these weights to neighboring cards so that cards with contextual information are recognized. Finally, use both the intrinsic utility and the propagated weights to identify which cards should be considered candidates for graphical browsing. Let's analyze each of these steps in greater detail.

Weights and Measures

A card's utility is due in part to the correspondence between terms within the card and the query terms stipulated by the user. This component of card utility is called the *intrinsic weight*, and the value can be calculated using a modification of Salton's well-known algorithm.

This algorithm assigns term weights to cards as a function both of the frequency of occurrence of the term in the entire search space and of the number of cards containing the term. The algorithm assigns a higher weight to cards containing infrequently used terms and also to cards containing several occurrences of a term that is not found in many other

continued

cards. The formula is as follows:

$$weight_{ij} = k \times freq_{ij} \times (\log(n) - \log(docfreq_j) + 1)$$

where $weight_{ij}$ is the weight component of card i due to term j; k is a constant; $freq_{ij}$ is the number of occurrences of term j in card i; n is the number of cards in the collection; and $docfreq_j$ is the number of cards containing the term j.

A card's utility also depends on its relationship to other cards. The term *extrinsic weight* describes the component of a card's total weight contributed by propagation from neighboring cards. In the hypertext medical handbook, a card's extrinsic-weight component depends on the weights of its immediate descendant cards. The following formula represents this dependency:

$$totalweight_i = \sum_j weight_{ij} + \frac{1}{y} \sum_d totalweight_d$$

where y is the number of immediate descendants of card i, and d is an immediate descendant of card i.

This propagation function is called recursively from the leaf cards to the root card. A graphical display of the search subtree and card weights (see figure 6) serves as a road map for browsing.

Term-weight assignment and propagation allow for cards to be ranked on the basis of an estimate of their utility to the user. In general, you hope for a ranking that will produce a short list of cards that are distributed throughout the hypertext. In this situation, you can quickly explore various subtrees, jumping from one location to another. You can, however, retrieve more aberrant lists. Consider, for example, the common case where the second-highest-ranking card is the parent of the highest-ranking card.

Presenting both cards on the browsing candidate list may suggest that the cards represent two markedly different avenues for browsing rather than the actual state of term-weight predominance in a single subtree. As an alternative, you could remove the highest-ranking child card and display only the parent, under the assumption that the increased context provided by the latter outweighs the decrease in weight due to card content. However, if this process is applied recursively, you ultimately will arrive at a card list containing only the root of the tree. You can apply several heuristics to manage these aberrant cases. One of the most useful heuristics halts the replacement process when the replacing parent card is of some fractional weight of the original highest-ranking card.

In addition to the obvious traditional problems associated with uncontrolled indexing vocabularies and full-text document retrieval, the approach used throughout this article is limited in many other ways. First, the propagation function does not take into account graphs, cycles, and link types with different semantics. Second, the current approach can't exploit user feedback in any meaningful way.

It would be desirable to update card weights dynamically on the basis of user responses to the card's contents. Unfortunately, most alternative approaches to this problem (e.g., Bayesian updating or Connectionism) appear too impractical for routine use.

Learning Its Lessons

Many points about creating and using hypertext are already clear. First, you have to distinguish hypertext programs from hypertext documents. You can use hypertext programs for tasks ranging from replicating mundane 3-by-5 card files to creating complex hypertext documents consisting of multiple interrelated cards and links. This distinction is crucial, since only card content is important for simple 3-by-5 card files, but both content and context are important when creating true hypertext.

Second, it's easy to build hypertexts and add links incrementally, but it's difficult to use hypertext effectively. Even with extensive search capabilities and graphical browsers, it's not always possible to retrieve desired information or to avoid getting lost in a hypertext graph.

Third, the computational complexity of information-retrieval algorithms suggests that alternative computer architectures might be more useful.

Finally, it's clear that many problems in the field are unresolved. Will effective updating and revisions require that links be bidirectional? Can we arrive at a standardized set of hypertext card types (e.g., text, graphics, sound, and video)? Will hypertext systems provide a true advantage over other media?

Traditional tools like outline processors have already incorporated many of hypertext's lessons. Similar innovations will affect E-mail, collaborative work tools, and others in the future. ■

Mark Frisse is an assistant professor of medicine and medical informatics at the Washington University School of Medicine in St. Louis, Missouri. He can be reached on BIX as "editors."

The Right Tool for the Job

Even the systems design process falls within the realm of hypertext

Michael L. Begeman and Jeff Conklin

Hypertext is an ideal model for the systems design process. We have been working on a hypertext project, the Design Journal, to provide a systems design team with a medium in which all of their work can be computer-mediated and supported. This includes such traditional documents as requirements, specifications, high-level design, and the design document itself; it also includes scenarios, design reviews, interviews with users, designers' early notes and sketches, design decisions and rationale, internal design constraints, meeting minutes, and so on.

The Design Journal places particular emphasis on capturing the *design rationale* as the center around which to integrate all the other documentation. This rationale includes design problems, alternative resolutions (including those later rejected), trade-off analyses among these alternatives, and a record of the tentative and firm commitments made during problem resolution. We have built a running prototype of the Design Journal; it's based on the Issue-Based Information Systems (IBIS) method and is called gIBIS (graphical IBIS).

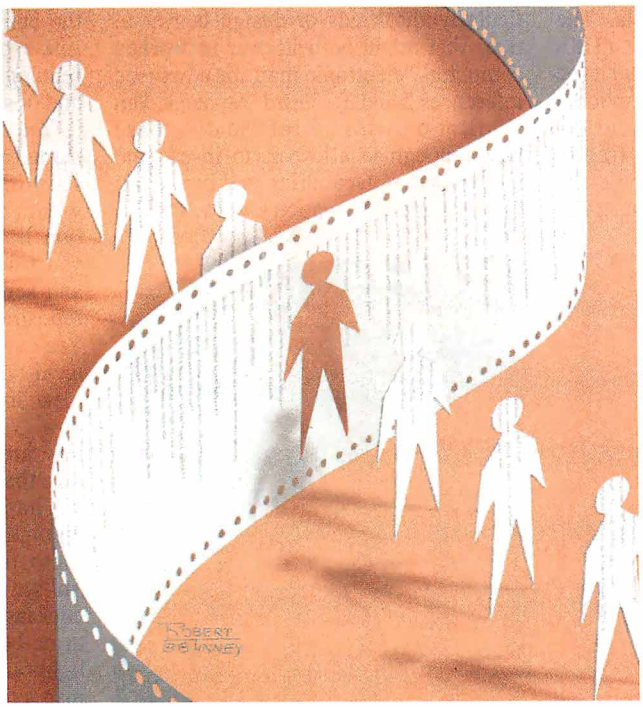


ILLUSTRATION: ROBERT TINNEY © 1988

The IBIS Method

The IBIS method was developed by Horst Rittel (see reference 1) and is based on the principle that the design process for complex problems is fundamentally a conversation among the stakeholders (i.e., designers, customers, and implementers) in which they pool their respective expertise and viewpoints to resolve design *issues*. Any problem, concern, or question can be an issue and may require discussion (if not agreement) for the design to proceed. In the IBIS model, this argumentation constitutes the design process.

IBIS focuses on articulating the key *Issues* in the design problem. Each Issue can have many *Positions*. A Position is a statement or assertion that responds to the Issue. Often Positions are mutually exclusive, but they needn't be. Each Position, in turn, can have one or more *Arguments* to support or object to it. Thus, each separate Issue is the root of a (possibly empty) tree; its children are Positions, and their children are Arguments.

There are nine kinds of links in IBIS. For example, a Position *Responds-to* an Issue, and this is the only place you can use the Responds-to link. An Argument either *Supports* or *Objects-to* its Position. Issues can *Generalize* or *Specialize* other Issues, and they can also *Question* or be *Suggested-by* other Issues, Positions, and Arguments. (The remaining two links are *Replaces* and *Other*.)

A typical IBIS discussion begins when someone posts an Issue node containing

continued

a question such as "How should we do X?" That person can also post a Position node proposing one way to do X, as well as some Argument nodes to support that Position. Someone else can post a competing Position responding to the Issue and can support the Position with Arguments, and so on. New Issues that the discussion raises can also be posted and linked into the nodes that most directly suggested them.

There is no stopping rule, nor is there a particular way of registering Issue resolution by agreeing on some Position. The goal of the discussion is for each stakeholder to try to understand the elements of the others' proposals, and perhaps to change the others' minds. The method inhibits unconstructive rhetorical moves, such as argument by repetition and name calling, and supports more constructive moves, such as seeking the central issue, asking questions and giving answers, and being specific in supporting your own viewpoint.

In implementing gIBIS, some changes and extensions have been made to allow needed flexibility, but the method has been changed as little as possible. The extensions to IBIS in the current gIBIS tool are three: an additional Other type for nodes and links, as an escape mechanism when you can't find a way to express a thought within the IBIS framework; an additional External type for nodes that contain non-IBIS material, such as requirements, documents, design sketches, or code; and the ability to let Positions specialize or generalize other Positions, and to let Arguments specialize or generalize other Arguments.

The gIBIS Tool

Three technological themes guided our design of gIBIS. First, we wanted to explore the capture of the rationale behind the design. Second, we wanted to support computer-mediated teamwork, particularly the kinds of design conversations that might be held over networked computers, electronic mail, or news (see references 2 and 3). Third, we needed an application with an information base large enough to allow us to investigate the navigation (searching and browsing) of very large information spaces.

The browser lets you see the IBIS graph structure, its nodes, and their interconnecting links.

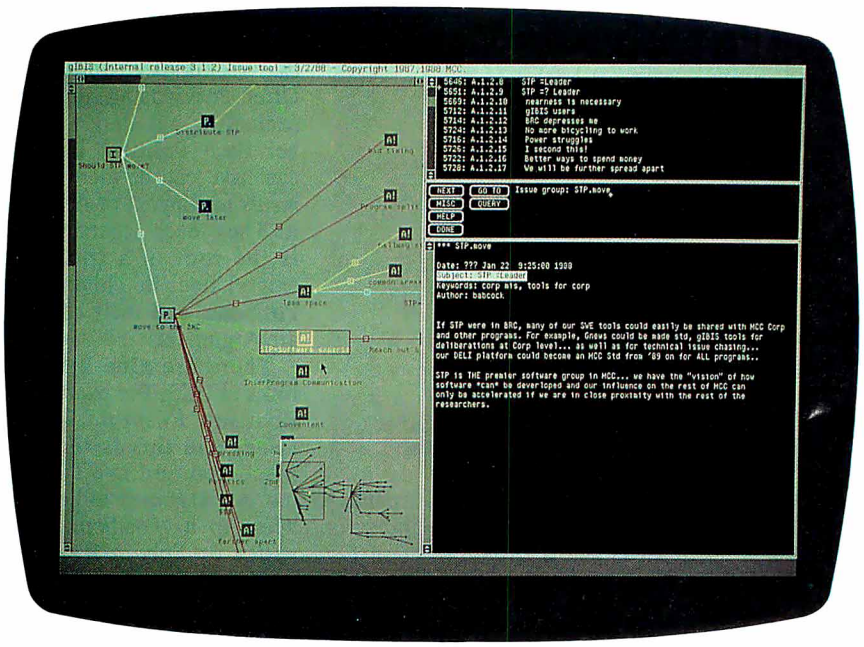


Photo 1: *The gIBIS interface. Note the graphical browser (left), the structured index (upper right), the control panel below the index window, and the inspection window (lower right).*

The pattern of gIBIS usage falls into two categories: Some people use the tool primarily as an isolated hypertext tool for structured thinking and design, while others use it primarily as a vehicle for structured communication.

The basic gIBIS interface is divided into four windows (see photo 1): a graphical browser on the left, a structured index into the nodes on the top right, a control panel below the index window, and an inspection window in which to view the attributes and contents of nodes and links. This interface is somewhat unusual among hypertext systems: To view the contents of a node or link, you must select it, and the contents will display in the inspection window.

The Browser

The browser lets you see the IBIS graph structure, its nodes, and their interconnecting links. Most of it is dedicated to a local view of the network: a zoomed-in view of the current area of interest, with nodes and links in full detail. The lower-right portion of the browser contains a global overview: a zoomed-out view of the entire network without node labels, link-type icons, and secondary links. A rectangular overlay indicates the scope and position of the current local view.

You can scroll the browser window by using traditional scroll bars or by "snap scrolling" (clicking the mouse anywhere within the local view to center that location in the window). This method lets you fine-tune the position of the display and scroll diagonally without having to reposition two independent scroll bars. You can also scroll to an area outside the local view by repositioning the local-view indicator in the global-view window. You simply drag the rectangle to a new area within the global view to update the local view.

The browser supports a direct-manipulation-style interface (see reference 4) to the display objects (nodes and links). You select a display object by clicking on it with the left button of your three-button mouse. The browser highlights and boxes it, puts its contents in the inspection window (see photo 1), and scrolls its index line to the top of the index window. A right-button mouse click displays context-sensitive menus that let you create, edit, delete, and move objects.

For example, if you press the menu button without selecting an object, a menu appears indicating that the only legal operation you can perform is Issue creation (i.e., the beginning of a new

continued

IBIS structure). By contrast, if you select an Issue node, the menu changes to reflect the legal operations on Issues. If you create a follow-up Position node, it is placed next to the Issue and linked to it with a Responds-to link. Then, the inspection window divides in half and a Node Creation window preloaded with a structured template appears beneath it.

You fill in the template's structured fields (e.g., Subject, Keywords, and so forth) and provide an optional description of the node's topic (i.e., an unstructured node body). When the node is complete, you push the Submit button in the control panel (which appears only during Node Creation and Editing); the node is then parsed and stored, and the browser and index windows are updated to include it.

When you follow the "Link to another node" menu item, you can choose from the set of legal outgoing link types for the current node, and the new link appears stretching between the source node and the current mouse position. You move the mouse to the destination node (by "rubber banding") and then drop the end of the link there.

You can also select canonical IBIS subnets (i.e., a single Issue followed by its Positions, and their Argument nodes) as a single entity. The gIBIS tool supports the movement and automatic layout of these subnets as wholes. Further, it lets you gather a subnet into a single composite Issue-Position-Argument (IPA) node; this node provides additional structure to analyze competing Positions and commit to one of them (Issue resolution).

While it has a structure and body all its own, the IPA node by default inherits its label, subject, and keywords from the root Issue of the underlying subnet. Selecting the composite means traversing the underlying subnet and composing an "inherited" body, which is shown in the inspection window along with any composite-specific text (see figure 2). Since the inherited body can become quite long with a large subnet, a function key lets you suppress (or reveal) it in the inspection window.

The node-index window provides an ordered, hierarchical view of the nodes in the current network.

The Node-Index Window

The node-index window provides an ordered, hierarchical view of the nodes in the current network. To traverse the network, you follow Primary links in depth-first order starting from each Issue. The Issues, Positions, and Arguments are given sequence numbers like you might find in an outline editor (see reference 5). For example, the Subject line for Issue 8 is I.8; it has no children, so that's all there is. The Subject line for Issue 9 is I.9; its first Position node (P.9.1) has two Argument nodes (A.9.1.1 and A.9.1.2), and so forth. Issues are ordered by creation date. The view-configuration panel lets you tailor the index to reflect by Subject, Author, Keyword, or node Label.

You can select nodes through the index as well as the browser. Clicking on a node's index line makes that node current: Its icon is highlighted in the browser, the window is scrolled, if necessary, to bring it into the local view, and its contents appear in the inspection window. This browsing method provides a linear, compressed view of the data in the network.

The Control Panel

The control panel is composed of a set of buttons that extend gIBIS's functionality beyond simple node and link creation. Each button hides a menu that extends or tailors its basic function. The Next button, for example, normally records that you have read the current node before it displays the next one. But if you press the right-hand mouse button while over the Next button, the hidden menu will appear. This is a slight extension of basic

continued

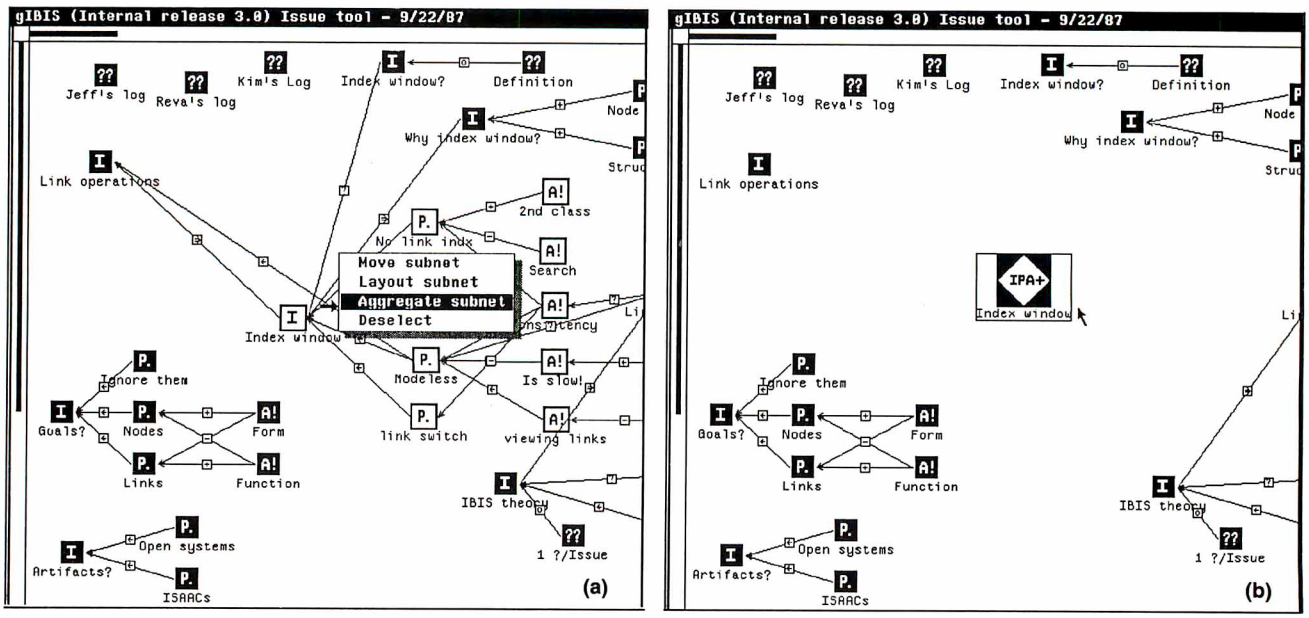


Figure 1: *A canonical IBIS subnet* **(a)** *before and* **(b)** *after aggregation. Since the "inherited" body can be quite long in a large IBIS subnet, a function key lets you suppress or reveal the inherited text body in the inspection window.*

Problems in Paradise

One common but subtle difficulty in hypertext systems is that sometimes it's unnatural to break your thoughts into discrete units, particularly if you don't understand the problem well and those thoughts are vague, confused, and shifting. With gIBIS, this effect is pronounced, because the IBIS method imposes on you a rather austere selection of node and link types; gIBIS makes you think within a particular framework (i.e., you focus on Issues without necessarily resolving them), and this can be disruptive.

The early phase of considering a writing or design problem is critical and fragile and must be allowed to proceed in a vague, contradictory, and incomplete form for as long as necessary (see references 1 and 2). However, any insights should be captured, and gIBIS should support the emergence of a coherent structure as it develops.

Design conversations often feature commitments like "Let's try X—it has advantage Y." This is a Position and a supporting Argument, but no Issue is mentioned. Since you don't always see the Issue or Position immediately, it would be nice to have a "proto-node" in which to record ideas, snippets of text, and perhaps graphical sketches before having to structure them.

Ultimately, of course, it's valuable to have separated these elements into Issues, Positions, and Arguments. But when you're struggling to solve a problem, the mental effort required to separate it into discrete thoughts, identify their types, label them, and link them can be prohibitive.

Finding the Right Answer

In the IBIS method, you resolve an Issue by selecting one of the Positions that respond to it as being "the right answer," or at least "the Position we are committing to for now." You could mark the Position node as Selected and display it by marking such nodes distinctively in the browser, perhaps with a somewhat different color.

We have recently added an Issue resolution feature to gIBIS. It combines indicating resolution with the aggregation into Issue-Position-Argument nodes; once an Issue is part of an IPA node, you can resolve it. At the moment, you change the value of the Resolved field to True and indicate which Position holds the resolution.

The rationale for adopting a particular conclusion may require more explanation; for instance, perhaps all the argumentation didn't occur within gIBIS. Sometimes, resolving an Issue transcends the original options. Such resolutions may combine elements of the original options and abandon prior assumptions or presuppositions. Sometimes, when a breakthrough occurs, it's clearly the right solution.

The gIBIS tool needs to allow such leaps in argumentation and not force the Issue to a well-structured resolution. This may be as simple as providing the free-text annotation of an IPA tree or the marking of some discussions as "irrelevant in light of Position X."

Getting the Whole Picture

Using hypermedia for cooperative work has its problems as well. Sometimes, an unexpected problem can emerge when several users work cooperatively in a shared Issue group. Unless each author writes clearly and completely, while you might understand the individual nodes, it's hard to follow the thread of thoughts as it winds through several dozen nodes. That is, the hypertext tool forces the author to express ideas in a fined-grained, separated manner, and this obscures the larger idea being developed.

This is a familiar problem common to many hypertext systems: The freedom of choice inherent in branching documents requires greater care from both the author and the reader. The separation of Position and Argument in IBIS (i.e., an idea and its justification) could also be another factor.

However, there may be a more subtle issue here: Traditional linear text provides a continuous, unwinding context thread as ideas are proposed and discussed—a context that the writer constructs to guide you to the salient points and away from the irrelevant ones. Indeed, a good writer anticipates questions and confusions that you may encounter and carefully crafts the text to prevent them.

The hypertext (or at least the gIBIS) author, however, is encouraged to make discrete points and separate them from their context. Sometimes, the gIBIS author, in a hurry to capture a design Issue and its analysis, may write only the bare minimum necessary to record the essence of the Issue, Positions, and Arguments. Even the careful author, however, may not anticipate all the routes to a given node, and so may fail to develop the context sufficiently to clarify its contents.

Using a *path* may linearize a network's segments sufficiently to provide context (see references 3 and 4). And there are higher-level constructs that aggregate a set of nodes. The new IPA-node type combines all of an IBIS subtree's nodes (an Issue, its Positions, and their Arguments) into a single node and lets you append additional IPA-specific text as well. This linearizes the discussions of individual Issues and reduces the sense of fragmentation you sometimes have when reading a gIBIS network, but it's probably not sufficient to restore the context in which those nodes were created.

Finally, part of the context is the relative importance of the points presented, and we need to incorporate an "importance" meter into gIBIS nodes. One possibility would be to incorporate one of three keywords, HI IMPORTANCE, MED IMPORTANCE, or LO IMPORTANCE, into each node at creation. This would guide you to the most salient points first (see also reference 5); it could also be used to control the level of clutter in the browser display.

Staying on Track

It's common in conversations to "go meta" and make a comment on the *process* (as opposed to the content) of the discussion (i.e., "But that isn't the issue here"). Similarly, in IBIS discussions, sometimes you need a meta-discussion when one person in an Issue group feels that another has misused the IBIS structure to present ideas. For example, if B feels that A's Issue node is actually two Issues and a Position, B needs a way to express this and to initiate a discussion about it.

There are three levels of collaborative work: ***substantive*** (the content of the work), ***annotative*** (comments about substance), and ***procedural*** (comments about procedures and conventions) (see reference 4). In IBIS, you can theoretically treat all three levels as Issues. For

example, B could post an Issue, connected by a Questions link to A's Issue, asking "Isn't this really two Issues and a Position?" While this is a valid move, it has drawbacks.

B's Issue is by its nature meta-substantive, although whether it's annotative or procedural is unclear. But by placing it in the network, B creates an Issue that adds complexity to the browser display without shedding any light on the problem being discussed; B also initiates a discussion that may change the network, after which this meta-discussion will have only historical interest.

This problem has several resolutions. You could have special meta-level Issue, Position, and Argument nodes to distinguish them from substantive ones. Or you could label nodes as "only of historical interest." Such nodes could be archived or have their display suppressed so they wouldn't normally be visible. You could also give each node its own meta-layer (only displayed on request) for such discussions. In a simple version of this option, you can append a meta-line at the end of any gIBIS node and then begin an annotative or procedural discussion there. The node's author might append a response or revise the network to correct the structural error.

Lost and Found

A hot issue in hypertext research is how to use a graphical browser effectively to navigate networks with more than a few dozen nodes. This is part of the more general problem of disorientation, particularly its visual and spatial aspects in a large data space. Although gIBIS has addressed this problem with its global-view and query mechanisms, many hypertext systems have not.

Keeping Current

Any database must be able to manage changes to its data. Often, a versioning scheme that allows older versions of the data to be marked and archived is used. In gIBIS, the issue of change is of unusual importance, because the very nature of an "Issue base" is its use for evolving discussions in which older material may be accurate and highly important, inaccurate and of only historical interest, or anything in between. For example, the original form in which an Issue is framed may be biased toward a particular Position, or it may contain a presupposition that is later made explicit and rejected. How can you handle this "outdated" form of the Issue?

Sometimes, the Issue and its discussion subnet may be isolated and wrong; then it's easy to decide to archive that subnet and delete it. But more often, parts of the subnet will be wrong, misleading, or irrelevant, while others are still relevant or important and part of an active region of the network. How do you prevent these partially invalid segments from poisoning the network?

Perhaps you could systematically indicate the age and relevance of network material by, say, displaying older nodes as yellowed or frayed (unless they have been recently visited and updated). Like importance, salience, and confidence, age and relevance are somewhat subjective measures and can be only partially automated. Another possibility for managing change is completely human: As Issue networks grow in size and importance, organizations should have people whose job is to maintain the currency and consistency of the Issue base.

REFERENCES

1. Brown, J. S. "Notes Concerning Desired Functionality, Issues and Philosophy for an AuthoringLand." Xerox PARC CIS Working Paper, 1982.
2. Smith, John B., Stephen F. Weiss, Gordon J. Ferguson, Jay D. Bolter, Marcy Lansman, and David V. Beard. "WE: A Writing Environment for Professionals." Technical Report 86-025, Department of Computer Science, University of North Carolina at Chapel Hill, 1986.
3. Bush, Vannevar. "As We May Think." *Atlantic Monthly*, July 1945, pp. 101–108.
4. Trigg, Randall, Lucy Suchman, and Frank Halasz. "Supporting Collaboration in NoteCards." Proceedings of CSCW '86: The Conference on Computer-Supported Cooperative Work, MCC/STP, Austin, Texas, December 1986.
5. Lowe, David G. "Cooperative Structuring of Information: The Representation of Reasoning and Debate." International Journal of Man-Machine Studies, vol. 23, 1985.

functionality and leaves the current node marked unread.

For those functions with no extensions, the menu provides a longer explanation of the button's functionality. For example, the Goto button loads a particular Issue group's data into the browser; it hides a Help menu that tells you to "enter an Issue group name and push this button."

The Misc button hides a grab bag of functionality. For instance, the Tool Config item lets you tailor particular aspects of the interface. If you select it, a new window appears that contains the gIBIS configuration parameters, their current settings, and any constraints on their legal settings. These parameters are divided according to whether they affect the index, the browser, or the inspection window.

Primary and Secondary Links

When a node is created, it's usually automatically linked into the existing network of nodes. This automatic first link is its *primary* link. Later, you may connect that node to others in the network, but all subsequent links are *secondary* and differ from the primary one both visually and navigationally.

Filtering out the secondary links from a canonical IBIS subnet results in a hierarchy that becomes the basis of the index window's structured listing. For example, let's say that three Positions respond to an Issue, and two of them have supporting Arguments. The Positions are mutually exclusive, so each Argument also objects to the other Positions; hence, secondary links make these connections explicit.

It's easier to understand the IBIS network if, on first pass, the browser displays only the primary links and "turns off" the secondary links. The Next button leads you through the network in the canonical IBIS order (the same sequence as the index window). The primary-link view shows clearly how the current node relates to the surrounding conversational structure. After the first pass, you can make the secondary links visible, if you wish, to see the cross-relationships encoded in the network. (In keeping with the design philosophy of tightly coupled windows, selecting a node with the Next button causes the same scrolling and highlighting as selection via the browser or index window.)

The Use of Color

We designed gIBIS for use on Sun workstations with color monitors. Thus, color

continued

is used to indicate node- and link-type information, as well as such special node states as "currently selected" and "matches the current query." You can also configure gIBIS to customize the color mapping.

This flexibility caused some trouble at first, and we quickly added a set of standardized color mappings. Having colored nodes and links turns out to be one of the most compelling aspects of gIBIS. You can quickly learn the type mappings for the most commonly used nodes and links, and type identification then becomes a rapid, reflexive activity. While you may occasionally change your mappings with the Tool Config panel for special purposes (like making some links invisible for presentations), most users commonly set up their colors and leave them alone.

If you have a monochrome monitor, the information encoded by color is duplicated with icons. While gIBIS by default presents both color and icons, both can also be suppressed. Usually, the color-monitor user suppresses the link icons to make the browser view appear less cluttered.

Using color presents its own set of problems, however. For one, you must have a color display. And you are limited to a small number of color mappings. The gIBIS tool contains nine link types and is probably near the limit of people's ability to reliably perform the mapping. By adding the link-type icons, the mapping complexity drops, and more link types could be safely added.

More surprising, however, is the large machine-to-machine variation among color monitors in overall brightness, convergence, and RGB-gun saturation. This variation has eliminated the possibility of using a single, standardized set of color mappings for all machines. The color settings that produce bright, highly defined images on one screen can be dark, muddy, and indistinct on another. To address this, the four sliders at the bottom of the Tool Config window let you fine-tune the color map to your machine.

Search and Query

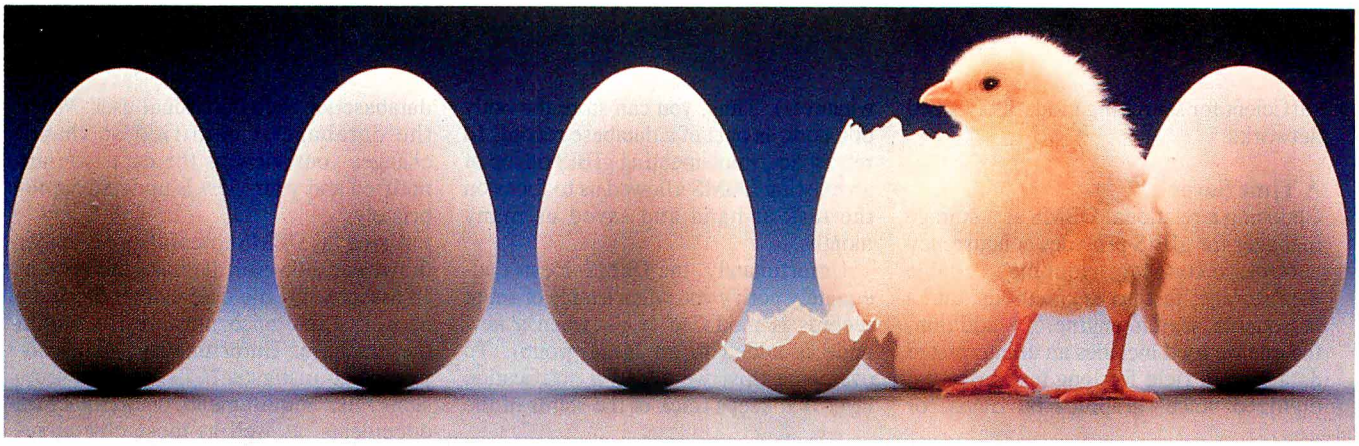
Another control-panel feature is the Query button. Pressing it brings up a small query-construction window. It contains a small control panel and a specification section for "query by example," which lets you create a proto-node against which the nodes in the current IBIS net will be matched. When you press the Execute button, the query is parsed and evaluated, and its results displayed in both the browser (selected nodes turn a bright yellow in both the local and global views) and the index window (the window shows only the index lines for those nodes satisfying the query).

You can then examine those nodes using standard navigation techniques. Pressing the Help button reveals another window (obscuring the browser window), which contains instructions on how to formulate queries, their appropriate grammar, and a number of examples.

This query-specification technique lets you formulate node-content searches based on the logical AND of predicates over node attributes. The grammar could be extended to allow full Boolean expressions over the predicates, but there has been little demand for it. These more sophisticated queries may be required when the networks become very large, but the simple query engine in gIBIS is

continued

TRY THE NEW GENERATION FOR $30



KNOWLEDGEPRO
KnowledgePro uses Topics to store "chunks of knowledge." Topics can contain data, hypertext, procedures, calculations, rules, lists and pictures. Using a dozen simple commands, non-programmers can use topics to explain complex procedures, rules or recommendations. Using the other 100 plus commands, professional programmers can create sophisticated expert system tools and applications quickly and easily.

HYPERTEXT
Hypertext can be a powerful tool for organizing text, graphics and data, but without an underlying structure the user becomes lost in a maze of information. KnowledgePro adds structure, control and intelligence to create an exciting new teaching medium.

Once you've used KnowledgePro you'll never go back to your shell!

Q. Who's using it?
A. Engineers, Educators, Lawyers, Scientists, Managers, Authors, Bankers, Software Developers, Expert System Developers, Computer VARs and VADs, Trainers, Consultants, Experts in Agriculture, Manufacturing, Insurance, Petroleum, Government and many many more.

Q. What are they doing with it?
A. Intelligent tutorials, smart manuals, procedure guides, rule books, computer aided instruction, sales and promotion, data analysis, non-linear documents, text analysis, diagnostics, software front-ends, expert systems, training and education, hypertext authoring, case studies, insurance claim determination, investment analysis, intelligent forms - there seems to be no limit to the diversity of applications.

Q. What can I do with the demo system?
A. The KnowledgePro demonstration system comes with a 100 page manual and lots of examples to get you started. You can create and save small working knowledge bases. The only commands that you can't use are those for handling external files or chaining knowledge bases. We even credit your $30 toward the cost of the full system.

Q. How much is the full development system?
A. KnowledgePro costs $495 and there are no run-time charges, so you don't have to pay more when you distribute your applications. The Database Toolkit (for access to dBASE and Lotus 123 files) costs $49 and the Graphics Toolkit (for access to PC Paintbrush pictures) costs $89. Our KnowledgeMaker induction system (for creating rules from data) costs $99. KnowledgePro runs on IBM PC, AT and PS/2 compatible machines with 640K memory.

TO ORDER Call 518-766-3000 (Amex, Visa, M/C accepted) or send $30 + $5 shipping & handling for the demo ($38 total foreign) or $495 + $8 shipping & handling for the full system ($553 total foreign) to Knowledge Garden, Inc., 473A Malden Bridge Road, Nassau, NY 12123. In NY State please add 7% sales tax.

KnowledgePro®
By Bev & Bill Thompson
The first Knowledge Processor.

published by Knowledge GARDEN Inc.

In association with McGraw Hill

KnowledgePro is a registered trademark of Knowledge Garden, Inc., Lotus 123 is a registered trademark of Lotus Development Corp., dBASE is a trademark of Ashton Tate. IBM is a registered trademark of International Business Machines Inc., KnowledgeMaker is a trademark of Knowledge Garden Inc. Photo Tcherevkoff ©

Circle 136 on Reader Service Card

sufficient for searching moderately sized networks.

A Time Saver

Choosing a relational DBMS as a storage manager for gIBIS provides concurrency control, record-level locking, reliable data storage, fast access methods, and a reasonable search engine. In addition, the one we used includes an uninterpreted data type (a field for long text passages, digitized voice, graphics bit maps, or whatever). Thus, you can store the body of a node as part of a database record. In retrospect, implementing gIBIS on top of an existing DBMS allowed us to focus on the task at hand and saved us many months.

Unfortunately, the DBMS doesn't adequately notify you when a table or set of records is modified (e.g., when a new node is added to an Issue group). To overcome this, we added a notification layer that keeps track of the status of the database for each individual user. When the database is modified so that it changes your view of the data, you are notified and your view is updated appropriately.

Using a DBMS presents one major drawback, however: It closes the system. In essence, all the objects that the Issue networks reference must reside within the database. Unfortunately, many objects that instigate Issue-based discussions, like requirements or architecture documents, as well as those objects resulting from these networks, such as code and documentation, are external to the database. A special *surrogate* type of node lets gIBIS reference external objects, such as text, graphics, or spreadsheets, in a "blind faith" sort of way.

A surrogate has two parts: a pointer to the external object (usually a fully qualified path name) and the name of an optional display program that gIBIS should invoke to display the object. If you don't specify a program name, the default display program assumes that the external object is a text file and loads it into the standard inspection window. If you specify a display program, gIBIS invokes it and passes it the external path name as an argument.

A Useful Structure

IBIS is a powerful method for research thinking and design deliberation. If you're working alone, the Issue-Position-Argument framework helps to focus your thinking on the hard, critical parts of a problem and to detect incompleteness and inconsistency in your thinking. If you're collaborating in an Issue group, the structure gIBIS imposes on discussions is useful and exposes axe grinding, hand waving, and clever rhetoric. It has a tendency to make assumptions and definitions explicit.

You can trace some of these advantages to the *semi-structured* nature of IBIS networks (see reference 6). The writer can structure a complete message without any constraints on what is said, while the reader has a recurrent structure in the text that aids search and comprehension. The explicit rhetorical structure of IBIS reveals at least the general structure of an unfolding discussion. Indeed, a distinct advantage stems from the particular structure that IBIS provides. That is, a good match exists between some of the cognitive structures and processes of design and the three node types and nine link types that compose IBIS.

However, we have found some major shortcomings in gIBIS. There is no spe-

continued

cific node type for goals and requirements. There is no particular support for making a decision (or reaching a consensus) among the various Positions of an Issue, and no way to indicate that such a decision has been made. Design decisions usually result in adding solution elements to the design itself (e.g., code, module structure, and so on), but these elements are not supported by gIBIS and must be stored externally. (For further discussion of these and other shortcomings, see the text box "Problems in Paradise" on page 260.)

A Synergy of Tool and Method

The noncomputerized IBIS method is cumbersome and would not have reached the popularity that it has here in our lab without the gIBIS tool to support it. Although gIBIS is not the only hypertext system available in our environment, it has achieved wider and more prolonged usage in a much shorter time than has PlaneText, the other system (see reference 7). We speculate that this is due to a particularly good match between the requirements of the IBIS method and the hypertext facilities of the gIBIS tool.

For example, one clear success has been in using color to indicate the types of the IBIS nodes and links. Perhaps this is partly because there are only a few distinct node and link types in IBIS, and each has reasonably well-defined semantics, so the browser display can use bright primary colors that, after a while, become strongly associated with their meanings. Despite its narrow design and rigid functionality, gIBIS provides facilities that are easy to learn and quite helpful with ill-defined design problems. ■

REFERENCES

1. Rittel, H., and W. Kunz. "Issues as Elements of Information Systems." Working paper no. 131. Institut für Grundlagen der Planung I.A. University of Stuttgart.
2. Eveland, J., and T. Bikson. "Evolving Electronic Communication Networks: An Empirical Assessment." Proceedings of CSCW '86: MCC/ACM conference on computer-supported cooperative work, 1986.
3. Horton, M., and R. Adams (Center for Seismic Studies, Arlington, Virginia). "How to Read the Network News." Distributed by Mr. Adams quarterly over the Usenet news network.
4. Norman, D. A., and S. W. Draper. *User Centered System Design*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1986.
5. Hershey, William. "Idea Processors." BYTE, June 1985.
6. Malone, T., K. Grant, K. Lai, R. Rao, and D. Rosenblitt. "Semi-Structured Messages Are Surprisingly Useful for Computer-Supported Cooperation." Proceedings of CSCW '86: MCC/ACM conference on computer-supported cooperative work, 1986.
7. Conklin, J. "Hypertext: A Survey and Introduction." *I.E.E.E. Computer*, vol. 20, no. 9, September 1987.

ACKNOWLEDGMENT

This is a shortened version of a paper to appear in ACM Transactions on Office Information Systems, *vol. 6, no. 4.*

Michael L. Begeman and Jeff Conklin are members of the MCC Software Technology Program (Austin, TX) and the authors of gIBIS. They can be reached on BIX as "editors."

Hyper Activity

HYPERTEXT PRODUCTS

Business FileVision $395
Macintosh
Marvelin Corp.
3420 Ocean Park Blvd.
Suite 3020
Santa Monica, CA 90405
(213) 450-6813
Inquiry 958.

Document Examiner
Feature of Genera software environment that comes bundled with Symbolics workstations
Symbolics, Inc.
11 Cambridge Center
Cambridge, MA 02142
(617) 621-7500
Inquiry 957.

Graphic KRS (Knowledge Retrieval System) $400
Workstations
Text KRS $300
Workstations
Hyper KRS $3000
(includes 1 Hyper Indexer and 10 Hyper KRS)
Additional workstation copies $125
Knowledgeset Corp.
60 Garden Court, Building A
Monterey, CA 93940
(415) 968-9888
Inquiry 963.

Guide
Mac $199.95
IBM PC, AT, PS/2s $275
IBM XT $300
Owl International, Inc.
14218 Northeast 21st St.
Bellevue, WA 98007
(800) 344-9737
(206) 747-3203
Inquiry 959.

HyperCard $49
Mac Plus, SE, and II
Apple Computer, Inc.
20525 Mariani Ave.
Cupertino, CA 95014
(408) 996-1010
An assortment of public domain stacks for HyperCard is available on BIX in the "stackware" area of the "listings" conference.
Inquiry 960.

KMS (Knowledge Management System) $1995
Sun 3, 386i, and 4 workstations; Apollo DN 3000 and DN 4000 workstations
Scribe Systems, Inc.
Commerce Court, Suite 240
4 Station Square
Pittsburgh, PA 15219
(412) 281-5959
Inquiry 961.

Knowledge Pro, a knowledge processor $495
IBM PC, XT, AT, and PS/2s under MS-DOS
Knowledge Garden, Inc.
473A Malden Bridge Rd.
Nassau, NY 12123
(518) 766-3000
Inquiry 962.

MacSMARTS $195
Mac 512 or higher
MacSMARTS Professional $495
Mac 512 or higher
Cognition Technology Corp.
55 Wheeler St.
Cambridge, MA 02138
(617) 492-0246
Inquiry 964.

Marcon, a DBMS with hypertext-like indexes $495
IBM AT or higher
Marcon Plus $795
IBM AT or higher
AIRS (Automated Information Reference Systems), Inc.
335 Paint Branch Dr.
College Park, MD 20742
(301) 454-2022
Inquiry 965.

HYPERTEXT RESEARCH AND DEVELOPMENT

Bell Communications Research (Bellcore)
435 South St.
Morristown, NJ 07962
(201) 829-2000
Superbook, *a text browser*
Telesophy, *on-line literary system*
Thoth II, *a system that embeds semantics into hypertext*

Brown University
Institute of Research in Information and Scholarship
P.O. Box 1946
Providence, RI 02912
(401) 863-2001
Intermedia, *an interactive teaching and learning environment (in development)*

Carnegie-Mellon University
Computer Science and English Departments
Pittsburgh, PA 15213
(412) 268-2565
Notes, *a hypertext writer's tool (in development)*
ZOG, *a multiuser hypertext system (in development)*

MAD Intelligent Systems
55 Wheeler St.
Cambridge, MA 02138
(617) 492-1982
Developing hypertext through machine-generated links. Common Lisp software that runs on a Mac II and Unix machines. Prototype in use by the New York Stock Exchange.

MCC (Microelectronics and Computer Technology Corp.)
Software Technology Program
3500 West Balcones Center Dr.
Austin, TX 78759
(512) 343-0978
gIBIS, *a problem-analysis tool that runs on Sun workstations (in development)*
PlaneText, *a Unix-based, general-purpose system (in development)*

University of Maryland
Department of Computer Science
Human Computer Interaction Laboratory and Institute for Advanced Computer Studies
College Park, MD 20742
(301) 454-4255
Hyperties, *an instructional, interactive encyclopedia system (in development)*

University of North Carolina at Chapel Hill
Department of Computer Science
CB #3175
Sitterson Hall
Chapel Hill, NC 27599
(919) 962-1792
WE, *an interactive writing environment*

University of Southern California
Computer Science Department
Los Angeles, CA 90089
(213) 743-2311
DIF, *a hypertext system with software engineering tools (in development)*

The Xanadu Operating Co.
8480 Fredericksburg, Suite 138
San Antonio, TX 78229
(512) 927-6073
Xanadu, *a worldwide hypertext library (in prototype for Sun workstations)*

Xerox Palo Alto Research Center
Intelligent Systems Laboratory
3333 Coyote Hill Rd.
Palo Alto, CA 94304
(415) 494-4000
NoteCards, *an information analyst's support tool*

RECOMMENDED READING

Conklin, Jeff. "A Survey of Hypertext." *IEEE Computer*, September 1987.

Halasz, Frank. "Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems." *Communications of the ACM*, July 1988.

Hypermedia: The guide to interactive media production (premier issue from MIX Publications, 6400 Hollis St., #12, Emeryville, CA 94608, (415) 653-3307).

Hypertext '87 Conference Proceedings. University of North Carolina at Chapel Hill, Department of Computer Science (CB #3175, Sitterson Hall, Chapel Hill, NC 27599).

Nelson, Theodor H. "Managing Immense Storage." BYTE, January 1988.

Salton, G., and M. J. McGill. *Introduction to Modern Information Retrieval*. New York: McGraw-Hill, 1983.

CONFERENCE

HyperExpo Boston
World Trade Center
Boston, Massachusetts
October 15–16, 1988