A Survey of Hypertext

Jeffrey Conklin
MCC
Software Technology Program
P.O. Box 200195
Austin, Texas 78020
conklin@mcc.com
(512) 343 0978

(Converted from ACM's Hypertext on Hypertext HyperCard stack by paolo petta, April 1995)

Abstract

Hypertext is a computer-supported medium for information in which many interlinked documents are displayed with their links on a high-resolution computer screen. The links may be directly activated by a pointing device such as a mouse, which causes the document referenced by the link to appear instantly in a new window on the screen. While the concepts of hypertext are not new, the technology to make it effective is new. This paper reviews most of the existing hypertext systems, and then explores in some detail the fundamental features of hypertext and some of the design options in constructing hypertext systems. The advantages and disadvantages of hypertext are discussed in terms of four major application categories: macro literary systems, problem exploration systems, structured browsing systems, and systems developed to explore hypertext technology.

Preface

About one year ago I wrote the first version of this technical report. It was a modest paper, about 40 pages long, and it was incomplete and somewhat out of date in many ways. There was a great deal of interest in that TR, however, and many people wrote with suggestions about ways the paper could be extended and brought up to date. Incorporating those changes lead to Revision 1 in February, 1987. During this time more suggestions and news of new hypertext systems continued to challenge my filing system.

At the urging of several colleagues I sent Version 1 to Bruce Shriver at IEEE Computer , who replied quickly that he was very interested in publishing it if I could both make it at least 20% shorter and completely fill in the chart of the systems reviewed. While doing this I made several additions, polished quite a bit of the text, and spent many hours one memorable weekend on the phone with Andy van Dam finding out about a whole new twist on the history of hypertext. The version for Computer also got quite a bit of additional figures and photos. Bruce wanted to keep the bibliography of the Computer article short, so I agreed to limit myself to 20 or so references, but we agreed to offer the MCC TR, with its more complete bibliography, as a reader service option at the end of the article.

The response from the Computer article has been enormous. Requests for the “more detailed version” continue to come in (many of the , fortunately, from IEEE's computer with addresses already printed on sticky labels). Over 600 have been recieved to date. It was gratifying to see the impact of the article on the small but growing hypertext community at Hypertext '87 in Chapel Hill. Apple's introduction of HyperCard is going to add to the intellectual frenzy that is heating up about hypertext. New systems, and changes to old ones, are occuring at a pace that suggests that I could easily make this TR into a serial publication.

My problem is that my main research interest is still in the area of supporting design deliberation and capturing design rationale, topics which relate to hypertext only as it is and enabling technology. I find myself wishing someone else comes along soon who is more eager than I to be the “Mr. Know-it-all” of hypertext, so I can get back to work. Until then, I'm waiting for the signs that hypertext has made it to the “big time” — like being invited to be on the *Today* show, or at least *David Letterman* .

Jeff Conklin
Austin, Texas
December, 1987

1. Introduction

Most modern computer systems share a foundation which is built of *directories* containing *files* consisting of *text* composed of *characters* . For much of our current way of doing business this linear organization is sufficient. However, even traditional documents have parts that seek to break from the linear flow (e.g. footnotes), and there is a growing application for text that can have a richer structure. For example, the documentation of a computer program(1) is usually either squeezed into the margins of the program, in which case it is generally too terse to be useful, or its text is interleaved with the text of the program, which breaks up the flow of both program and documentation. In the development of large software systems, where there are many kinds of documents, all of which heavily reference each other, this problem is more severe.

However, as workstations(2) grow cheaper, more powerful, and more available it is becoming increasingly attractive to extend the traditional notion of "flat" text files organized hierarchically by allowing more complex organizations of the material. Mechanisms are provided which allow direct machine-supported references from one textual chunk to another; information is collected in smaller chunks because this is more consistent with such a referencing facility. These extensions fall under the general category of *hypertext* (also known as *nonlinear text*). Ted Nelson, one of the pioneers of hypertext, once defined it as "a combination of natural language text with the computer's capacity for interactive branching, or dynamic display … of a nonlinear text … which cannot be printed conveniently on a conventional page."

This paper is a survey of existing hypertext systems, their applications, and their design. It is meant to be both an introduction to the world of hypertext and, at a deeper cut, a survey of some of the most important design issues that go into fashioning a hypertext environment.

One of the most promising applications for hypertext is in structuring on-line access to the large number of documents currently involved in the system development process, e.g. requirements, specifications, designs, standards, policies, memos, notes, etc. Not only can hypertext provide more rapid and natural access to these documents, but it can smoothly integrate the official public documents with personal notes and memos. This report provides a background for several lines of research within the MCC Software Technology Program (STP) on uses of hypertext within Leonardo such as design document integration, capture of design rationale, and teamwork and meeting support.

1.1 What is Hypertext?

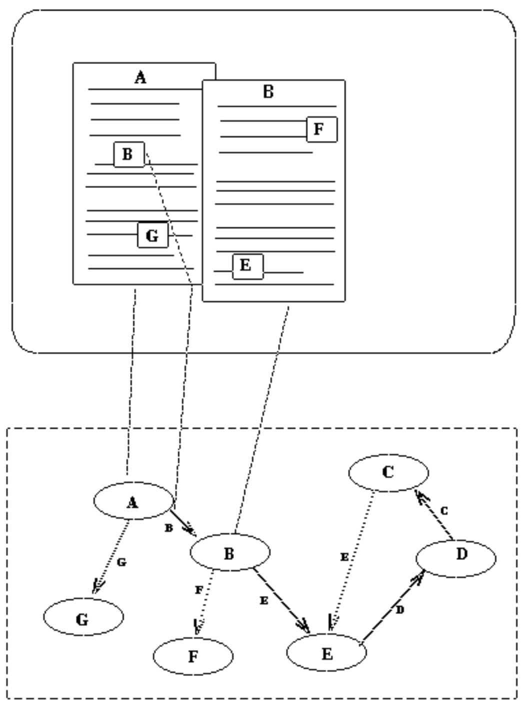
The concept of hypertext is quite simple: windows on the screen are associated with objects in a data base (see Figure 1), and links are provided between these objects, both graphically (i.e. as labelled icons) and in the data base (i.e. as pointers).

But this simple idea is creating much excitment. Several universities have created laboratories for research on hypertext, many articles have been written about the concept just within the last year, and the Smithsonian Institute has created a "National Demonstration Laboratory" to develop and display hypertext technologies. So what is all the fuss about? Why are some people willing to make such extravagant claims about "idea processing" and "a basis for global scientific literature"?

In this paper I will attempt to get at the essence of hypertext, and discuss its advantages (i.e. what problems it solves) and disadvantages (i.e. what problems it creates). It is my opinion that this technology opens some very exciting new possibilities, particularly for new uses of the computer as a communication and thinking tool. However, the reader who has not used hypertext should expect that at best it will come across as a collection of interesting features, just as a description of electronic spreadsheets will not get across the real elegance of that tool. In fact, one must live in a hypertext environment for a while for the collection of features to gel into a useful tool.

One problem with identifying the essential aspects of hypertext is that the term has been used quite loosely in the past 20 years for many different collections of features. Such tools as window systems, electronic mail, and teleconferencing share features with hypertext. Early hypertext systems emphasized three aspects: a database of nonlinear text, "view filters" which selected information from this database, and "views" which structured the display of this information for the terminal. The availability of workstations with high resolution displays has shifted the emphasis to more graphical depictions of nodes, links, and networks, such a using one window for each node. This paper focuses on *machine supported links* (i.e. rapid automatic link following both within and between documents) as the essential feature of hypertext systems, and treats other aspects as extensions to this basic concept.(3) It is this linking capability which allows a nonlinear organization of text. A heavy use of windows, and a one-to-one correspondence between windows and nodes in the database, is common to many hypertext systems, but we consider this feature to be of secondary importance.

One way to delimit what hypertext is is to point out what it is not . Briefly, there are several systems which have some of the attributes of hypertext but which do not qualify. One is window systems; while window systems do have some of the interface functionality, and therefore some of the "feel", of hypertext, window systems have no single underlying database, and therefore lack the database aspect of hypertext. Another candidate is file systems; one could claim that a file system is a database, and that one moves among "nodes" (files) by simply invoking an editor with their names. Hypertext demands a more sophisticated notion of links, and more machine support for them, than typing names to a text editor prompt. Similarly, most outline processors (such as "ThinkTank") have little or no support for references between outline entries, although their integrated hierarchical database and interface do approximate hypertext better than the other candidates mentioned. Text formatting systems such as Troff and Scribe allow a tree of text fragments in separate files to be gathered into one large document; however, this structure is hierarchical, and there is no interface for on-line navigation within this (essentially linear) document. Finally, database management systems (DBMSs) have "links" of various kinds, e.g. relational and object oriented, but again lack the emphasis on having a single coherent interface to the database which is the hallmark of hypertext.



Hypertext database
Figure 1

The correspondence betweeen windows and links in the display and nodes and links in the database. The link named "B" in window A has been activated by a pointing device, causing a new window named "B" to be created on the screen and filled with the text from node B in the database. (Most hypertext implementations allow links to have names that are different from the node they point to.)

As videodisc technology comes of age there is growing interest in the extension of hypertext to the more general concept of *hypermedia* , in which the elements which are networked together can be text, graphics, digitized speech, audio recordings, pictures, animation, film clips, and presumably tastes, odors, and tactile sensations. At this point little has been done to explore the design and engineering issues of these additional modalities, although many of the high-level design issues are likely to be shared with hypertext. Therefore this survey will primarily address the more conservative text-based systems.

1.2 A Glimpse of Using Hypertext

It is useful to have a sense of the central aspects of using a hypertext system, particularly if you have never seen one. Below is a list of such features for a somewhat idealized hypertext system. Some existing systems have more features than these, and some have fewer or different ones — the point here is to create a sense of using such a system, both for writing and reading.

- The database is a network (or graph) of textual (and perhaps graphical) nodes (which can be thought of as a kind of "hypergraph" or "hyperdocument".)
- Windows on the screen correspond to nodes in the data base on a one to one basis, and each has a name or title which is always displayed in the window. However, only a small number of nodes are ever "open" (as windows) on the screen at the same time.
- Standard window system operations are supported: windows may be repositioned, resized, closed, and put aside as small window icons. Both window and icon position and size (and perhaps also color and shape) are cues to remembering the contents of the window. Closing a window causes the window to disappear after any changes that have been made are saved to the database node. Clicking

with the mouse on the icon of a window that has been put aside causes it to open into its window instantly.

2. Hypertext Implementations

The history of hypertext is rich and varied, because hypertext is not so much a new idea as a natural and general use to which to put computers. As such there have been many contributors to the idea, and each of them seems to have had something different in mind. We gather most of them together for review in this section, in an effort to present some historical flow to the development of the ideas as well as to sketch some of the hypertext applications that have been thought of to date. We have not described the individual systems and ideas reviewed here in any detail — for that the reader is invited to consult the literature directly.

One sort of manual hypertext is the traditional use of 3x5 index cards for note taking. Note cards often reference each other, as well as being arranged hierarchically (e.g. by shoebox or rubber-banded bundles). One particular advantage of note cards is that their small size modularizes the notes into small chunks, making it easier to reorganize a set of cards when new information suggests a restructuring of the notes. Of course, one problem with note cards is that it can be difficult to find a specific card if there are many of them.

Another kind of manual hypertext is exemplified by dictionaries and encyclopedias, each of which can be viewed as a graph of textual nodes joined by referential links , and in that sense they are a very old form of hypertext. As one reads an article or definition, explicit references to related items indicate where to get more information about those items. True, the majority of people's “transactions” with a dictionary make use of the linear (alphabetic) ordering of its elements (i.e. definitions) for accessing a desired element. But an encyclopedia can be best used in the mode of exploring the local nodes in the “network”, once one has found the desired entry through alphabetic lookup. Indeed, many reference and technical documents — consider the microfiche parts diagrams used by your automobile mechanic — are rich with cross references.

There are even some ancient documents in which references to other parts of the document, or to other documents, constitute a major portion of the work. Both the Talmud, with its heavy use of annotations and nested commentary, and Aristotle's writings, with their reliance on references to other sources, are ancient candidates for hypertextual representation.

But if one insists, as most modern proponents of hypertext do, that navigation through hypertextual space must be computer supported in order to qualify as true hypertext, then the field is narrowed considerably, and the history likewise shortened.

In some ways the people who first described hypertext — Bush, Engelbart, Nelson — all had the same vision for hypertext as a path to ultimate human-computer interaction, a vision which is still alive today among hypertext researchers. Thus the historical review below stresses the early development of ideas about hypertext as much as the more contemporary implementation efforts.

Because of the difficulty of precisely classifying hypertext systems according to their features, our description here will list systems according to application. There are generally four ways in which hypertext systems have been used:

macro literary systems: the study of technologies to support large on-line libraries in which inter-document links are machine supported — all publishing, reading, collaboration, and criticism takes place within the network;

problem exploration tools: tools to support early unstructured thinking on a problem, in which many disconnected ideas come to mind, such as early authoring and outlining (“idea processors”), problem solving, and programming and design;

browsing systems: similar to macro literary systems, but smaller scale — systems for teaching, reference, and public information, where ease of use is crucial;

general hypertext technology: general purpose systems designed to allow experimentation with a range of hypertext applications — most commonly applied to reading, writing, collaboration, etc..

These categories are somewhat informal — often it is only the single application to which a system has been applied that determines which category it is described in. Bear in mind that some of the systems mentioned below are full-scale environments, while others are still only conceptual sketches. Some systems have focused more on the development of the “frontend” — the user interface aspects — while others have focused on the database issues of the "backend". (Various features of those systems described here which have been implemented are summarized in Figure8).

2.1 Macro Literary Systems

The earliest visions of hypertext all center on the integration of colossal volumes of information readily accessible via a simple and consistent interface. The whole network publishing system constitutes a dynamic corpus to be enriched by readers without defacing the original documents — thus the difference between authors and readers is diminished. The advent of the computer has brought this vision closer to reality, but it has also made it clear how monumental the problems (organizational and procedural as well as technical) are. This section reviews workers whose intent was to place whole libraries on line.

2.1.1 Bush's Memex

Vannevar Bush, President Roosevelt's Science Advisor, is credited with first describing hypertext in his 1945 article "As We May Think" [Bush45], in which he calls for a major post-war effort to mechanize the scientific literature system. In the article he introduces a machine for browsing and making notes in an extensive on-line text and graphics system. This "memex" contained a very large library as well as personal notes, photographs, and sketches, had several screens, and a facility for establishing a labelled link between any two points in the entire library. Although the article is remarkably foresightful, Bush did not anticipate the power of the digital computer, and thus his memex uses microfilm and photocells to do its magic. But Bush did anticipate the information explosion, and was motivated in developing his ideas by the need to support more natural forms of indexing and retrieval:

> *The human mind … operates by association. Man cannot hope fully to duplicate this mental process artificially, but he certainly ought to be able to learn from it. One cannot hope to equal the speed and flexibility with which the mind follows an associative trail, but it should be possible to beat the mind decisively in regard to the permanence and clarity of the items resurrected from storage.*
>
> [Bush45]

2.1.2 Engelbart's NLS/Augment

Less than two decades later Douglas Engelbart, at Stanford Research Institute, was influenced by Bush's ideas, and in 1963 wrote "A Conceptual Framework for the Augmentation of Man's Intellect" [Enge63]. Engelbart's vision was that computers might usher in a new stage of human evolution, characterized by "automated external symbol manipulation":

> *In this stage, the symbols with which the human represents the concepts he is manipulating can be arranged before his eyes, moved, stored, recalled, operated upon according to extremely complex rules — all in very rapid response to a minimum amount of information supplied by the human, by means of special cooperative technological devices. In the limit of what we might now imagine, this could be a computer, with which individuals could communicate rapidly and easily, coupled to a three-dimensional color display with which extremely sophisticated images could be constructed …*

His proposed system, H-LAM/T (Human using Language, Artifacts, and Methodology, in which he is Trained) included the human user as an essential element: the user and the computer were dynamically changing components in a symbiosis which had the effect of "amplifying" the native intelligence of the user. This is still a common vision among developers of hypertext systems.

Five years later, in 1968, Engelbart's ideas about augmentation had become more specific, and had been implemented as "NLS" (oN Line System) by the Augmented Human Intellect Research Center at SRI. NLS was developed to be an experimental tool on which the research group developed a system that would be adequate to all of their work needs, by

> *placing in computer store all of our specifications, plans, designs, programs, documentation, reports, memos, bibliography and reference notes, etc., and doing all of our scratch work, planning, designing, debugging, etc., and a good deal of our intercommunication, via the consoles.*
>
> [Enge68]

These consoles were very sophisticated by the standards of the day, and included television images and a variety of input devices, including one of Engelbart's best known inventions, the mouse.(6)

Files in NLS were structured into a hierarchy of segments(7) called "statements", each of which bore an identifier of its level within the file. For example, a document might have statements "1", "1a", "1a1", "1a2", "1b", etc., though these identifiers did not need to be displayed. Any number of *reference links* could be established between statements within and between files. Note that this is a structure which is primarily hierarchical, but which allows non-hierarchical links as well — the importance of supporting both kinds of structures is a point to which we return later.Several styles of traversal of the statements in files were provided by the system.

NLS provided for viewing filters for the file structure: one could clip the level (depth) of hierarchy displayed, truncate the number of items displayed at any level, and write customized filters (in a "high-level content analysis language") that displayed only statements having the specified content. NLS also introduced the concept of multiperson distributed conferencing/editing.

NLS has evolved over the years, is now called Augment (or NLS/Augment), and is marketed as a commercial network system by McDonnell-Douglas. In developing NLS the emphasis has been on creating a consistent environment for "knowledge workers" — i.e. office automation for software engineers. The system now includes many forms of computer-supported communication, both asynchronous (e.g. email with links to all documents, journaling of ideas and exchanges, bulletin boards) and synchronous (e.g. several terminals sharing the same display, teleconferencing). There are facilities for document production and control, organizational and project information management (e.g. a shared calendar subsystem), and software engineering.

2.1.3 Nelson's Xanadu Project

During Engelbart's development of Augment another hypertext visionary, Ted Nelson, was developing his own ideas about augmentation, but with an emphasis on creating a unified literary environment on a global scale. It was Nelson who coined the term "*hypertext* ", and his thinking and writing are the most extravagant of any of the early workers. He named his hypertext system Xanadu, after the "magic place of literary memory" in Samuel Taylor Coleridge's poem"Kubla Khan". In Xanadu, storage space is saved by making heavy use of links: only the original document and the changes made to it are saved. The system makes it easy to reconstruct previous versions of documents.

> *Under guiding ideas which are not technical but literary, we are implementing a system for storage and retrieval of linked and windowing text. The document , our fundamental unit, may have windows to any other documents. The evolving corpus is continually expandable without fundamental change. New links and windows may continually add new access pathways to old material. Fast proprietary algorithms render the extreme data fragmentation tolerable in the planned back-end service facility.*
>
> [Nels80]

2.1.4 Trigg's Textnet

Randall Trigg wrote the first and to date the only PhD. thesis on hypertext, wherein he describes his "Textnet" system as supporting "nonlinear text", in which documents are organized as "primitive pieces of text connected with typed links to form a network similar in many ways to a semantic net".

In the tradition of the field, Trigg's system is just a first step in the direction of his vision:

> *In our view, the logical and inevitable result [of the computer revolution] will be the transfer of all such [text handling] activities to the computer, transforming communication within the scientific community. All paper writing, critiquing, and refereeing will be performed online. Rather than having to track down little-known proceedings, journals or unpublished technical reports from distant universities, users will find them stored in one large distributed computerized national paper network. New papers will be written using the network, often collaborated on by multiple authors, and submitted to online electronic journals.*
>
> [Trig83].

Textnet implements two basic types of nodes: those which have textual content — "*chunk* s" — and those which (hierarchically) organize other nodes — "*toc* s" (for table of contents). Thus Textnet supports both hierarchical trees (via the toc nodes) and nonhierarchical graphs (via the typed links).

Trigg further proposes a specific taxonomy of link types for use by collaborators and critics in Textnet (see Table 1). The idea is that there are generally a specific set of types of comments, and that there is a link type for each comment. For example, there are "refutation" and "support" links, and, more specifically, there are

Normal link types

Citation	Generalization/Specification	Summarization/Detail
C-source	Abstraction/Example	Alternate-view
C-pioneer	Formalization/Application	Rewrite
C-credit		
C-leads	Argument	Simplification/Complication
C-epon	A-deduction	Explanation
	A-induction	
Background	A-analogy	Correction
Future	A-intuition	Update
Refutation	Solution	Continuation
Support		
Methodology		
Data		

Commentary link types

Comment	*Points*	*Data*
Critical	Pt-comment	D-comment
Supportive	Pt-trivial	D-inadequate
	Pt-unimportant	D-dubious
Environment	Pt-irrelevant	D-ignored
E-comment	Pt-redherring	D-irrelevant
E-misrepresent	Pt-contradict	D-inapplicable
E-vacuum	Pt-dubious	D-misinterpreted
E-ignored	Pt-counter	
E-Isupercede	Pt-inelegant	*Style*
E-Irefute	Pt-simplistic	S-comment
E-Isupport	Pt-arbitrary	S-boring
E-Irepeat	Pt-unmotivated	S-unimaginative
		S-incoherent
Problem Posing	*Arguments*	S-arrogant
P-comment	A-comment	S-rambling
P-trivial	A-invalid	S-awkward
P-unimportant	A-insufficient	
P-impossible	A-immaterial	
P-ill-posed	A-mislead	
P-solved	A-alternate	
P-ambition	A-strawman	

Table 1: Trigg's link types

links to say that a point is irrelevant (“Pt-irrelevant”), that data cited is inadequate (“D-inadequate”), or that the style is rambling (“S-rambling”). In all Trigg described over 80 link types, and argued that the disadvantage of a limited set of link types was outweighed by the possibility of specialized processing on the hyperdocument afforded by a definite and fixed set of primitives.

In addition, Textnet supports the definition of *paths* : ordered lists of nodes used to browse linear concatenations of text, and to dump such scans to hard-copy. The path facility relieves the hypertext reader from having to make an *n-way* decision at each link; rather, the reader is provided a default pathway through the network (or part of the network), and can simply read the material in the suggested order as if it were a linear document.

Trigg joined Xerox PARC after completing his thesis, and was one of the principal architects of the Xerox NoteCards system (see below).

2.2 Problem Exploration Systems

These are systems which are designed to be highly interactive and to provide rapid response to a small collection of specialized commands for the manipulation of information. They can be thought of as being steps toward electronic spreadsheets for text and symbolic processing. One important feature of most of these tools is a facility, a la Englebart's NLS, for suppressing detail at various levels specified by the user (e.g. the outline processors all have single keystroke commands for turning on and off the display of subsections of a section). This is an unusual but natural facility. One of the very things that hypertext and similar tools are best at is the collection of large amounts of relatively unstructured information. But such collections are of little use unless there are adequate mechanisms for filtering, organizing, and browsing. These are the primary desiderata of these authoring/thinking/programming systems.

2.2.1 Goldstein and Bobrow's PIE

In 1980 Ira Goldstein and Danny Bobrow wrote a collection of papers proposing a “Personal Information Environment” (*PIE*), primarily for use in software development [Gold80, Gold87]. Their concern was to help the software designer with the various views, or “perspectives”, that he could have on the evolving system: the performance view, the reliability view, the management view, etc. The software system is represented as a hypertext network of code, specifications, documentation, etc. Nodes in the network have multiple perspectives: “Each perspective describes a different aspect of the program structure represented by the node, and provides specialized actions from that point of view.”

Nodes are organized into layers of the system representation, but nodes also have “contexts”: contexts and layers are used to represent alternative designs. PIE also has the notion of contracts: demons which monitor the changes in nodes and their connections to assure that constraints are kept. Constraints can be created directly by the designer.

2.2.2 Issue Based Information Systems

Horst Rittel and his students have introduced the notion of *Issue Based Information Systems* (IBIS) [Ritt73] to handle systems analysis in the face of “wicked problems”. Rittel describes wicked problems (as opposed to “tame” ones) as being those for which the traditional systems analysis approach (i.e. (1) define the problem, (2) collect data, (3) analyze the data, (4) construct a solution) is inadequate. Wicked problems lack a definitive formulation, and the problem space cannot be mapped out without understanding the solution elements -- in short, the only way to really understand the problem is to solve it. Wicked problems have no stopping rule -- the design or planning activity stops for considerations that are external to the problem, e.g. time, money, or patience. Solutions to wicked problems are not Right or Wrong, they just have degrees of sufficiency. Rittel argues that solving wicked problems calls for a process that allows the many viewpoints, ideas, values, concerns, etc. of all those involved to be exchanged and argued, so that what emerges is a common understanding of the major issues and their implications (i.e. to understand another's viewpoint is to understand the whole problem better). For this process he proposes IBIS's to support this design/planning conversation.

IBIS systems can thus be thought of as a marriage of news and teleconferencing systems, in which many people can participate in one conversation, and hypertext, which allows participants to move easily between different issues and different threads of argument of the same issue. The current version of Rittel's IBIS runs on an Apple PC, and is being ported to Sun workstations. It has three basic types of nodes (Issues, Positions, and Arguments) and nine types of relations (e.g. Questions, Supports, Contradicts, etc.) that are used to link nodes. The basic idea is that someone posts an Issue, then that person or others post Positions about that Issue, and then the positions are argued using Argument nodes. Of course, any of these three types of nodes can be the seed of a new Issue (See Figure 3). The current set of relationships between nodes is: Responds-to, Questions, Supports, Objects-to, Specifies, Generalizes, Refers-to, and Replaces (which connects a new version of an issue to the previous version).

In STP (MCC's Software Technology Program) we have developed a graphical tool to support concurrent teamwork that uses IBIS as a design method. The tool, called gIBIS (for graphical IBIS), runs on color workstations and uses an unusual segmenation of the screen to separate the browser from the node contents (see Figure4)(9).

The gIBIS browser uses iconic shapes and color to clearly indicate type and state information for nodes and links. It displays the issue networks from two tightly coupled vantage points: a global (or zoomed-out) view from which users can view the entire scope of the network, and a local (or zoomed-in) view which reveals the fine structure of the network.

The gIBIS interface provides context-sensitive menus which constrain the users to making only "legal" methodological moves, thereby ensuring the taxonomic integrity of the networks.

Users can instantaneously access any node in the network by directly mousing it in the browser, by selecting it through the hierarchically-ordered index window, or by use of the NEXT button, which leads users through the network in a structure-based, linearized fashion.

An integral search and query mechanism allows users to rapidly search through issue networks by constructing a proto-node whose structure and content mirrors that of the nodes they wish to retrieve (i.e. a "query by example" approach).

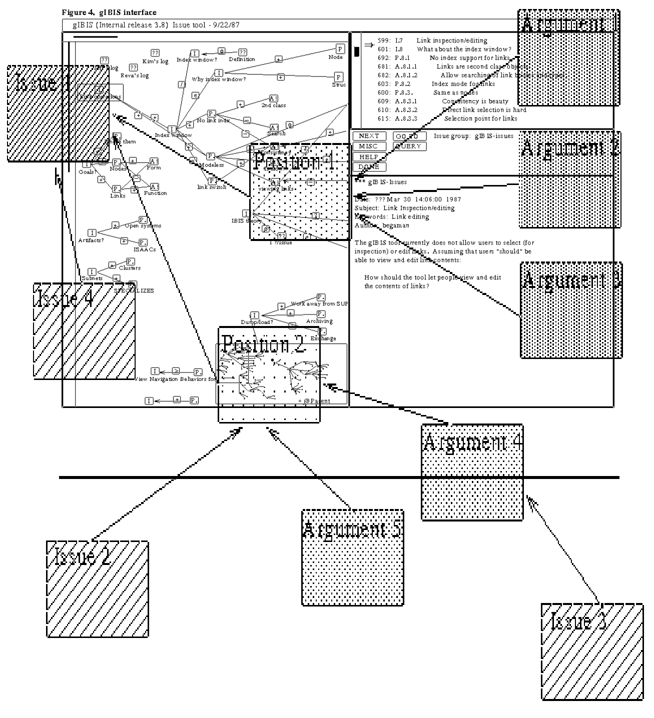


Figure 3

A segment of a possible IBIS-style discussion showing the topology of the IBIS network. Each node contains information on the type of the node, the time and date of creation, the author, a short phrase describing the content, a longer body of text with the text of the comment, a list of keywords, and a list of the incoming and outgoing links.

The tool supports simultaneous access and update of issue groups by multiple users on a common Local Area Network. gIBIS provides the necessary concurrency control, locking, and update notification to allow real-time interactive network construction by teams of cooperating designers.

Experience to date suggests that the interface is very easy to learn and synergizes well with the IBIS method. It is not clear, however, how useful color will be as a distinguishing feature in the browser as the number of node types in the system, such as requirements, goals, and artifacts, proliferates.

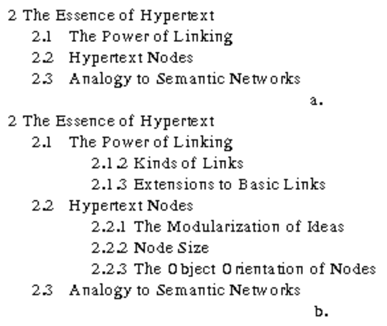
2.2.3 Lowe's SYNVIEW

David Lowe's SYNVIEW system is similar in concept to Rittel's IBIS but goes in a different direction to propose that the participants, in addition to posting their own issues and arguments, assess previous postings as to their validity and relevance to their predecessor postings. The assessment is done by a kind of quantitative voting, e.g. if you think that Joe's response to Sam makes a good point but is not really a direct response to Sam's posting, you might grade it, say, "5,1" (where 5 is a high validity rating and 1 is a low relevance rating). These values will be averaged into the existing values for that posting. The various displays of the argument structure show the values for each posting, allowing readers to focus, if they choose to, on those argument trails having the highest voted validity.

> *Through debates on the accuracy of information and on aspects of the structures themselves, a large number of users can cooperatively rank all available items of information in terms of significance and relevance to each topic. Individual users can then choose the depth to which they wish to examine these structures for the purposes at hand. The function of this debate is not to arrive at specific conclusions, but rather to collect and order the best available evidence on each topic.*
>
> [Lowe85]

2.2.4 UNC's WE

A group at the University of North Carolina at Chapel Hill has been developing a *writing environment* called WE [Smit86c]. Their research starts with a cognitive model of the communication process in which reading is viewed as taking the *linear* stream of text, comprehending it by structuring the concepts *hierarchically* , and absorbing it into long term memory as a *network* . Writing is seen as the reverse process: a loosely structured network of internal ideas and external sources is first organized into an appropriate hierarchy (an outline) which is then "encoded" into a linear stream of words, sentences, etc.



```
2 The Essence of Hypertext
   2.1  The Power of Linking
   2.2  Hypertext Nodes
   2.3  Analogy to Semantic Networks
                                        a.
2 The Essence of Hypertext
   2.1  The Power of Linking
            2.1.2 Kinds of Links
            2.1.3 Extensions to Basic Links
   2.2  Hypertext Nodes
            2.2.1 The Modularization of Ideas
            2.2.2 Node Size
            2.2.3 The Object Orientation of Nodes
   2.3  Analogy to Semantic Networks
                                        b.
```

Table 2: Alternate views of the same outline. Part a. has the third level of detail suppressed. Part b. shows the third level exposed.

WE is designed to support the upstream part of writing — it contains two major view windows, one graphical and one hierarchical, and many specialized commands for moving and structuring material (i.e., nodes and links with attached text) between these two views. Normally a writer will begin by creating nodes in the graph view, where they can be placed anywhere within the window. At this stage, nodes can be placed in "piles" if they seem to be related, or placed between two piles if they are somewhat related to both — there is little or no structure imposed on the conceptual material. As some conceptual structure begins to emerge from this process nodes can be copied into the hierarchy window, in which the commands are specialized for tree operations.

There are four different display modes in the hierarchy window: (1) the tree can be laid out on its side, with the root node on the left; (2) the tree can be hung vertically with the root at the top; (3) child nodes can be displayed *inside* their parent node — called the "Chinese Box" display; and (4) the hierarchy can be shown in the traditional outline view.

A third window is an editor for the material within the "current node", which is selected by pointing with the mouse. WE uses a relational database for the storage of the nodes and links in the network, and a fourth window on the screen is for queries to this database. A fifth "control panel" window is used to control system modes and the current working set of nodes.

WE is designed to be an experimental platform to study what tools and facilities will be useful in a writer's environment, so the real validation of these ideas, as with so many of the systems described here, will come with further experiments and analysis.

2.2.5 Outline Processors

An outline processor is a word processing program which is specialized for processing outlines, in that its main commands deal with movement among, creation of, and modification of *outline entries* . In this respect these programs commercialize many ideas from Engelbart's NLS/Augment. Outline processors also have at least simple text editors and do some text formatting as well, so that it is possible to go from outline to finished document within the one tool. One of the most powerful features of outline processing is the suppression of lower levels of detail in the outline. As with Engelbart's NLS/Augment, one can view just the top level of the outline, or the top n levels, or can walk the tree opening up just those entries that are relevant or useful to the idea being worked on (see Table 2).

In addition, each outline entry may have a textual "body" of any length associated with it, and this body can be made to appear or disappear with a single keystroke. It turns out that this is a real boon to the writing process, because it allows the writer to have a view of both the immediate text being composed and the global context for it. It also facilitates rapid movement between sections, particularly in large documents, because in outline mode a "distant" section is never more than a few keystrokes away.

Most outline processors are personal computer programs,(10) and they have done much to bring some of the concepts underlying hypertext into popularity. The first of these was called *ThinkTank* [Hers84], and was released in 1984. It has since been joined by a host of others, with names like *MaxThink* , *Executive Writer/Executive* Filer , *Thor* , *Framework* , *Kamas* , *Fact Cruncher* , *Freestyle* , *Idea!* , and *PC-Outline* (see

[Hers85, Fost85, Spez86, Lewi86, and Wood86). There are two very recent additions to the field: *Houdini* is an extension of MaxThink that supports rich non-hierarchical internode references; and *ForComment* is a word processor that allows up to 15 people to apply hypertext-like annotations to a document in preparation (and can operate over a Local Area Network in real time).

Aside from Houdini most outline processors do not support inter-entry references, except by “cloning” the whole entry and displaying it in the new location. Only a few others provide windows for nodes, and none of them provide explicit “mouseable” link icons, so it is problematic whether they qualify as hypertext as I have defined it here. But it was *ThinkTank* that first coined itself — somewhat pretentiously — to be an *idea processor* , and all of these programs do have the property, shared with hypertext, that they respect *segments of text* as first class objects, and support manipulations that coincide with the way one manages ideas, e.g. hierarchicalization, restructuring (i.e. of the hierarchy), and rapid foreground/background (text/outline) shifting. Thus, they foreshadow the inevitable proliferation of hypertext features into the mainstream of computer applications.

2.3 Structured Browsing Systems

The systems reviewed in this section were designed primarily for applications in which either there is a large amount of existing information or it is very important that the information be very easy to access, even by beginners. These systems pose different problems for their designers: ease of learning and ease of use are paramount, and great care goes into crafting the interface. On the other hand, in these systems writing (adding new information) is usually either not allowed to the casual user or is not particularly well supported.

2.3.1 CMU's ZOG and Knowledge Systems' KMS

ZOG is a menu-based display system developed in 1972 at Carnegie-Mellon University [Aksc84b]. It consists of a potentially large database of small (screen-sized) segments which are viewed one at a time. ZOG was developed with the particular goal of serving a large simultaneous user community, and thus was designed to operate on standard terminals on a large time sharing system. In 1981 two of the principals on the ZOG Project, Donald McCracken and Robert Akscyn, started the company Knowledge Systems and developed a commercial successor to ZOG called KMS (for Knowledge Management System)(11).

Each segment of the ZOG/KMS database, called a *frame* , has by convention a one-line title at the top of the screen which includes a unique name for the frame, a few lines of text below the title stating the issue or topic of the frame, a set of numbered (or lettered) menu items of text called *selections* , and a line of standard ZOG commands called *global pads* at the bottom of the screen. (Some of these commands, for example, are: edit, help, back, next, mark, return, and comment.) The selections interconnect the frames. When an item is selected (by typing its number or letter at the terminal keyboard), the frame so indicated is displayed on the screen, replacing the previous frame. The structure is generally hierarchical, though cross-referencing links can be included. In addition, an item in a frame can be used to activate a process.

In 1982 ZOG was installed and used as a computer-based information management system on the nuclear-powered aircraft carrier USS CARL VINSON, and is probably the largest and most thoroughly tested hypertext system in service in the field. It has also been used for more interactive process applications such as policy analysis, authoring, as a communication system, and for code management. Historically, however, ZOG made its name more as a bulletin board/textual database/CAI tool than as an interactive system, and hence is included in this section on Browsing.

It would seem that a drawback of the ZOG/KMS style of viewing a single frame at a time is that users may become more easily disoriented, since there is no spatial event corresponding to moving from frame to frame. In the KMS system this tendency has been offset by minimizing system response time, so that frame to frame transition takes about half a second. Any tendency to promote user disorientation is greatly reduced by the fact that the user can move very quickly among frames and thus become reoriented with very little effort. Whether or not response speed obviates the need for a graphical or higher level view of the hyperdocument probably depends on the linkage structure of the document, i.e. the degree of hierarchy, and the goals of the users.

2.3.2 Emacs INFO Subsystem

The help system in Emacs, called INFO [Stal81], is much like ZOG. It has a simpler set of standard commands, and its control input is done by single letters or short commands typed at the keyboard. It is primarily hierarchical, but a user can jump to a different place in the hierarchy by typing in the name of the destination node. It is used as an on-line help system in EMACS. INFO has the same tendency to allow user disorientation which is shared by all of the systems which only display a single frame at a time.

2.3.3 Shneiderman's Hyperties

At the University of Maryland Ben Shneiderman has headed up a project called *Hyperties* , a Hypertext system based on the Interactive Encyclopedia System. Hyperties is being developed in two directions: as a practical and easy to learn tool for browsing in instructional databases (e.g. in a museum exhibit about Austria and the Holocaust), and as experimental platform for studies on the design of hypertext interfaces. As a hypertext system which has already seen some use “in the field” (e.g. at a Washington, D.C. museum exhibit), the emphasis is on making the system easy and fun for users who have never used a computer before. As an experimental platform, it has been used in five experimental studies involving over 220 subjects [Shne86].

In Hyperties the basic units are short articles (50-1000 words typically), which are interconnected by any number of links. Links are highlighted words or phrases in the article text and are activated by touching them with a finger (on a touch-sensitive screen) or using the arrow keys to jump to them.(12) Activating a link causes the article about that topic to be brought up in its own window on the screen. The system keeps track of the user's path through the network of articles, allowing easy return from exploratory side paths.

In addition to a title and a body of text, each article has a short (5 to 25 word) description of the article which can be displayed very quickly. This allows the user an intermediate position between bringing up the full article and trying to guess from the link name precisely what the article is about.

Hyperties runs on the IBM PC(13). Recently graphics capabilities have been added to the system. Current implementation efforts focus on support for videodisc images, and a browser is being developed which will provide string search, bookmarks, multiple windows, user annotation, etc.

2.3.4 Symbolics Document Examiner

The most advanced of the on-line help systems, this tool displays the pages from the entire (twelve volume) manual set on the Symbolics Lisp machine screen [Walk85]. Certain textual fields in the document (printed in bold) are mouse sensitive, and when moused cause the relevant section of the manual to be added to the current working set of manual pages. The system allows the reader to place “bookmarks” on any topic and to move swiftly between bookmarked topics. The protocol for link following is tailored to browsing in a reference manual or encyclopedia — mousing a link only causes it to be placed on a list of current topics. Then, mousing an entry in this list causes that link to followed, bringing up the referenced topic in the main viewing window.

The system also supports on-line string search of pre-identified keywords, and includes search for whole-words, leading substrings, and embedded substrings. The system is thus well-designed for the specific task of browsing through a technical manual and pursuing several aspects of a technical question and/or several levels of detail simultaneously. The user cannot make any changes or additions to the manual set (although it is possible to save personalized collections of bookmarks).

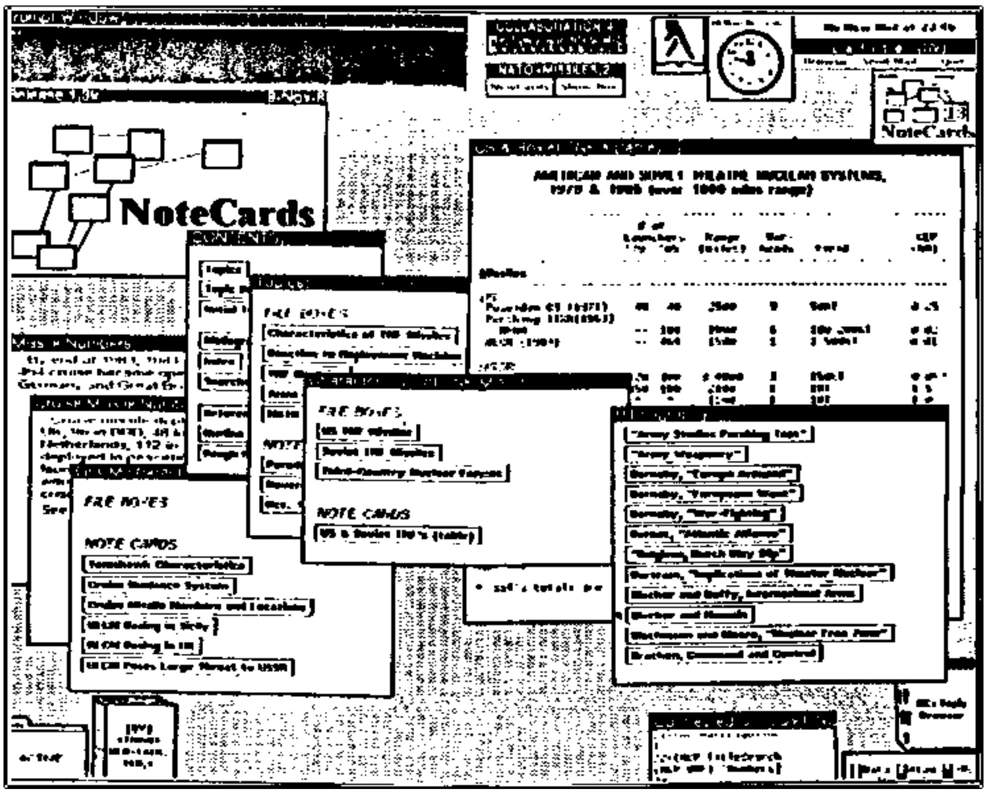
2.4 General Hypertext Technology

The previous sections have reviewed hypertext systems according to their application class. The following systems all have been applied to at least one application, but their primary purpose is experimentation with hypertext itself as a technology. For example, while NoteCards has been used for authoring, programming, personal information management, project management, legal research, engineering design, and CAI, its developers view it primarily as a research vehicle for the study of hypertext.

2.4.1 Xerox PARC's NoteCards

Perhaps the best known version of full hypertext is the *NoteCards* system developed at Xerox PARC by Frank Halasz(14), Tom Moran, and Randy Trigg [Hala87a]. The original motivation in building NoteCards was to explore the development of an information analyst's support tool, one that would help gather information about a topic and produce analytic reports at regular intervals and on demand. It was observed that the general procedure that such analysts used was (a) reading sources (news reports, scholarly articles, etc.), (b) collecting clippings and filing them (in actual shoe boxes!), and (c) writing analytic reports. It was

Figure 5: A sample of the NoteCards screen showing NoteCards and FileBoxes.



also observed that throughout the process, analysts are forming *analyses* and *conceptual models* in their heads. The research goal of the PARC team was to develop technology to aid the analyst in forming better conceptual models and analyses, and to find better expressions of these models and analyses.

A "programmer's interface" makes NoteCards an open architecture that allows users to build (in Lisp) new applications on top of NoteCards, including making it easy to customize the browser. NoteCards allows easy creation of new types of nodes. Forty or fifty such specialized node types have been created to date, including text, video, animation, graphics, and actions(15). The new version also allows several users to work in the same *Notefile* at the same time [Trig86b].

Part of NoteCards' success is due to the fact that it was developed and used on Xerox D series Lisp machines, which are powerful enough workstations that they allow rapid creation and browsing of many hypertext nodes, as well as having a high resolution screen that allows windows and link and node icons to be displayed in excellent resolution (see Figure 5). Another part of its success could be credited to the imaginative contributions of such people as John Seely Brown [Brow82]. There are currently between 50 and 100 users of NoteCards, many of them outside of Xerox, even though it is not a supported product, and several of these users have constructed very large databases in the system (e.g. 1600 nodes with 3500 links between them). (Unfortunately, the Xerox D machines are quite expensive, and they are the only delivery vehicle for NoteCards at this time.)(16)

2.4.2 Brown University's Intermedia

One of the largest hypertext development efforts to date is the Intermedia project of the Institute for Research in Information and Scholarship (IRIS) at Brown University [Yank85, Garr86a]. The Intermedia project builds on two decades of work and three prior generations of hypertext systems [Yank85]. The first was the Hypertext Editing System designed by Ted Nelson, Andy van Dam, and several Brown students for the IBM 2250 display in 1968, and was used by the Houston Manned Spacecraft Center to produce Apollo documentation. FRESS, the File Retrieval and Editing System, was a greatly enhanced multiterminal timesharing version designed by Andy van Dam and his students that became available in 1969 and was commercially reimplemented by Phillips in the early 1970's. FRESS was used in production by hundreds of faculty and students over more than a decade including an English poetry class that did all of its reading and writing on a communal hypertext document. Like NLS, FRESS featured both dynamic heirarchy and bi-directional reference links, keyworded links and nodes, and, unlike NLS, no limits to sizes of nodes. On graphics terminals multiple windows and vector graphics were supported. The third project, the Electronic Document system, was a hypermedia system emphasizing color raster graphics and navigation aids.

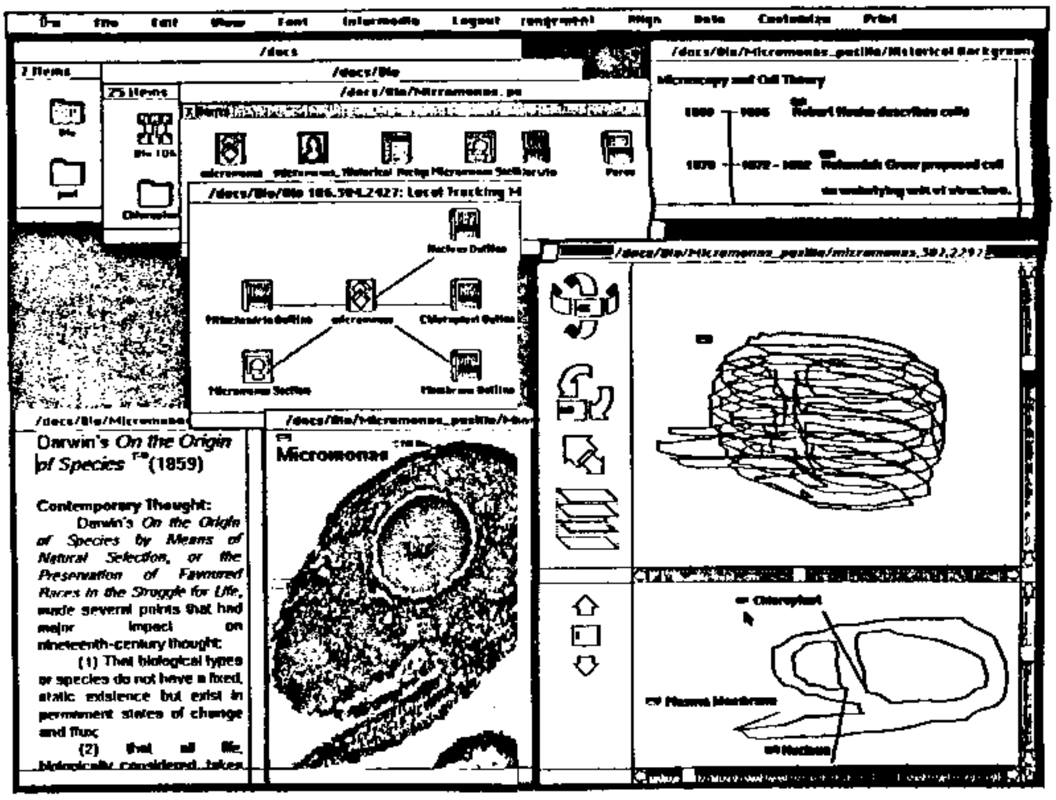
As part of Brown's overall effort to bring graphics-based workstations into effective use within the classroom, the Intermedia system is being developed as a framework for a collection of tools that allow authors to create links to documents of various media such as text, timelines, diagrams and other computer-generated images, video documentaries, and music. Two courses, one on cell biology and one on English literature, have been taught using the system. Current applications include: InterText, a text processor; InterDraw, a graphics editor; InterVal, a timeline editor that allows users to interactively organize information in time and date sequences; InterSpec, a viewer for sections of 3-D objects; and InterPix, a scanned-image viewer (see Figure 6).

Under development are a video editor, a 2-D animation editor, more complex methods for filtering the corpus and for creating and traversing trails.

Intermedia is being developed both as a tool for professors to organize and present their lesson material via computer and as an interactive medium for students to study the materials and to add their own annotations and reports.

> *For example, in the English literature course the first time a student is searching for background information on Alexander Pope, he or she may be interested in Pope's life and the political events that prompted his satiric criticism. To pursue this line of thought the student might retrieve the biography of Pope and a time line summarizing political events taking place in England during Pope's life. Subsequently, the student may want to compare Pope's use of satire with other later authors' satiric techniques. This time the student may look at the same information about Pope but juxtapose it with information about other satiricists instead of a time line. The instructor (and other students, if permitted) could read the student's paper, examine the reference material, and add personal annotation links such as comments, criticism, and suggestions for revision. While revising the document, the student could see all of the instructor's comments and examine the sources containing the counter-arguments.*
>
> [van Dam, personal communication]

Figure 6: A snapshot of the Intermedia screen


/docs
/docs/Bio
/docs/Bio/Micromonas_pusilla/Historical Background
Microscopy and Cell Theory
Micromonas
Chloroplast
Nucleus

Darwin's *On the Origin of Species* (1859)

Contemporary Thought:
Darwin's *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*, made several points that had major impact on nineteenth-century thought:
(1) That biological types or species do not have a fixed, static existence but exist in permanent states of change and flux;
(2) that all life, biologically considered, takes

Like most of the serious workers on hypertext, the Intermedia team is especially concerned with providing the user with ways of managing the increased complexity of the hypertext environment. For example, they propose that multiple links emanating from the same point in a document may confuse the reader, and that it may sometimes be better to have a single link icon in the material (text or graphics) which can be quickly queried via the mouse to show the specific outgoing links, their names, and their destination nodes [Garr86a]. They also propose a construct called webs to implement context dependent link display: every link belongs to one or more webs, and is only visible when one of those webs is active. To view documents with the links that belong to a particular web, a user opens a web and then opens one or more of its documents. Although other webs may also reference the document, only the links which were made in the current web are displayed. As a result the user does not have to sift through the connections made in many different contexts.

Another matter of study in the Intermedia project is into ways of providing an effective browser for a network that may include hundreds or even thousands of nodes. The Intermedia browser has two kinds of displays: a global map, which shows the entire hyperdocument and allows navigation within it, and a local map, which presents a view centered on a single document and displaying its links and nearest neighbors in the web. In addition, there are several levels of detail at which nodes and links can be displayed, from showing only whole documents and the links between them to showing each link and its approximate location within its documents.

The Intermedia project has many people working on it and a serious institutional commitment to its long term development. While it is still in the early stages, we can expect the combination of creative hypertext experiments and the classroom as a proving ground to contribute directly to the development of effective cooperative work environments based on hypertext (17).

2.4.3 Tektronix Neptune

There is one hypertext system that has been particularly designed as an open, layered architecture, the Tektronix Neptune system [Deli86b]. The heart of Neptune is, at the bottom level, a transaction-based server called the Hypertext Abstract Machine (HAM). Neptune consists of the HAM, written in C and running on a

UNIX server, and a graphical interface layer, written in Smalltalk-80. The HAM is a generic hypertext model which provides operations for creating, modifying, and accessing nodes and links. It maintains a complete version history of each node in the hypergraph, and provides rapid access to any version of a hypergraph. It provides distributed access over a computer network, synchronization for multi-user access, and transaction-based crash recovery.

The interface layer provides several browsers: a *graph browser* provides a pictorial view of a sub-graph of nodes and links; a *document browser* supports the browsing of hierarchical structures of nodes and links; and a *node browser* accesses an individual node in a hyperdocument. (Other browsers include attribute browsers, version browsers, node differences browsers, and demon browsers.)

Each end of a link has an offset within its node, whether that node is textual or graphical.(18) The link attachment may refer to a particular version of a node, or it may always refer to the current version. The HAM also provides two mechanisms that are useful for building application layers: it provides that nodes and links may have an unlimited number of attribute/value pairs, and special high-speed predicates are included for querying the values of these pairs in the entire hyperdocument, allowing higher level applications to define their own accessing mechanisms on the graph. Also, a demon mechanism is provided that invokes arbitrary code when a specific HAM event occurs.

Neptune, Notecards, and Intermedia appear to be the best examples of large systems designed to deliver hypertext database and interface capabilities in an open, generalized way. Apple's Hypercard also appears to be quite general, although it still has some significant limitations compared its conceptual ancestors.

2.4.4 diSessa's Boxer

Andrea diSessa has developed a highly interactive programming language called Boxer [diSe85, diSe86] which is specifically tailored to be easy for non-computer specialists to learn. Boxer uses the notion of a box as the fundamental unit of information in the system, containing other boxes or data such as text or graphics. For example, a program is a box containing boxes for its input and output variables, and other boxes which specify behavior. The system supports alternate views of some boxes: a box which specifies a graphics routine may also be viewed as a box which displays that routine.

Since Boxer is a programming language, the use of non-hierarchical links is not so important as in the other kinds of applications described here. However, Boxer does provide a kind of cross-reference facility called a port which is simply a view of a box at some other place in the system. This is in some ways more powerful than a simple link since the referenced box appears at the point it is referenced; presumably, a box can be changed through any of its ports, and all ports reflect such changes.

Hierarchy is more naturally expressed in Boxer. Boxes are nested within each other two-dimensionally, with filtering used to manage the level of clutter on the screen. This has the advantage of showing a hierarchy of nodes with a natural graphics: the windows of lower-level nodes are nested directly within their parents. In most hypertext systems that support hierarchical structures, no attempt is made to display the parent-child relationship once the nodes are opened as windows.

2.4.5 Pitman's CREF

CREF, the Cross Referenced Editing Facility, was developed by Kent Pitman in 1984. CREF was a prototype of a specialized text and graphics editor which was developed originally as a tool for use in analyzing the transcripts from psychological experiments (known as “protocols”), but along the way Pitman investigated some interesting and subtle aspects of hypertext implementation [Pitm85].

Much of the interactive feel of CREF reflects the style of use and programming of the Lisp Machine, on which it was built. Chunks of text, called segments , constituted the nodes in the system. Segments were arranged in linear series, and could have keywords and various kinds of links to other segments. The notion of a linear set of segments is natural to the protocol analysis problem, since the first step with such protocols is to segment them into the “episodes” of the experimental session.

CREF organizes segments into *collections* , which can either be defined implicitly by a predicate (called “abstract collections”) or by an explicit list (called “static collections”). At any time the “selected collection” appears as a continuous length of text with the segment boundaries marked by named horizontal lines (e.g. “Segment 1”, “Segment 2”); this view can be edited as if it were a single document.

One way of forming an abstract collection is by selecting segments using a boolean predicate over keywords. To extend the power of this keyword facility, CREF allows the user to specify a type hierarchy on the keywords. For example, if “card 105” is defined as a type of (i.e. a child of) “card”, then collections based on the keyword “card” will also contain segments which have only “card 105” as a keyword.

CREF supports four kinds of links: REFERENCES links are for general cross referencing among segments; SUMMARIZES links impose hierarchy — a “summary” is a segment which has one or more summarizes links to other segments; SUPERSEDES links implement versioning by copying the superceded segment and freezing it; and PRECEDES links place a linear ordering on segments. (This last link type seems to have been incompletely thought out at the time of the technical report.)

Finally, CREF allows multiple analysts to compose different “theories” about a protocol, using the same segmented data. Each “theory” imposes its own structure on the data, and has its own collections, diagrams, keywords, etc. This is similar to the notion of “contexts” or “webs” used in other systems (e.g. [Deli86b]).

2.4.6 Hypertext on the Apple MacIntosh

Three programs have been written for the Apple MacIntosh that provide hypertext facilities: FileVision, Guide, and Hypercard. FileVision is primarily oriented to graphics nodes. The advertising describes applications in which indexing is best done visually, for example, a database for a travel agency in which a map of a region contains icons for the main cities in that region. Clicking on the icon for a city in the display brings up a window containing a map of that city, with icons for the major landmarks in the city. Clicking on any of these icons brings up data about the landmark, or perhaps even a picture.

Guide is a more recent development [OWLI87]. It does not provide the graphics capabilities of FileVision (graphics are supported but cannot contain links), but Guide does support textual hypertext data very well. There are three kinds of links in Guide: *replacement* links, which cause the text in the current window to be completely replaced by the text pointed to by the link; *note* links, which display the destination text in a pop-up window (which remains only as long as the mouse button is depressed); and *reference* links, which follow traditional hypertext in bringing up a new window with the destination text.

The newest and most exciting of the MacIntosh systems is Apple's own *HyperCard* (tm) [Good87]. Developed by Bill Atkinson (the father of MacPaint), HyperCard is being sold for a very low price (about $49) and is being bundled for free with all new Apple Macintosh Plus' (tm), SE's (tm), and II's (tm).

Bill Atkinson admits freely that he wanted to build, not hypertext, but a “software erector set” that allowed non-programmers to easily construct sophisticated interfaces. HyperCard is lacking several features that would qualify it as hypertext in the fullest sense, including textual links (i.e. links which can be embedded in text and become a part of the text), links which can be traversed in both directions, and a graphical browser. HyperCard allows only one “card” to be presented at a time, the material in cards is not scrollable, and the links (called “ *buttons* ”) occupy fixed locations in their cards.

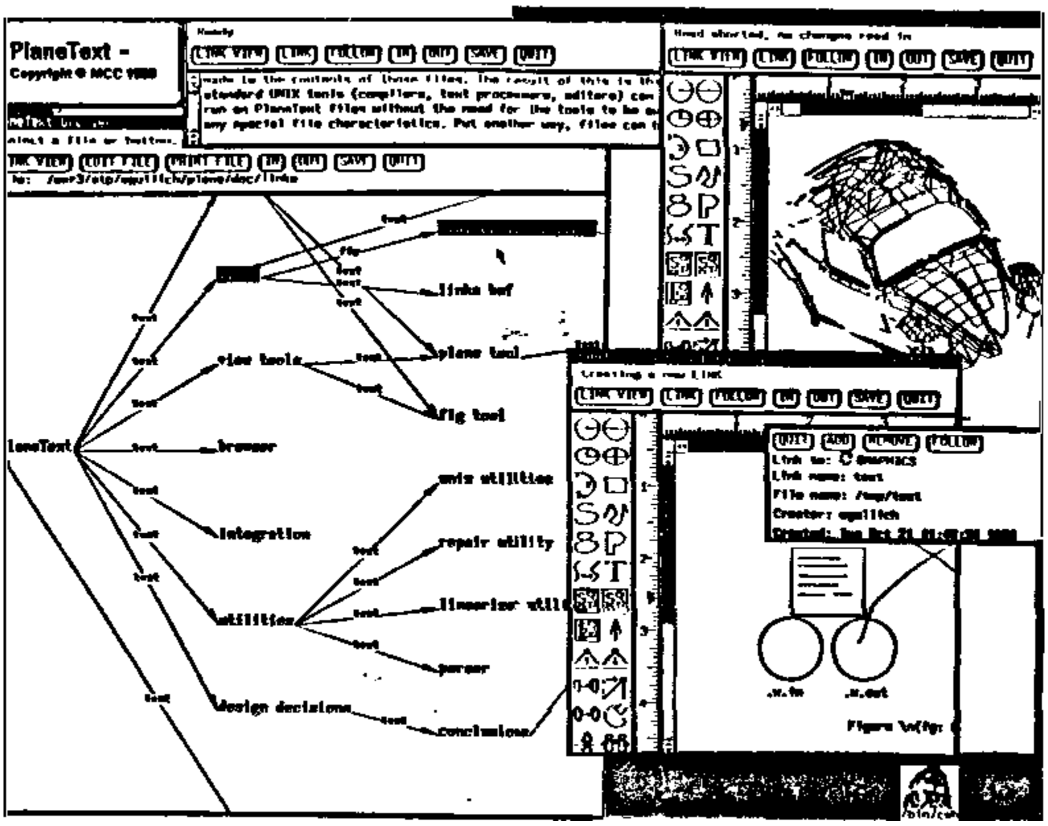
On the other hand, links in Hypercard are very powerful. A “button” on the screen is connected to a script, written in the HyperTalk programming language, that is activated by a mouse click in the button region. HyperTalk is a Smalltalk-like object-oriented language; objects such as cards and buttons are defined in a type hierarchy, and procedure invocation is by message passing. In the terminology of this report, HyperCard does not so much have links as it has procedural attachment, and one of the actions of a procedure can be as simple as following a link reference and bringing the destination card onto the display.

The HyperCard node is a “card”. Cards live in “stacks”, and in fact there is a standard hierarchy of objects, ranging from HyperCard at the top down through home, stack, card, field, and button. However, Hypercard does not have a general mechanism for hierarchically structuring cards — cards are grouped into stacks, but stacks cannot be grouped into stacks, so the “hierarchy” can only be one level deep. The system presents cards one at a time, and cards contain fields and buttons which define actions.

Cards are somewhat more graphically than textually oriented — the buttons in a card are layed out in fixed locations on the plane of the card, and text, while supported, is treated as a kind of graphical object. Thus cards have a certain layout, and there is no text formatting in the sense of links which are embedded in and afixed to a point in the text (“sticky links”). As mentioned above, there is also no graphical browser in HyperCard for showing the local or global arrangement of cards within the network.

These shortcomings may not be so severe, however. Many “hacks” and “workarounds” can be programmed in HyperTalk, and there seems to be at Apple a considerable commitment to maintaining and improving this product. Given the very positive early reviews of this product and the strong user enthusiasm, we can expect HyperCard to become something of a standard semi-hypertext system.

Figure 7: A snapshot of the PlaneText screen.



made to the contents of those files. The result of this is that standard UNIX tools (compilers, text processors, editors) can be run on PlaneText files without the need for the tools to be a any special file characteristics. Put another way, files can

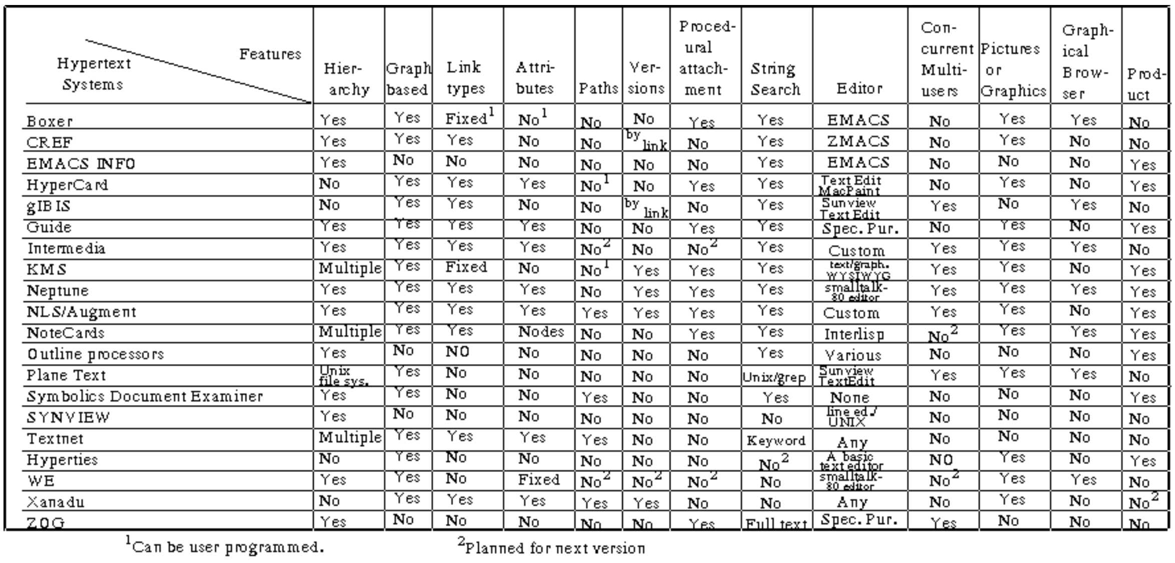
2.4.7 MCC's PlaneText

PlaneText, developed by Eric Gullichsen in STP, is a very recent addition to the family of general hypertext systems [Gull86a]. Created initially as a quick prototype of hypertext, the first version soon found application by several STP groups working with large amounts of relatively unstructured information. The adoption of the prototype as a tool led concurrently to the realization of the value of hypertext for such applications and the weaknesses of that early version. This had lead to a "hyperactivity" project with STP to design a next generation hypermedia system tailored to software engineering. One application of the system is that the PlaneText user manual is accessible in the computer as a PlaneText network (see parts of Figure 7).

PlaneText is based on the Unix file system and the SUN SunView window manager. Each node is a Unix file, and links appear as names in curly brackets ({}) whose display can be turned on and off. Links are implemented as pointers saved in separate files, so that the linked files themselves are not changed by creating hypertext references between them. This design allows for the smooth integration of hypertext into the rest of the Unix-based computational environment, including such tools as Mail and News, and allowing hypertext annotation of standard source code files. In addition, the Unix file directory system serves as a "free" mechanism for creating hierarchical structures among nodes(20).

One limitation of PlaneText is the use of the SunView text editor, which is a minimal screen editor and which, because it is part of a standard commercial software package, is not amenable to experimentation and extensions(21) (see Section 6.3).

PlaneText has already been a useful tool in analyzing complex design information. It is perhaps too early to say, however, how PlaneText will rank in the world of hypertext, since it will only be directly available to participant companies in MCC/STP. The rest of the world must wait until one or more of these companies develops PlaneText into a product.



Hypertext Systems \ Features	Hier-archy	Graph based	Link types	Attri-butes	Paths	Ver-sions	Proced-ural attach-ment	String Search	Editor	Con-current Multi-users	Pictures or Graphics	Graph-ical Brow-ser	Prod-uct
Boxer	Yes	Yes	Fixed[1]	No[1]	No	No	Yes	Yes	EMACS	No	Yes	Yes	No
CREF	Yes	Yes	Yes	No	No	by link	No	Yes	ZMACS	No	Yes	No	No
EMACS INFO	Yes	No	No	No	No	No	No	Yes	EMACS	No	No	No	Yes
HyperCard	No	Yes	Yes	Yes	No[1]	No	Yes	Yes	Text Edit MacPaint	No	Yes	No	Yes
gIBIS	No	Yes	Yes	No	No	by link	No	Yes	Sunview Text Edit	Yes	No	Yes	No
Guide	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Spec. Pur.	No	Yes	No	Yes
Intermedia	Yes	Yes	Yes	Yes	No[2]	No	No[2]	Yes	Custom	Yes	Yes	Yes	No
KMS	Multiple	Yes	Fixed	No	No[1]	Yes	Yes	Yes	text/graph. WYSIWYG	Yes	Yes	No	Yes
Neptune	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	smalltalk-80 editor	Yes	Yes	Yes	Yes
NLS/Augment	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Custom	Yes	Yes	No	Yes
NoteCards	Multiple	Yes	Yes	Nodes	No	No	Yes	Yes	Interlisp	No[2]	Yes	Yes	Yes
Outline processors	Yes	No	NO	No	No	No	No	Yes	Various	No	No	No	Yes
Plane Text	Unix file sys.	Yes	No	No	No	No	No	Unix/grep	Sunview TextEdit	Yes	Yes	Yes	No
Symbolics Document Examiner	Yes	Yes	No	No	Yes	No	No	Yes	None	No	No	No	Yes
SYNVIEW	Yes	No	No	No	No	No	No	No	line ed./ UNIX	No	No	No	No
Textnet	Multiple	Yes	Yes	Yes	Yes	No	No	Keyword	Any	No	No	No	No
Hyperties	No	Yes	No	No	No	No	No	No[2]	A basic text editor	NO	Yes	No	Yes
WE	Yes	Yes	No	Fixed	No[2]	No[2]	No[2]	No	smalltalk-80 editor	No[2]	Yes	Yes	No
Xanadu	No	Yes	Yes	Yes	Yes	Yes	No	No	Any	No	Yes	No	No[2]
ZOG	Yes	No	No	No	No	No	Yes	Full text	Spec. Pur.	Yes	No	No	No

[1]Can be user programmed. [2]Planned for next version

Figure 8: A summary of implemented hypertext systems and their features. See text for explanation.

2.5 Summary

Some of the features of the systems reviewed in this section are summarized in Figure 8. In the figure, each column represents a feature of some hypertext systems. The meanings of the column names are:

- Hierarchy: is there specific support for (possibly multiple) hierarchical structures?
- Graph-based: is there direct support for non-hierarchical (cross reference) links?
- Link types: can links have types (i.e. is link type information used directly by the system)?
- Attributes: can user supplied attribute/value pairs (that are used by the system) be associated with nodes or links?
- Paths: (also, "trails") can many links be strung together into a single permanent object?
- Versions: can nodes or links or both have more than one version?
- Procedural attachment: can arbitrary executable procedures be attached to events (such as mousing) at nodes and/or links?
- String Search: can the hyperdocument be searched for strings (including keywords)?
- Editor: what editor is used to create and modify the contents of nodes?
- Concurrent Multiusers: can several users edit the hyperdocument at the same time?
- Pictures or Graphics: is some form of pictorial or graphical information supported in addition to text?
- Product: is the system commercially available as a supported product?

The systems in this section were presented in terms of four broad categories: macro literary systems, problem exploration systems, structured browsing systems, and general hypertext technology. It should be mentioned that a major area of research within MCC/STP is into systems which would aid the entire process of design, particularly the upstream aspects, and that such systems will need to combine at least the features of problem exploration and structured browsing systems. Indeed, in time this may become an important fifth category in which to consider hypertext systems of the future.

It should also be noted that the history of hypertext presented here suggests that the concept and advantages of hypertext were clear several decades ago, but that widespread interest in hypertext was delayed until the supporting technology was cheap and readily available. This may be misleading. There is the feeling among the "elders" of the field that something else has changed as well — that there is today an easy acceptance of the role of the computer as a tool for processing ideas, words, and symbols — in addition to numbers and mere "data" — and as a vehicle of inter-human communication, and that this acceptance is a recent development. Those who gave presentations of their hypertext systems 20 years ago, using expensive state of the art hardware, report that there was a strong lack of interest in the computer science community, and that it seemed to stem as much from a lack of understanding of the basic concepts as from a lack of hardware resources.

If this is so, then the recent upsurge in interest in hypertext may signal that as a profession we are now ready to consider computer technology as being as much a tool for communication and “augmenting the human intellect” as for analysis and information processing. Hypertext is certainly a large step in that direction.

3 The Essence of Hypertext

It is tempting to describe the essence of hypertext as a documentation form consisting of high-speed text windows and machine processable links between the textual “chunks”. But this is a little like describing the essence of a great meal by listing the ingredients. Perhaps a better description of hypertext is that it is a new technologically-enabled medium for thinking and communication.

The thinking process does not build new ideas one at a time, starting with nothing and turning each idea out as a finished pearl. Thinking seems rather to proceed on several fronts at once, developing ideas at different levels and on different points in parallel, each idea depending on and contributing to the others.

The recording and communication of such entwined lines of thought is challenging, because communication is in practice a serial process, and is in any case limited by the bandwidth of human linguistic processing. Oral communication of parallel themes must resort to heavy marking of items (e.g. stresses, pauses, intonation) which must be remembered by the listener while other lines of argument are developed. Graphical forms can use lists, figures, and tables to present ideas in a less than strictly linear form, and the reader/viewer is allowed to shift his gaze to be reminded of the content of items which must be understood together. One of the most challenging aspects of good writing, especially good technical writing, is the presentation of several parallel lines of a story or an argument in a way that they weave together coherently.

But traditional flat text binds us to writing and reading paragraphs in mostly-linear succession. There are tricks for signalling branching in the flow of thought when it is necessary: parenthetical comments, footnotes, inter-sec\\-tional references (e.g. “see Chapter 4”), and bibliographic references all allow the author to say “here is a related thought, in case you are interested”. There are also many rhetorical devices for indicating that a set of ideas belong together but are being presented in linear sequence. But these are rough tools at best, and often do not provide the degree of precision or the speed and convenience of access that we would like.

Hypertext, however, allows and even encourages the writer to make such references (see Section 4, on the Advantages of Hypertext), and allows the readers to make their own decisions about which links to follow and in what order (see Section 5, on the Disadvantages of Hypertext). It eases the restriction on the thinker and writer that any given idea is either “in”, and must be made a part of the flow of a paper's stream of thought, or is “out” and will have to be saved somewhere else, if at all. And it allows annotations on a text to be saved separately from the reference document, and yet still be tightly bound to the referent. Thus we see that the linked-ness of hypertext provides much of its power. High-speed windows are necessary for following links transparently, but it is the machine processable links which extend the text beyond the single dimension of linear flow (see Section 4 for a more specific description of the benefits of hypertext).

At the same time, there are applications in which it is the “node-ness” of hypertext which is so powerful. Particularly when using hypertext as a thinking, writing, or design tool, there can be a natural correspondence between the objects in the world and the nodes in the hypertext data base. By taking advantage of this object oriented aspect, a hypertext user can build flexible networks which model his problem (or solution). In this application the links are important -- since they form the “glue” that holds the nodes together -- but one senses that much of the action is happening in the nodes themselves. Thus in some applications it is the machine processable links which distinguish hypertext; in others, it is the modularity of chunking text into nodes that is more important.

Thus we are confronted with a kind of paradox: in some applications, it is the machine processable links which distinguish hypertext; in others, it is the modularity of chunking text into nodes that is more important. But more fundamentally, hypertext seems to be useful both as a database tool and as a representation scheme. As a database tool the feature of explicit machine supported links provides a novel way of accessing data, one which is distinct from using queries.

From a computer science viewpoint the essence of hypertext is precisely that it is a hybrid that cuts across traditional boundaries. Hypertext is a *database method* , providing a novel way of directly accessing data which is quite different from the traditional use of queries. At the same time, hypertext as a *representation scheme* is a kind of "semantic network" which mixes informal textual material with more formal and mechanized operations and processes. Thirdly, hypertext is an *interface modality* that features "control buttons" (link icons) which may be arbitrarily embedded within the content material by the user. These are not separate applications of hypertext -- they are metaphors for a functionality that is an essential union of all three. In this report we will concentrate on the lower level issues relating to nodes and links and the database view. Specifically, the next two sections do just that.(22)

3.1 The Power of Linking

In the next two sections hypertext links and nodes are explored in more detail as the basic building blocks. The following section explores methods of linking and the dynamics of following links.

3.1.1 Link Following

We have said that the most distinguishing characteristic of hypertext is that there is machine support for the tracing of references. This can be as simple as references having numbers or short names which the user can enter manually to indicate a desire to follow the reference link. For example, some hypertext systems (e.g. ZOG and help systems) provide a numbered menu of links to each successor frame, and the user selects a link by typing the number or letter associated with the name of the link. Of course, following links can be as sophisticated as having the references themselves be in a distinctive font or icon and be mouse-sensitive, with the system responding to a mouse button click (called "mousing") on such a reference by bringing the referenced node up onto the screen.

What are the requirements for links and link behavior? Support for the tracing of links is essential to hypertext, but it is difficult to define strictly. For example, how much effort can we expect of the user in following a reference and still consider the system hypertext? After all, if the references in a document are just the names of files, any editor could be a candidate for hypertext because the user can just enter the command for editing a file and then type in the name of the desired file. The accepted lower limit of referencing support can be specified as follows: no more than a couple of key strokes (or mouse movements) are required from the user to follow a single link. In other words, the interface must provide that links act like "magic buttons" that quickly and easily transport the user to a new place in the hyperdocument.

Another aspect of hypertext that is essential is the speed with which the system responds to referencing requests: there must be only the briefest delay in following a link, and much design work goes into this feature in most systems. It is widely accepted that a delay of more than one or two seconds is too long (Halasz, personal communication; also, [Mant79]). This constraint may appear at first to be more like a petty concern of impatient users than a hard requirement for hypertext. However, the need for rapid response is just as critical as the existence of machine support for reference tracing -- if I can go to a referenced page in a book faster or easier than the computer system can bring up that "page", then it would seem that there is nothing very "hyper" about my hypertext system.

However, because of fundamental hardware limitations, not all link traversals can be instantaneous. No matter how fast a hypertext system is in following links, one can always specify a size of hyperdocument that either will not fit with the system or will cause unacceptably slow system response. Thus, as important as rapid response is providing cues to the user about the *expected delay* for a given query or traversal; for example, some visual feature of the link icon could indicate whether the destination node is in memory, on the disk, somewhere else on the network, or archived offline, suggesting increasing retrieval times.

The benefits of electronic storage and manipulation of my text are balanced by the loss of many familiar and important features of printed text: portability, ease of annotation, and even the physical concreteness of paper documents.(23) It will be difficult to provide on a computer screen the equivalent effectiveness of the more subtle tactile and visual cues such as size, color, texture, absolute and relative position, weight and heft, etc. that paper documents offer. The promise of computer mediated document handling is that the special talents of computer systems (e.g. high speed indexing within enormous documents, multiple complementary organizations of the same material, etc.) make up for the loss of the unique features of paper. In summary, if machine-supported links are the essence of hypertext, then performance of link tracing is also essential.

3.1.2 Properties of Links

Links connect things. There are two fundamental questions one can ask about a link: *What things does it connect?* and *How does it connect them?* These questions will form the basis of our survey of peroperties of links.

Functions of Links

- First, however, we must observe that links can be used for several different functions:
- They can connect the reference to another document to the document itself.
- They can connect a comment or annotation to the text about which it is written.
- They can indicate that some text is a subsection of some other piece of text, or other kinds of organizational information (i.e. the link between a table of contents entry and its section).
- They can connect two successive pieces of text, or a piece of text and all of its immediate successors.
- They can connect entries in a table or figure to longer descriptions, or even subtables or figures.

Characteristics of Links

Links can have names and can, in fact, have a rich set of properties. Some systems allow display of links to be turned on and off (i.e., turned off so that a document appears as ordinary text).

The introduction of links into a text system means that an additional set of mechanisms must be added for creating new links, deleting links(24), changing link names or attributes, listing links, etc.

The two ends of a link

Links have two ends, and are usually considered to be directed, although it is common to support going "backwards" along the link. The origination of the link is called the "link source", and usually acts as the *reference* . The source can logically be either a single point or a region of text. At the other end, the "destination" of the link usually functions as the *referent* , and can also be either a point or a region (see Figure 9).(25) For example, the most common implementation of hypertext links (used, for example, by ZOG, NoteCards, and PlaneText) is to have the source be a point and the destination be a node. For most purposes, a node can be considered to be a region. This is convenient for many of the applications listed above, because it provides the sense of a chunk of text -- a region -- written about or referenced by some smaller textual chunk, often a sentence. Since we are accustomed to single point references to sentences (e.g. footnotes), it is natural to accept a link with a point source.

Points and regions

A *point* (source or destination) is some icon indicating the presence of the link; it usually shows the link's name, and indicates whether it is a source or destination point (in systems, such as Neptune, which support both). It is important that a point behave as a single character or icon: the editor cursor should jump over it as if it were a single character (i.e. it must be immutable using the standard text editor); and it must be able to move with its containing text if that text is moved. But there is a problem with copying and deletion of text containing link points: what does it mean to delete one end of a link, or to make a copy of it somewhere else in the file? The options are: (1) to have copying and deletion of text containing link points signal that the link is being copied or deleted, and optionally to request verification from the user; (2) to prohibit such actions until any contained links have been dealt with via the link manipulation mechanisms; and (3) to have the link point behave as if it were uncopyable and undeletable, so that it remained after a deletion and did not copy. Most systems implement the first option, on the assumption that the user really regards the link as an integral part of the text, so that it should be deleted or copied with its containing text.

A *region* (source or destination) is a set of contiguous characters which is displayed in some way as being a single unit. In Figure 9 the link destination is a region, in this case, an entire node. This is the easiest way to implement regions, since any attempt to set a region off from neighboring text within a node raises a tough implementation issue: how to display the region to the user. It should be highlighted somehow, using reverse video, fonts, or color. But these options open the door to tough issues about the handling of overlapping regions -- how can several overlapping regions be highlighted clearly at once? The options seem to be: only displaying a single region at a time (but then where are the other link regions and how do they get selected?), or using some elaborate color scheme to indicate regions and their overlap. Of course, an easy resolution is to disallow overlapping regions. (The Intermedia designers propose to draw a light box around

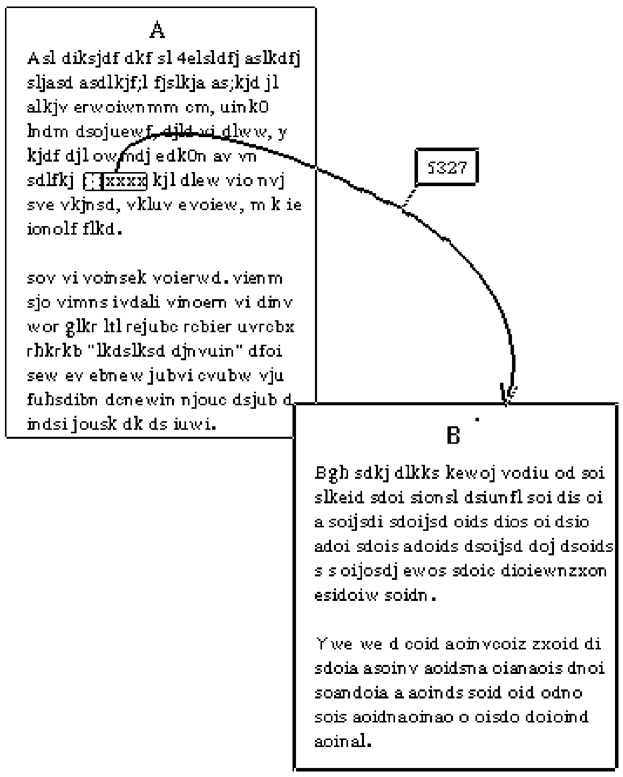


Figure 9

An example of a link with a point source and a region destination. The source of the link is a token in the text of document A which contains a textual identifier ("xxx"). The identifier may be (1) the name of the destination node (in this case it would be "B"), (2) the name of the link, or (3) an arbitrary string which is neither the name of the link nor the destination node. The destination of this link is a region which *is* the node B. The link has an internal name (5327) which is normally invisible to the user.

---

regions and a darker box around region/region overlaps, thus showing a single level of overlapping [Garr86a]; perhaps this is sufficient.)

Another difficulty with link regions is that it is less clear how to show the name of the link, unless it is done arbitrarily at the beginning or end of the region (and again, this name must be uneditable).

Finally, one must consider copying, movement, modification, and deletion of the region and of substrings within it. For example, when one moves a major portion of the text in a destination region someplace else in the node, does the link destination move with it or stay with what remains? Also, special provisions must be made for editing (deleting, moving, or copying) the defining end points of a region.

Reference by Name and by Value

Reference by name is the weakest form -- it means that the link source simply has the name of the destination node, and thus is invalidated if the name or location (directory) of the destination node is changed. This is the easiest way to implement links, but is generally considered too brittle.

Reference by value means that the destination has been made into a node with its own internal name, and the link points to this node. Most systems take the extra trouble to implement these internal (and usually invisible to the user) names, since it allows nodes to be moved or renamed without loosing their in-links.

Organizational, referential, and keyword links

There are two methods for *explicitly* linking two points in hypertext: referential (non-hierarchical) and organizational (hierarchical) links. In addition, there is an *implicit* linking that occurs through the use of keywords. The referential links are the kind discussed above, the kind which most clearly distinguish hypertext, so no more need be said here about them.

Organizational links connect a parent node with its children and thus form a strict tree subgraph within the hypertext network graph. They correspond to the "IS-A" (or "Superconcept") links of semantic net theory ([Wood75, Brac77]), and thus are treated quite differently from referential links(26). For example, rather than appearing as explicit highlighted tokens in each node, organizational links are sometimes traversed by a separate mechanism at the node control level (i.e. special goto-parent, goto-first-child, and goto-next-sibling commands). In other cases there are oganizational *nodes* (e.g. "toc" nodes in Textnet and "FileBoxes" in NoteCards) which record the organizational structure. The relative importance of hierarchical and non-hierarchical structures is discussed below.

One of the chief advantages of text storage on a computer is the feature of searching large and complex documents and sets of documents for substrings and keywords.(27) Naturally, this is also a valuable aspect of hypertext (though with hypertext the search problem may be somewhat more complex, simply because it is search in a graph). Indeed, most users of large hyperdocuments insist on having some mechanism for scanning them.

We propose that in hypertext initiating a search is done for the same reason that one follows a link, i.e. to get from one node to another. The difference is that explicit hypertext links typically connect one node to another, whereas keyword search links a virtual *set* of nodes. Therefore search of the hypertext graph amounts to a definition of a network of implicit, virtual links, and keywords can be regarded as a kind of implicit, computed link.(28) The value of this insight is that it may allow design of a hypertext interface which is consistent across all link-tracing-like activities.

Link Sources

Almost all modern versions of hypertext allow link sources (i.e. references to other nodes) to occur freely within the body of a node. A few systems, e.g. ZOG, gIBIS, and most help systems, only provide out-links from the entire node; this form is fine for these browsing systems, but is inadequate for applications in which annotations and references need to be placed within text.

The most common means of displaying link sources is as short highlighted textual fields, in which the text is the name of the link itself or the name of the destination node. The link source is usually a point (cannot be edited). Brown's Intermedia allows regions for both source and destinations.(29)

Link Destinations

Here the story is the complement of link sources: link destinations are almost always whole nodes. Only one system reviewed here, the Tektronix Neptune system, provides for link destinations to occur freely (as point destinations) within text in a node. It is not yet clear what the benefits and drawbacks are of point and region destinations (although these issues are discussed at some length in [Garr86a]).

Link Direction

In almost all implementations links are designed to be traversed in either direction, but this raises a problem, because the names placed on links are often non-symmetrical relations (e.g. references, summarizes, etc.). A link labeled "References" in one direction should be labeled **Is-Referenced-By** when followed in the other direction. This is fine if there are a fixed set of link labels so that the systems designers can build in the symmetrical pairs, but is more of a problem if new link labels can be designated by the user. Note that there are some generally difficult linguistic problems with symmetrical relations, e.g. the reverse of **Is-In-Front-Of** is not necessarily **Is-In-Back-Of** . Perhaps the easiest way to avoid this problem is to use a single link label and to include the direction as part of the display of the link and its endpoints.

Tracing links backwards also leads to a problem when the link sources are point sources. The semantics of a point source is that signalling the source in some way (e.g. mousing it) causes the link destination to come into view. But signalling the destination for a backwards traversal does not mean that the point source necessarily comes into view: it can mean only that the window containing the source is indicated. Locating the point source inside the node/window may be left to the user -- this is a design decision.

Some Characteristics of a Representative Notefile

Uses: mostly personal information management; some authoring
Size: 3.7 Mbytes

Total cards: 1577
- FileBoxes: 502
- Cards: 1075
  - Text: 667
  - Mail: 367
  - Other: 41

Total links: 3460
- Filing: 2521 (hierarchical) - 73%
- Mail: 678
- Other: 261 (non-hierarchical) - 7.5%

Total link types: 32
- System: 32
- Mail: 5
- User: 14

Table 3: Some Characteristics of a Representative Notefile (from Halasz, personal communication)

To Tree or Not to Tree

Some hypertext systems (e.g. NLS/Augment) support only hierarchical structure, others provide no specific support for hierarchical structure (e.g. Xanadu, Hyperties), and others support both kinds of structures (e.g. Textnet, NoteCards).

It would be nice to know how sufficient strictly hierarchical structure is, in general, and, more deeply, for which applications it is sufficient and for which it is not. On the one hand, it could be argued that the human mind is a "taxonomy building machine", and that hierarchical structures are the most natural structure for organizing information. On the other hand, there are clearly cases where cross-hierarchical links are required.(30) One source of evidence about the relative importance of these two alternatives can come from examination of the practices of someone using hypertext for an actual project: to what extent do they organize their material (hierarchically) and to what extent are they making inter-nodal references? Frank Halasz, one of the developers of NoteCards, has gathered statistics on the "hyperspace" of a single representative NoteCards user (see Table 3); this person had 1577 nodes (cards) in all, 502 of which were "FileBoxes" (hierarchical nodes). Connecting these nodes were a total of 3460 links, 2521 (73%) of which connected FileBoxes to each other or to individual notecards, 261 (7.5%) of which were non-hierarchical referential links, and the remainder of which were mail links (used by the system to tie mail messages to other nodes). This example, for what it is worth, suggests that hierarchical structure is very important in organizing a hypertext network, and that referential links are important but less common.

One advantage of strictly tree-oriented systems such as NLS/Augment, ZOG, EMACS INFO, and the outline processors, of course, is that the command language for navigation is considerably simpler: from any node, the most one can do is go to the parent, a sibling, or a child. This simplicity can also help with avoiding the disorientation problem, since a simpler cognitive model of the information space will suffice.

Of course, the great disadvantage of any hierarchy is that its structure is a function of the few specific criteria that were used in creating it. For example, if one is interested is exploring what is common among life forms that live in the sea versus those that live on land, the traditional decomposition into plants and animals splits the world in a way that may make it harder to spot generalities among land- and sea-based life. The creator of a hierarchical organization must anticipate the most important criteria for later access to the information. One solution to this limitation is to allow the information elements to be structured into *multiple hierarchies* , thus allowing the world to be "sliced up" into several orthogonal decompositions. Any hypertext system which has hierarchy nodes, such as Textnet (*toc* nodes) and NoteCards *FileBox* nodes, can do this quite easily, and these are the only systems reviewed here which explicitly claim to support multiple hierarchies. Indeed, one early user of NoteCards used the system in doing the research and writing for a major project paper; he imposed one organization on the data and his writings while doing the research, and then quite a different (yet coexistent) organization on the same material to produce his paper. This user reported that he benefitted greatly from having his research notes and thesis in NoteCards. As a generalization, it seems that more engineering oriented hypertext users prefer heirarchical organizations, whereas those with more artistic or humanistic applications place more value on cross referencing.

3.1.3 Extensions to Basic Links

There are several ways in which the basic notion of hypertext links can be extended. One is to consider that links can connect more than two nodes -- such *cluster links* could be useful for referring to several annotations with a single link, and also have a natural application for providing specialized organizational structures among nodes -- indeed, the *toc* nodes of Textnet and the *FileBoxes* of NoteCards are both cluster links.

Another useful feature is the ability to place attribute/value pairs on links and to query the network for them. The Neptune system, for example, has an architecture that is optimized for this function. Coupled with specialized routines in the database interpreter (the "HAM"), these attribute lists allow users to customize links in several ways, including devising their own type system for links and performing high-speed queries on the types.

Finally, one can imagine that it could be useful to allow procedural attachment to links (as is also done in Neptune and Boxer), so that traversing a link also performed some user-supplied side-effect such as customizing the appearance of the destination node.

3.2 Hypertext Nodes

While we have said that the essence of hypertext is machine-supported linking, we must also consider the nodes which become so linked. Most users of hypertext favor using nodes which express a single concept or idea, and are thus much smaller than traditional files. In a sense, hypertext introduces an intermediate level of organization between characters and files, a level which has the vaguely semantic aspect of being oriented to the expression of ideas. But this sizing is completely at the discretion of the hypertext writer -- nothing prevents a writer from packing a whole book into a single node, nor, at the other extreme, from using one node per word. Indeed, the process of determining how to modularize a document into nodes is an art, because its impact on the reader is not well understood (but see [Shas86]). For many applications a good length for nodes is considered to be from one to several paragraphs.

3.2.1 The Modularization of Ideas

Hypertext invites the writer to modularize ideas into units such that (a) an individual idea can be referenced elsewhere (i.e. this is a scope of reference concern) and (b) alternative successors of a unit can be offered to the reader (e.g. more detail, an example, bibliographic references, or the logical successor). But the writer must reckon with the fact that a hypertext node, unlike a textual paragraph, is a strict unit which does not blend seamlessly with its neighbors. Some hypertext systems (Notecards, CREF, Boxer, FRES, NLS) allow nodes to be viewed together as if they were one big node, and this is an essential option for some applications, e.g. writing and reading prose. But the boundaries between nodes are discrete, and require sometimes difficult judgements about how to cleave the subject matter into suitable chunks.

The process of identifying a semantically based unit, such as an idea or concept, with a syntactic unit, such as a paragraph or hypertext node, is not unique to hypertext. Manuals of style notwithstanding, traditional text has rather loose conventions for modularizing text into paragraphs, but this is acceptable because paragraph boundaries have a relatively minor effect on the flow of the reading. Paragraph boundaries are sometimes provided just to break up the text and give the eye a reference point. Thus, decisions about the distribution of sentences among paragraphs are not critical.

Hypertext, on the other hand, enforces a rather stern information hiding -- the only clue you have as to the contents of a link destination node is the name of the link (or the name of the node, if that is provided instead). The writer is no longer making all the decisions about the flow of the text: in hypertext the reader can and must constantly decide which links to pursue. If these links come at the end of a node, then it is a natural place to stop and consider which path to take. But if the links punctuate the text one must occasionally suspend the task of reading one node in order to initiate the task of checking out and perhaps reading another node. And of course the process is recursive. This is one reason why performance of hypertext systems is so critical: if one must wait more than a moment to see destination nodes one is likely not to follow many links (see also Section 5.2). Intermedia provides a one sentence "explainer" that pops up to explain any link as a shortcut in making this judgement.

But in any case, writing and reading hypertext imposes on the writer and reader the need for more *process awareness* , since one must constantly be making meta-level judgements about the flow of ideas. Of course, for the applications for which hypertext is best suited these are the kind of judgements that one is making anyway, and hypertext merely offers a way to act directly on these judgements and see the results quickly and graphically.

3.2.2 Node Size

Aside from the criterion of one idea or concept per node another constraint on node size is the practical consideration in most hypertext systems that it takes some time for a new node/window to be created when a link is activated, hence one does not generally create many small nodes, due to the excessive overhead of link traversal for the reader.

Yet another constraint on node size is that in most hypertext systems nodes are the lowest level of resolution of link destinations, making the size of the link destination equivalent to the size of the node. This means that if you wish to refer to one paragraph in a long document you have two options: place the link source in the referring node and make the whole document the link destination, or place the link source in the document, at the paragraph, and place the annotation in a node that is the link destination. Obviously the latter is the preferable solution, but it has the problem, discussed above, that in many systems an attempt to backtrace the link from the annotation node will only get the whole document, not the link source itself, leaving the user to find the referenced paragraph unaided. (The best solution is probably to allow, but not force, region to region links.)

Thus there is a tension between wanting many small nodes, in order to maximize the degree of organization of the information, and just a few larger nodes, to minimize the delay and complexity of navigating between nodes. One of the chief goals of hypertext researchers should be to find ways of decreasing this delay and complexity so that there is the greatest possible range of useful node sizes available to users.

3.2.3 Ideas as Objects

While it is difficult to document, there is something very compelling about reifying the expression of ideas into discrete objects to be linked, moved, and changed as independent entities. Alan Kay and Adele Goldberg [Kay77b] observed of Smalltalk that it has the feature of giving objects a perceptual dimension by allocating them a rectangular piece of screen real estate. It offers enhanced retrieval and recognition over computer processed flat documents, because to a much greater degree abstract objects are directly associated with perceptual objects, the windows and icons on the screen.

Paragraphs, sections, and chapters in a book, viewed through a standard text editor or word processor, simply don't stand out as distinct entities. This is particularly apparent when one can both view one's text hierarchically, e.g. as an outline, at the same time that one is working on adding new sections and embellishing existing ones. People don't think in terms of “screenfulls”, they think in terms of ideas, facts, and evidence, and hypertext, via the notion of nodes as individual expressions of ideas, provides a vehicle which respects this kind of thinking and working.

3.2.4 Nodes with Types

In some hypertext systems nodes are considered to be untyped, that is, they are just generic boxes for holding text. For some applications it is useful to consider typing the nodes, so that they come in different “flavors”. This is particularly useful if, in addition, one is considering giving the nodes some internal structure, since the types can be used to differentiate the various structural forms. Systems such as NoteCards, Intermedia, and gIBIS make extensive use of typed nodes.

For example, in our research in STP we have been implementing a hypertext interface for a design environment called the *Design Journal* . The purpose of the Design Journal is to be a kind of active scratchpad in which the designer can deliberate about design decisions, both individually and in on-line design meetings, and can integrate the design itself with this less formal kind of information. For this purpose we have speculated about providing a set of natural kinds of nodes for the designer to use: notes, goals/constraints, artifacts, and decisions. Notes would be used for everything from “Ask Bill for advice on Module X” to specific problems and ideas relating to the design. Goals/constraints are for the initial requirements as well as discovered constraints within the design. Artifacts are the elements of the output: The Design. And decisions are used to capture the branch points in the design process, the alternatives considered by the designer, and some of the rationale for any commitment (however tentative) that has been made. Assumptions are also captured, in the form of decisions with only one alternative.

Having typed nodes helps the user differentiate at a glance the broad classes of nodes that he is working with, by providing a specialized color, size, or iconic form for each node type, and may also be used to segregate nodes of different types into different parts of the screen. In addition, it is possible to define specialized operations on different node types, such as display views, dependency tracking, and internal computations.

3.2.5 Semistructured Nodes

To continue the example of the Design Journal, we have developed a model for the internal structure of decisions which suggests that not only does one require typed nodes, one may also wish to treat nodes as “active forms” to be filled out by the user. The advantage of providing a template for node contents is both to assist the user in being complete and to assist the computer in interpreting the textual content of the nodes. That is, the less that the content of a node is undifferentiated natural language (e.g. English) text, the more likely it will be that the computer can do limited kinds of processing and inference on the textual subchunks. This is closely related to Malone's notion of “semistructured” information systems [Malo87a].

The name of the model of decision structure is ISAAC [Conk86], and the model proposes that there are four major components to a design decision:

1. An issue, including a short name for the issue and a short paragraph describing it in general terms;
2. A set of alternatives, each of which resolves the issue in a different way, each having a name and short description, and each potentially linked to the design documents or elements that implement the alternative;
3. An analysis of the competing alternatives, including the specific criteria being used to evaluate them, the tradeoff analysis among these alternatives, and links to any data that the analysis draws upon;
4. A commitment to one of the alternatives (however tentatively), or a vector of preferences over the alternatives, and a subjective rating about the correctness or confidence of this commitment.

Without getting into the details of the theory of design process capture, we wish merely to stress here that the internal structure of ISAACs suggests that much more would be happening as one created and filled in such a hypertext node than is the case when no internal structure exists.

Of course, the reader may wonder why each of the elements listed above is not being treated as its own hypertext node. The problem with that treatment in the case of the ISAAC model is that the parts of an ISAAC frame are much more tightly bound together than ISAACs frames are bound to each other. One could not have an analysis part without an alternatives part, for example, yet if we treat them as separate hypertext nodes we have failed to build this constraint into the structure. This issue touches also on the issue of node size: to what extent is information so tightly bound together that (a) you always want to view it together, (b) you never want to take it apart, (c) you rarely even want to reference parts of it outside of the context of the rest.

In hypertext this tension presents itself as the twin notions of nodes with internal structure and composite nodes.

3.2.6 Composite Nodes

One advantage of hypertext is that a collection of paragraphs, represented as nodes, may be traversed in many different orders, depending on the desires of the writer and the reader. However, sometimes one wishes to combine the features of separate nodes and single nodes by gathering several nodes into a single composite node. An ISAAC node can be thought of in this way. Composite nodes are most useful for situations in which the separate items in a bulleted list, or the entries in a table, were each distinct nodes, but which also cohered into a higher level structure (e.g. a list or table). In these cases one would like to be able to group the nodes and treat them as a single *composite node* . Note that this feature tends to attack the fundamental association of one interface object (window) per database object (node), and thus must be done well to avoid complicating the hypertext idiom unduly.

A composite node facility would allow nodes to be collected and structured into a single node. The composite node could be moved and resized (within limits), and would close up to a suitable icon reflecting its contents. However, the subnodes would be separable and rearrangeable through a *subedit* mode. The most flexible means of arranging subnodes is through the use of a constraint language (such as that developed by Symbolics for “Constraint Frames”) in which the subnodes are described as panes in the composite node window and the inter-pane relationships are specified as dynamic constraints on size and configuration.

Another motivation for composite nodes is to help manage the number of named objects in one's environment. Pitman described the problem this way:

> *In this sort of system, there is a never-ending tension between trying to name everything (in which case, the number of named things may grow quickly and the set may become quickly unmanageable) or to name as little as possible (in which case, things that took a lot of trouble to construct may be hard to retrieve if one accidentally drops the pointers to them).*
>
> [Pitm85]

Another problem with composite nodes is that as the member nodes grow and change the aggregation can become misleading or incorrect, and it is often difficult to tell that this is happening. A familiar example of this for writers is the section of a paper which is so thoroughly rewritten over time that the section title is no longer accurate. Sometimes this “semantic drift” can be difficult to catch.

3.3 Analogy to Semantic Networks

The idea of building a directed graph of informal textual elements is similar to the AI concept of semantic nets. A semantic network is a knowledge representation scheme consisting of a directed graph in which concepts are represented as nodes and the relationships between concepts are represented as the arcs (or edges, links, etc) between them.(31) The important idea is that concepts in the representation are indexed by their semantic content rather than by some arbitrary (e.g. alphabetical) ordering. One benefit of semantic networks is that they are natural to use, since related concepts tend to cluster together in the network. Similarly, it is easier to spot a concept which is incompletely or inconsistently defined, since a meaningful context is provided by those neighboring concepts to which it is already linked.

The analogy to hypertext is straightforward: hypertext nodes can be thought of as representing single concepts or ideas, internode links as representing the semantic interdependencies among these concepts, and the process of building a hypertext network as a kind of informal knowledge engineering. The difference is that knowledge engineers are usually striving to build representations which can be mechanically interpreted, whereas the goal of the hypertext writer is often to capture an interwoven collection of ideas. As we shall discuss below, the field of AI also provides some natural extensions to the hypertext concept (such as structured nodes and procedural attachment) which have proven powerful with semantic networks.

4 The Advantages/Uses of Hypertext

Intertextual references are not new. The breakthrough of hypertext is simply that they are machine supported. However, traditional literature is richly interlinked, and is also usually hierarchically organized. The medium of print has for the most part restricted the flow of reading to be following the flow of linearly arranged passages. However, even the process of following links is fundamental in traditional literature, and is much of what library and information science is about. Any one who has done research knows that a considerable portion of that effort lies in obtaining referenced works, looking up cross references, looking up terms in a dictionary or glossary, checking tables and figures, and making notes on notecards. Even in simple reading one is constantly negotiating references to other chapters or sections (via the table of contents or references embedded in the text), index entries, footnotes, bibliographic references, sidebars, figures, and tables. It is not uncommon for readers to be invited to “skip the following section” if they are not interested in greater technical detail.

4.1 But there are problems with the traditional methods.

- Most references can't be traced backwards: unless the author has provided for it with a special table, there is no easy way to find where a specific book or article is referenced in a document, nor can the author of a paper easily determine who has referenced his or her paper.
- It is up to the reader to keep track, as he winds his way down various reference trails, of which documents he has visited and which he is done with. Hypertext automatically stacks the windows for

successive references on top of or near each other on the screen, providing a natural mechanism for winding one's way back over such a trail: as each window is closed, it reveals the document and paragraph that were under consideration when the trail was followed.

- Annotations must be squeezed into the margins or appended someplace else. In many hypertext systems the user is free to add his own links, either between existing documents or to new annotational nodes of his own.
- Finally, following a referential trail among paper documents requires substantial physical effort and delays, even if one is working at a well-stocked library. If the documents are on-line, the job is easier and faster, but no less tedious. The vision of the "global hypertext library" (e.g. [Bush45, Nels80, Trig83]) extends the power of the mouse to on-line access to any document ever published.

4.2 Some advantages of hypertext from the application perspective

The following sections discuss some advantages of hypertext from the application perspective used above in the discussion of existing systems.

4.2.1 New Possibilities For Authoring and Design

Hypertext may offer new ways for authors and designers to work. Authoring is usually thought to be a word and sentence level activity, and clearly the word processor (32) is the right tool for authoring at this level. However, it should also be clear that much of authoring has to do with structuring of ideas, order of presentation, and conceptual exploration. Few authors simply sit down and pour out a finished text, and not all editing is just "wordsmithing" and polish. Here we are using the term "authoring" in the broad sense that it is *design of a document* , and thus is a subtopic of design in general.

More specifically, we propose that the unit of this higher level kind of authoring is the concept or idea, and it is this level of work that is best supported by hypertext, since the concept can be expressed in a node. As the writer thinks of new ideas they can be developed in their own nodes, and then linked into existing ideas -- or left isolated if it is too early to make such associations. Likewise, snippets of text can be set aside in their own node. Given the proper view facility, the hypertext nodes can be treated as outline entries, or organized into a tree, where they can be further refined in the context of the adjoining ideas. Restructuring of the emerging document is much easier with the global view that is allowed by hypertext: the order of presentation of ideas and the structure among them (e.g. whether to discuss A and B at the same level or subsume B into the discussion of A) can be easily understood and changed. Notice that using hypertext as a tool for thinking, writing, and designing stresses its "noded-ness" (see Section 3.2).

If the final product is to be a flat document, however, we note that it is necessary, toward the end of the writing process, to have a view of adjacent nodes which is relatively seamless, so that one can really see if the transition between nodes is rhetorically smooth. Finally, if the document is to be reviewed and critiqued by colleagues, hypertext provides an excellent facility for such annotations (see below).

4.2.2 New Possibilities For Reading and Retrieval

Hypertext may also offer new possibilities for accessing large or complex information sources. A linear (non-hypertext) document can only be easily read in the order in which the text flows in the book. The essential advantage of non-linear text is the ability to organize text in different ways depending on differing viewpoints. Shasha describes it this way:

> *Suppose you are a tourist interested in visiting museums in a foreign city. You may be interested in visual arts. You may want to see museums in your local area. You may only be interested in inexpensive museums. You certainly want to make sure the museums you consider are open when you want to visit them. Now your guidebook may be arranged by subject, by name of museum, by location, and so on. The trouble is: if you are interested in any arrangement other than the one it uses, you may have to do a lot of searching. You are not likely to find all the visual arts museums in one sections of a guidebook that has been organized by district. You may carry several guidebooks, each organized by a criterion you may be interested in. The number of such guidebooks is a measure of the need for a non-linear text system.*
>
> [Shas86]

Another advantage is that it is quite natural in a hypertext environment to temporarily suspend reading along one line of investigation to look into some detail, example, or related topic. Bush described an appealing scenario in his 1945 article:

> *The owner of the memex, let us say, is interested in the origin and properties of the bow and arrow. Specifically he is studying why the short Turkish bow was apparently superior to the English long bow in the skirmishes of the Crusades. He has dozens of possibly pertinent books and articles in his memex. First he runs through an encyclopedia, finds an interesting but sketchy article, leaves it projected. Next, in a history, he finds another pertinent item, and ties the two together. Thus he goes, building a trail of many items. Occasionally he inserts a comment of his own, either linking it into the main trail or joining it by a side trail to a particular item. When it becomes evident that the elastic properties of available materials had a great deal to do with the bow, he branches off on a side trail which takes him through textbooks on elasticity and tables of physical constants. He inserts a page of longhand analysis of his own. Thus he builds a [permanent] trail of his interest through the maze of materials available to him.*
>
> [Bush45]

As we have seen, Bush's notion of a *trail* was a feature of Trigg's Textnet [Trig83], allowing the hypertext author to establish a *mostly linear* path through the document(s). The main (default) trail is well marked, and the casual reader may read the text in that order without troubling with the sidetrails.

4.2.3 Summary

We can summarize the operational advantages of hypertext as:

- ease of tracing references: machine support for link tracing means that all references are equally easy to follow to their referent, or backwards to their reference;
- ease of creating new references: users can grow their own networks, or simply annotate someone else's document with a comment (without changing the referenced document);
- information structuring: both hierarchical and non-hierarchical organizations can be imposed on unstructured information; even multiple hierarchies can organize the same material;
- global views: browsers provide table-of-contents style views, supporting easier restructuring of large or complex documents; global and local (node or page) views can be mixed effectively;
- customized documents: text segments may be threaded together in many ways, allowing the same document to serve multiple functions;
- modularity of information: since the same text segment can be referenced from several places, ideas can be expressed more modularly, i.e. with less overlap and duplication;
- consistency of information: references are embedded in their text -- if the text is moved, even to another document, the link information still provides direct access to the reference;
- task stacking: the user is supported in having several paths of inquiry active and displayed on the screen at the same time, such that any given path can be unwound to the original task;
- collaboration: several authors may collaborate, with the document and comments *about* the document being tightly interwoven (this feature is just beginning to be explored).

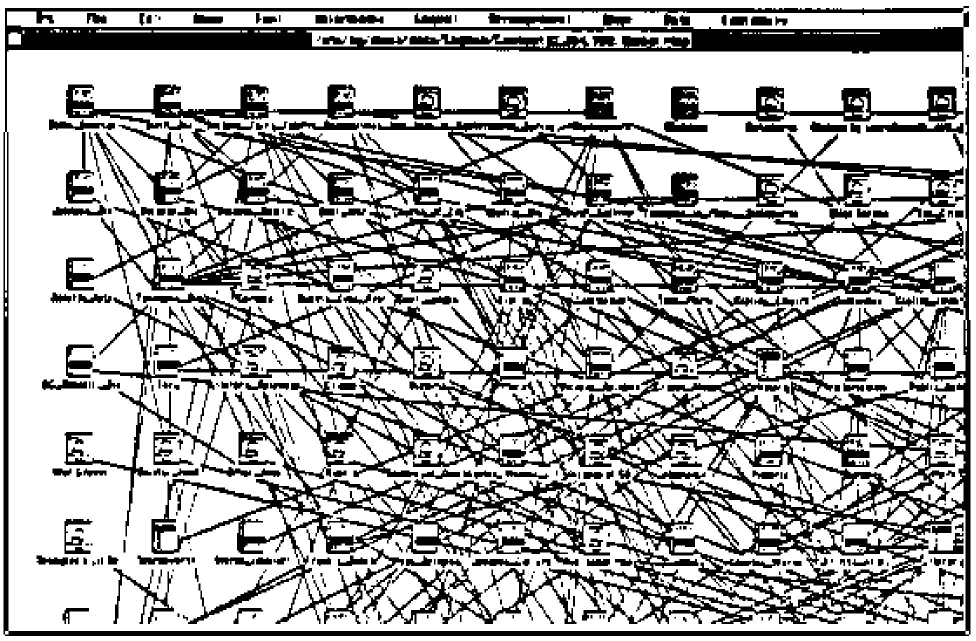
5 Disadvantages of Hypertext

There are two classes of problems with hypertext: those that are problems with the current implementations, and those that seem to be endemic to the fundamental concepts of hypertext. The problems in the first class include delays in the display of referenced material, restrictions on names and other properties of links, lack of or deficiencies in browsers, paucity of interesting or powerful specialized "views" of networks, etc. The following section outlines two problems that are more challenging than these implementation shortcomings, and that may in fact ultimately limit the usefulness of hypertext: *disorientation* and *cognitive overhead* .

5.1 Getting "Lost in Space"

Along with the power of being able to organize information much more complexly comes the problem of knowing (a) where you are in the network and (b) how to get to some other place that you know (or think) exists in the network. Let us call this the *disorientation problem* . Of course, one also has a disorientation

Figure 10: A rats nest of hypertext nodes and links.



problem in traditional linear text documents, but there are only two options about where the desired passage is: earlier or later in the text. Hypertext offers more degrees of freedom, more dimensions in which one can move, and hence a greater potential for the user to become lost or disoriented. In a network of 1000 nodes, it is easy to imagine that information could become hard to find or even forgotten altogether (see Figure 10).

There are two major technological solutions for coping with disorientation: graphical browsers and query/search mechanisms.

5.1.1 One type of solution: Graphical browsers

One solution to disorientation may lie in finding ways to tap the enormous human mental resources for visuo-spatial processing, or, in more familiar terms, computer graphics. Quite a bit of work is being done on graphical approaches to database management (e.g. [Done78, Bolt79, Hero80]; graphical and non-graphical approaches are discussed in [Weye82a]; the adventurous reader might also wish to check [Pass84]).

Hypertext browsers use a graphical display to provide higher-level views of nodes and links. By placing networks in a two or three dimensional space, providing them with features useful in visual differentiation (e.g. color, size, shape, texture), and maintaining certain similarities to our physical environment (e.g. no two objects occupy the same space, things only move if moved, etc.), browser designers are able to create quite viable virtual spatial environments. Users orient themselves by visual cues, just as when they are walking or driving through a familiar city.

However, there is no natural topology for an information space, except perhaps that higher level concepts go at the top or on the left side, so until one is familiar with a given large hyperdocument, one is by definition disoriented. In addition, it is very difficult to maintain an adequate virtuality for a large or complex hypertext network. Such parameters as (a) large numbers of nodes, (b) large numbers of links, (c) frequent changes in the network, (d) slow or awkward response to user control inputs, (e) insufficient visual differentiation among nodes and/or links, and (f) non-visually oriented users combine to make it practically impossible to abolish the disorientation problem with a browser alone.

5.1.2 Another type of solution: search and query

One solution to this dilemma is to apply standard database search and query techniques to locating the node or nodes which the user is seeking. This is usually done by applying some combination, using boolean operations, of keyword and full string search and predicates on other attributes of nodes or links (e.g. author, time of creation, type, etc.). Similarly, one can filter or "ellide" information so that the user is presented with a managable level of complexity and detail, and can shift the view or the detail suppression while navigating through the network. However, much research remains to be done on effective and standardized methods for ellision.

There are two major technological solutions for coping with disorientation: graphical browsers and query/search mechanisms.

5.1.3 Retrieval of Iconized Nodes: A Subproblem

When a node has been set aside, so that only its icon is on the screen, there is a subproblem of the disorientation issue: finding the icon of the node one has in mind. (The use of icons is discussed more in Section 6.2.) Although there have been only a few studies on the effectiveness of spatial layout in recall and retrieval [Jone86c], it is certainly the case that the desktop metaphor will provide some aid in organizing a busy hypertext workspace.

5.2 The Cognitive Task Scheduling Problem

One of the problems with authoring in hypertext is that it is difficult to become accustomed to the additional mental overhead required to create, name, and keep track of links. We call this "cognitive overhead". Suppose one is writing about X, and a related thought, about Y, comes to mind and seems important enough to capture. Ideally hypertext allows you to simply "press a button" (i.e. some mouse or keyboard action) and a new, empty hypertext window pops onto the screen, in which you record Y. Then you press another button, the Y window disappears, and you are in the X window right where you were when Y occurred to you.

Unfortunately, things are a bit more complex than this. If Y has just occurred to you it may still be hazy and tentative -- the smallest interruption could cause you to loose it. (Though stopping to elaborate it could cause you to forget what you were saying about X!) Coming up with a good word or short phrase to summarize Y may not be easy -- you have to consider not just what is descriptive but what will be suggestive when the reader encounters the link to Y within X. In particular, should you name the link to Y to suggest the contents of Y or to show Y's relationship to X? Some systems (e.g. NoteCards) provide that links can have both a *type* (e.g. "idea") and a label (e.g. "subsume A in B"). Coming up with good names for both can impose even more load on an author struggling with an uncertain point. (One way to reduce this problem is for the authoring system to support immediate recording of the substance of the idea, with creation and labeling of the link and/or the node after the thought has been captured.)

Beyond that, one should also consider if one has provided sufficient links into Y before returning to work on X. It may be that there are other or better ways to link Y into the network of thoughts than just at the point in X where Y came to mind.

The problem of cognitive overhead is also apparent in the process of reading hypertext -- as was pointed out above (Section 3.2.1), reading hypertext tends to present the reader with a large number of choices about which links to follow and which to leave alone, and these choices engender a certain overhead of meta-level decision making, an overhead that is absent when the author has already made many of these choices for you. In the moment of encountering a link, how do you decide if it is worth the distraction to follow up on the side path? Does the label appearing in the link tell you enough to decide? This could be called "informational myopia", and the problem is that even if the system response time is instantaneous (which it almost never is) there is a definite distraction, or a cognitive loading, that happens just from pausing to consider if the side path should be pursued. This problem can be eased by (a) having the cross referenced node appear *very* rapidly (which is the approach of KMS), by (b) providing an instantaneous one to three line explanation of the side reference in a pop up window (which is the approach of Intermedia), and by (c) having a graphical browser which shows the local subnetwork into which the link leads.

These problems are not new with hypertext, nor are they a spurious artifact of computer-based thinking. People who think for a living -- writers, scientists, artists, designers, etc. -- must contend with the fact that the brain can create ideas faster than the hand can write them or the mouth can speak them. There is always a balance between refining the current idea, returning to a previous idea to refine it, and attending to any of the vague "proto-ideas" which are hovering at the edge of consciousness. Hypertext simply offers a sufficiently sophisticated "pencil" to begin to engage with the richness, variety, and interrelatedness of creative thought. This has advantages when this richness is needed and drawbacks when it is not.

To summarize, then, the problems with hypertext are:

- disorientation: the tendency to lose one's sense of location and direction in a non-linear document; and
- cognitive overhead: the additional effort and concentration necessary to maintain several tasks or trails at one time.

These problems may be at least partially resolvable through improvements in performance and interface design of hypertext systems, and through research on information filtering techniques.

6 Some Design and Implementation Issues

In this section we review some of the issues that must be resolved in building a hypertext system. This section could have been titled "Some Features of Hypertext Systems", but we feel that hypertext is still too undeveloped and unexplored to describe these issues as features of something that exists. Rather, these are issues in the design of something that is still taking form, and the reader is invited to participate in this great experiment by considering the pros and cons of the issues presented here, and to add other issues to the list.

6.1 Views of Links within Nodes

The standard format for viewing a hypertext node is that its window contains icons for its links, which links cause new windows to open when activated. It should be noted, however, that some applications call for a different treatment of linking. In VLSI CAD, for example, each "layer" of a design can be represented by a single node which contains the elements of the layer below as subnodes represented as icons or smaller windows, a kind of "Chinese Boxes" view. Clicking on a part causes that part to become the new top level node, revealing more detailed design information and the next lower layer of the design. In this approach, the notion of logical containment is mapped much more strictly into the graphical containment among the windows. Such a two-dimensional layout is also used in the hypertext programming language Boxer [diSe85, diSe86], and is one of the views available in WE [Smit86c].

6.2 Iconization of Nodes

In window-based hypertext systems, there are two ways to remove an open hypertext window from the screen: making it go away completely, and setting it aside by reducing it to a small icon which takes up much less screen space but still hints at the content of the node. The latter option can be quite complex, depending on (a) the speed with which one requires that a window "iconize" and reopen, and (b) the cues that are offered about the content of a window by its icon. Speed is rarely a problem in window and hypertext systems, but there are many complex issues in abstracting the contents of a node for its icon. How much of the node's name should be squeezed into the icon? Can shape, size, color, etc. be used as reminders of the full-sized window, and thus indirectly of the node's content? Can the appearance of the full sized window be reduced in ways that are useful for recognition? If there are many icons on the screen, can they be arranged in ways that still allow convenient recovery of a recently open window? Can ways be found for the user to quickly customize a personal icon that will be more memorable by virtue of being hand made?

Once a window has been set aside, there are two aspects to its retrieval: "Can the icon be located?" and "Can it be recognized?" The former issue is most directly related to the general issue of retrieval and indexing, and was taken up in Section 3.1.1. Here we are dealing primarily with the recognition issue.

6.3 Special Hypertext Editors

Most hypertext systems to date have opted to use an existing screen-based text editor for editing the contents of hypertext nodes. This is a reasonable approach, since good editors are hard to build, and hypertext nodes are, after all, just small documents. However, there are some demands for processing hypertext nodes which are unmet by some of these editors.

Many hypertext systems invoke a new editor process for each new window. This is a natural enough approach when implementing hypertext on top of a window system, where one rarely needs more than a few editor processes around, but it is very inefficient for several reasons. One is speed of node creation -- a hypertext window must appear within a couple of seconds of being requested, and many full service editors (e.g. EMACS) take much too long. Another problem is the use of scarce system resources. Few window systems are designed to support 25 editor processes simultaneously, yet that is a common circumstance for an active hypertext user. This is particularly a problem in UNIX and similar operating systems in which process are relatively “heavy” and consume significant system resources.

A more sensible approach is to have a single specialized editor which can regard the set of open hypertext windows on the screen as its buffers, using the mouse to indicate which is the user's current window of focus. This approach not only provides better performance, but allows ease of *inter-window* communication necessary for such editing features(33) as:

- 1. text fragments (and their links) can be copied or moved between hypertext windows;
- 2. a window can be split into several new nodes;
- 3. several hypertext nodes can be merged into a single node; and
- 4. advanced features such as composite nodes (see Section 3.2.6) and template nodes can be manipulated appropriately.

The single editor approach probably requires that such an editor be built from scratch, unfortunately, because the details of managing “Which is the current buffer?” and “Where are the cursor locations within the various buffers?” are usually written at a very low level in the editor code. Changing them to use the mouse for tracking among windows scattered around the screen may require a complete redesign of these lower levels of code.

6.4 Methods of Network Storage

Hypertext systems must have some way of permanently storing the nodes and links of the user's networks. The most common forms of storage are file systems and data bases.

There are two obvious ways to implement nodes in a standard file system: making each node be a file (as in PlaneText), and putting all the nodes in one big file (as in NoteCards). One advantage of having each node be a file is that each node is individually accessible without locking out the other nodes from other users -- the big disadvantage of putting all of the nodes into a single file is that there cannot be multiple concurrent users of the same hyperspace (i.e. collection of hypertext nodes) unless a more fine-grained version of such standard file system services as file locking (write protection), versioning, and security are implemented by the hypertext system. Another problem with a single file for all nodes is that if some of the nodes are large, e.g. whole documents, then the file may become unworkably large (depending on file size limits in the operating system). Conversely, the disadvantage of having each node be a file is that the speed of node creation and access is limited by the speed of file creation and access in the operating system, which in some systems may be a substantial cost.

Another advantage of each node being a file is that the hypertext system merges smoothly with the existing file system and utilities (e.g. mail and news), whereas nodes stored in a single file require extra processing to merge into the file system.

The solution to these tradeoffs may be a kind of file system designed to expedite small file opening and access. Alternatively, one could design the hypertext system to quietly open all files referenced by links in a newly opened node, without showing the user, so that that overhead could occur while the user is first scanning the contents of the node. A similar scheme can be used to prepare new node windows for display without actually displaying them -- until they are called for.

Another, increasingly popular approach is to build the hypertext system upon a database, for example a relational database. This approach has several advantages, the main one being that database technology is designed precisely to address the problems of using files for permanent storage. Such mechanisms as versioning, locking, transactions, multi-user shared access, and queries are commonly included in commercially available database systems. The main disadvantage of relational databases is that network structure is not naturally represented as sets of relational tables, making such common queries as “give me all the in-pointing node/link pairs to node X” computationally expensive. Intermedia and gIBIS are based on relational database servers.

Of course, the optimum solution is to design a *network server* that knows about nodes and links and is optimised for the kind of “spreading activation” queries that would be common in hypertext networks if they were supported. Neptune's HAM is the first implementation of such a custom approach to the storage of hypertext networks, although Nelson's Xanadu system addresses many of the ultimate large scale design issues.

7 Summary

This paper has reviewed existing hypertext systems, the advantages and disadvantages of hypertext, and some of the top level design issues of building hypertext systems. It has been my intention to give the reader a clear sense of what hypertext is, what its strengths and weaknesses are, and what it can be used for. But also something more: that you come away from this paper excited, eager to try using hypertext for yourself, and with the sense that you are there at the beginning of something big, something like the invention of the wheel, but something that still has enough rough edges that no one is really sure that it will fulfill its promise.

To that end I mention one more book that might be considered to be in the literature on hypertext. *Neuromancer* [Gibs84] is a novel about a time in the distant future when the ultimate computer interface has been perfected: one simply plugs one's brain into the machine and experiences the computer data directly as perceptual entities. Other computers look like boxes floating in 3-dimensional space, and passwords appear as various kinds of doors and locks. The user is completely immersed in a virtual world, the “operating system”, and can move around and take different forms simply by willing it.(34)

This seems to be the ultimate hypertext system. The basic idea of hypertext, after all, is that ideas correspond to perceptual objects, and one manipulates ideas and their relationships by directly manipulating windows and icons. Current technology limits the representation of these objects to static boxes on a CRT screen, but it is easy to predict that animation, color, 3-D displays, sound, etc. -- in short, Nelson's *hypermedia* -- will keep moving in the direction of making the display more active and realistic, and hence the “data” represented richer and more detailed, and making the input more natural and direct. Thus, hypertext, far from being an end in itself, is just a crude first step toward the time when the computer is a direct and powerful extension of the human mind, just as Vannevar Bush envisioned when he introduced his Memex four decades ago.

Acknowledgements

I wish to thank Les Belady, Bill Curtis, Susan Gerhart, Raymonde Guindon, Eric Gullichsen, Frank Halasz, Peter Marks, Don Petersen, Andy van Dam, and Doug Engelbart for their thoughtful reading of previous drafts, and for their encouragement.

Author's Current Address (as of July 1991):

E. Jeffrey Conklin
Corporate Memory Systems, Inc.
3500 West Balcones Center Drive

Austin, TX 78759

End Notes

(1) *Documentation* is the unexecutable English text which explains the logic of the program which it accompanies.

(2) By “workstation” I simply mean any computer that can be characterized as having a high speed processor, high-resolution display, powerful graphics support, and distributed on a network.

(3) While this paper seeks to establish the criterion of machine supported links as the primary criterion of hypertext, this is by no means an accepted definition. Therefore this paper will also review and discuss some systems which do not have this full functionality, e.g. which have a weaker notion of links.

(4) Note that there are two classes of icons: those that function as place holders for windows that have been temporarily put aside, and those within windows that represent links to other nodes. (5) A “string” is a series of alphabetic and numeric characters of any length, e.g. “listening” or “G00274”

(6) They also introduced a five-key handset — a one-handed keyboard — by which alphanumeric text was entered by “chording” the 5 keys; although it is slower than two-handed typing, it has a considerable advantage for short commands when used with a mouse in the other hand.

(7) Segments were limited to 2000 characters in length, but according to the developers this was never a problem.

(8) For details contact Ted Nelson at 8480 Fredericksburg, San Antonio, Texas, 78229, (512) 692-7346.

(9) gIBIS will only be available from the STP participants once they have decided to productize and market it.

(10) The author has written an outline processor that sits on top of EMACS, called TREEMACS. This "mock lisp" code is at this time proprietary to MCC/STP, but may at some point become a deliverable to the Participant companies

(11) For further information contact Rob Akscyn at Knowledge Systems, (412) 241-2240.

(12) Rather that use a mouse for pointing to links, the Hyperties system uses the convention of always having some link selected, and responding to the arrow keys by shifting the selection to the nearest link in the direction of the arrow. One of their empirical studies showed this to be a faster and easier technique for selecting arbitrary highlighted fields on the screen.

(13) For information contact Cognetics Corporation, Princeton Junction, N. J., (609) 799-5005.

(14) Halasz is now in the Software Technology Program at MCC.

(15) An action node contains Lisp code which gets evaluated when a link to the node is activated.

(15) An action node contains Lisp code which gets evaluated when a link to the node is activated.

(16) For further information contact Lori Naylor at Xerox, (818) 351-2351.

(17) For further information about IRIS or Intermedia call the IRIS group in Providence, R. I. (401) 863-2001.

(18) Note that this means that, unlike most hypertext systems, the destination end of a Neptune link is an iconic point in the text of the destination node, as opposed to the whole node itself; see Section 3.1.2.

(20) The use of an existing tree structuring mechanism limits any hypertext system to only being able to handle a single hierarchical structure. As we see in Section 3.1.2, single hierarchical organizations may be too limited for advanced applications.

(21) Efforts are currently underway to adopt EMACS as an alternative to the SunView editor.

(22) The remainder of this section on the essence of hypertext is technical and detailed, and may be richer fare than the casual reader wishes to read. If you are less interested in the technical issues that go into the design of hypertext systems, you may wish to skip this section and go on to Section 4.

(23) I suspect we computer users may have lost sight of how important it is to be able to have manual contact with our text when we are dealing with complex collections of ideas - 3X5 notecards are still best for some applications.

(24) Link deletion is more problematical. For example, what should the policy be for nodes which are stranded by havin all links deleted? Should they be placed in "node limbo" until the user decides to do with them?

(25) The distinction between points and regions generally does not apply in non-mouse-based systems.

(26) Note this is distinct from any class hierarchy that is used, in the object oriented programming style, to define types and subtypes of nodes in the hypertext system.

(27) There is controversy over the merits of "full text" search as opposed to the use of preselected keywords. Keyword retrieval is only as good as the skill and thoroughness of the person selecting the keywords, but there is evidence that full text search does not really deliver on its promise of finding all, and only those, relevant documents (see[Blai85]). In addition, full text search can be computationally prohibitive in large networks.

(28) Indeed, the fact that keywords can be precompiled into inverted files makes it even clearer that these are a kind of virtual link.

(29) As of this writing region to region linking is only supported in graphical applications in Intermedia - only point sources and destinations are provided for text.

(30) Of course, it could be argued that the need for cross-hierarchical links simply means that the correct hierarchical structuring of the material has not been discovered.

(30) Of course, it could be argued that the need for cross-hierarchical links simply means that the correct hierarchical structuring of the material has not been discovered.

(31) This description of semantic nets is only deep enough to draw the analogy to hypertext. Readers interested in a deeper and more precise description of semantic nets are referred to Chapter 3 in [Arbi87].

(32) Actually, the term “word processor” is quite misleading. Most such tools accept input only at the character level, and manipulate characters, words, sentences, paragraphs with equal facility, so it is text, not words, that is being processed. But do they process these units? Processing would imply some additional work being done by the computer, such as changing the verb form if the subject was changed from singular to plural, or real-time spelling and grammar correction. So really we should return to the original term for these tools: *text editors* .

(33) Of course, it may also be that there are editors that are so tightly integrated with their operating system that they already provide all of these features. In this case, the need for a single editor process is much less important.

(34) A simpler theme was expressed in an earlier story by Verner Vinge [Vinge81]. It too is a very good story.

MCC TR STP-356-86, Rev. 2
Jeff Conklin
Software Technology Program
Classification: Non Proprietary
December 3, 1987

References

[Aksc84b] Akscyn, R. M., McCracken, D.L.: ZOG and the USS CARL VINSON: Lessons in System Development, Carnegie-Mellon Technical Report CMU-CS-84-127. In Proceedings of the First IFIP Conference on Human-Computer Interaction; Interact '84; London, England. 1984.

[Arbi87] Arbib, Michael , Conklin, Jeffrey, Hill, Jane: From Schema Theory to Language, Oxford Universty Press. 1987

[Blai85] Blair, D. C., Maron, M. E.: An Evaluation of Retrieval Effectiveness for a Full-Text Document-Retrieval System. Communications of the ACM 28, 3, 289-299. March 1985.

[Bolt79] Bolt, Richard A.: Spatial Data-Management, Arhcitecture Machine Group, Massachusetts Insitute of Technology. 1979.

[Brac77] Brachman, R.: What's in a Concept: Structural Foundations for Semantic Networks; Int. J. Man-Machine Studies 9, 127-152. 1977.

[Brow82] Brown, J.S.: Notes Concerning Desired Functionality Issues and Philosophy for an AuthoringLand, Xerox PARC CIS working paper, 1982.

[Bush45] Bush, V.: As We May Think. The Atlantic Monthly, July 1945: 101-108. Reprinted in Adele Goldberg (editor), A History of Personal Workstations, ACM Press, New York, 237-247. 1988.

[Conk85] Conklin, Jeff, Richter, Charles: Support for Exploratory Design. Proceedings of the AIAA/ACM/NASA/IEEE Computers in Aerospace V Conference, American Institute of Aeronautics and Astonautics. Also, MCC TR # STP-117-85. October 1985.

[Conk86] Conklin, Jeff: A Theory and Tool for Coordination of Design Conversations. MCC TR Number STP-236-86, MCC, Austin, Texas. 1986.

[Deli86a] Delisle, N., Schwartz, M.: Neptune: A Hypertext System for CAD Applications. Proceedings of ACM SIGMOD International Conference on the Management of Data. Washington, D.C. 132-142. January 1986.

[Deli86b] Delisle, N., Schwartz, M.: Contexts—A Partitioning Concept for Hypertext. Proceedings of the Conference on Computer-Supported Cooperative Work, 147-152. 1986.

[diSe85] diSessa, A.: A Principled Design for an Integrated Computational Environment, Human-Computer Interaction, Volume 1, Lawrence Erlbaum, 1-47. 1985.

[diSe86] diSessa, Andrea A., Abelson, H.: Boxer: A Reconstructable Computational Medium, Communications of the ACM 29(9), 859-868. 1986.

[Done78] Donelson, W.C.: Spatial Management of Information. ACM SIGGRAPH. 1978.

[Enge63] Engelbart, D.C.: A Conceptual Framework for the Augmentation of Man's Intellect. In: P.W. Howerton & D.C. Weeks (eds.) Vistas in Information Handling, Volume 1. London: Spartan Books. 1963.

[Enge68] Engelbart, D.C., English, W.K: A Research Center for Augmenting Human Intellect. Proceedings of 1968 Fall Joint Computer Conference, 33 Part 1, Montvale, N.J.: AFIPS Press, 395-410. 1968.

[Fost85] Foster, E.: Outliners: A New Way of Thinking, Personal Computing, 74. May 1985.

[Garr86a] Garrett, L.N., Smith, K.E., Meyrowitz, N.: Intermedia: Issues, Strategies, and Tactics in the Design of a Hypermedia Document System. Proceedings of the Conference on Computer-Supported Cooperative Work, Austin, Texas, 163-174. December 3-5, 1986.

[Gibs84] Gibson, A.: Neuromance, Ace Science Fiction. 1984.

[Gold80] Goldstein, I.P., Bobrow, D.G.: Descriptions for a Programming Environment. Proceedings of the First Conference AAAI. August 1980.

[Gold84] Goldstein, I., Bobrow, D.: A Layered Approach to Software Design, In D. Barstow, H. Shrobe, and E. Sandewall (eds.) Interactive Programming Environments. McGraw-Hill: 387-413. 1984.

[Good87] Goodman, D.: The Complete HyperCard Handbook. Bantam Books, New York. New York. 1987.

[Gull86a] Gullichsen, E., D'Souza, D., Lincoln, P., Casey, T.: The PlaneText Book. MCC Tech. Rep. STP-333-86(P). 1986.

[Hala86] Halasz, F.: NoteCards: An Experimental Environment for Authoring and Idea Processing, Xerox PARC. 1986.

[Hala87a] Halasz, F. G., Moran, T., Trigg, R.H.: NoteCards in a Nutshell. Proc. ACM CHI+GI'87 Human Factors in Computing Systems and Graphics Interface Conf., Toronto, Canada, 45-52. April 5-9, 1987.

[Hero80] Herot, C.F.: Spatial Management of Data. ACM Trans. Database Systems. Volume 5, Number 4, 493-514. December 1980.

[Hers85] Hershey, W.: Idea Processors, BYTE, 337. June 1985.

[Hers84] Hershey, William: ThinkTank. BYTE, 189. May 1984.

[Horo86] Horowitz, Ellis, Williamson, Ronald: SODOS: A Software Documentation Support Environment - Its Definition. IEEE Transactions on Software Engineering. Volume SE-12, Number 8. August 1986.

[Jaco84] Jacobs, M.A., Murray, D.J.: Filevision, Telos Software Products, Santa Monica, California. 1984.
Jones, W.P., Dumais, S.T.: The Spatial Metaphor for User Interfaces:

[Jone86b] Experimental Tests of Reference by Location Versus Name. In ACM: TOOLS, Volume 4, Number 1, 42-61. January 1986. Jones, W.P.: The Memory Extender Personal Filing System.

[Jone86c] HI-138-85, MCC, Austin, Texas. January 1986.

[Karl83] Karlgren, H., Walker, D.E.: The Polytext System-A New Design for a Text Retrieval System. In “Questions & Answers”, ed. Fernec Kiefer, 273-294. D. Reidel Publishing. 1983.

[Kay77b] Kay, A., Goldberg, A.: Personal Dynamic Media, Computer, 31-41. March 1977.

[Kimu86] Kimura, G.D.: A Structure Editor for Abstract Document Objects. IEEE Trans. Software Eng., Volume SE-12, Number 3. 417-435. March 1986.

[Know] Knowledge Systems Incorporated, 4758 Old William Penn Highway, Murrysville, PA 15668, (412) 241-2240.

[Knut84] Knuth, D.E.: Literate Programming. The Computer Journal, 27(2): 97-111. 1984.

[Lars87] Larson, Neil: MaxThink Journal, MaxThink, Piedmont, California, Volume 2:3. January 1987.

[Lewi86] Lewis, D.J.: A Good 'Shareware' Outline Processor. IEEE Software, 59. September 1986.

[Lowe85] Lowe, David G.: Cooperative Structuring of Information: the Representation of Reasoning and Debate. Int. J. Man-Machine Studies, Volume 23. 23:97-111. 1985.

[Malo87a] Malone, T. W., et al.: Intelligent Information Sharing Systems, Communications of the ACM, Volume 30, Number 5, 390-402. May 1987.

[Mant79] Mantei, M.M., McCracken, D.: Issue Analysis with ZOG, A Highly Interactive Man-machine Interface. In the Proceedings of the First International Symposium on Policy Analysis and Information Systems. 1979.

[McCr84] McCracken, D., Akscyn, R.: Experience with the ZOG Human-Computer Interface System. International Journal of Man-Machine Studies 21, 293-310. 1984.

[Negr79] Negroponte, N.: Books Without Pages. IEEE Int. Conference on Communications IV, 1-8. 1979.

[Nels80] Nelson, T.: Replacing the Printed Word: A Complete Literary System. Proceedings of IFIP Congress 1980, North-Holland, 1013-1023. 1980.

[Nels81] Nelson, T.H.: Literary Machines: The Report on, and of, Project Xanadu, Concerning Word Processing, Electronic Publishing, Hypertext, Thinkertoys, Tomorrow's Intellectual Revolution, and Certain other Topics Including Knowledge, Education and Freedom, available from the author ($15, 8480 Fredericksburg #138, San Antonio, Texas 78229). Also being published, along with Dream Machines, by Microsoft Press. 1981.

[OWLI87] Guide: Owl International Inc. 14218 NE 21st St. Bellvue,WA. 98007. 1987.

[Pass84] Passini, Romedi: Wayfinding in Architecture, Van Nostrand Reinhold Company Inc., Environmental Design Series, Volume 4. 1984.

[Pitm85] Pitman, K.M.: CREF: An Editing Facility for Managing Structured Text, A.I. Memo Number 829, M.I.T. A.I. Laboratory, Cambridge, Massachusetts. February 1985.

[Ritt73] Rittel, H., Webber, M.: Dilemmas in a General Theory of Planning, Policy Sciences, Volume 4. 1973.

[Robe81] Robertson, G., McCracken, D., Newell, A.: The ZOG Approach to Man-Machine Communication. International Journal of Man-Machine Studies, 14, 461- 488. 1981.

[Scha85] Schatz, B.R.: Telesopy. Technical Report Bellcore TM-ARH-002487, Bell Communications Research, 445 South Street, Morristown, NJ. 07960-1910. 1985.

[Shas85d] Shasha, D.: Netbook - a Data Model to Support Knowledge Exploration. In Proceedings of the Eleventh Conference on Very Large Databases, Stockholm. August 1985.

[Shas86] Shasha, D.: When Does Non-Linear Text Help? Expert Database Systems, Proceedings of the First International Conference, 109-121. April 1986.

[Shne86] Shneiderman, A., Morariu, J: The Interactive Encyclopedia System (TIES), Department of Computer Science, University of Maryland, College Park, Maryland, 20742. June 1986.

[Simo69] Simon, H.A.: The Sciences of the Artificial. Cambridge, Massachusetts: The MIT Press. 1969.

[Smit86c] Smith, J., Weiss, S. F., Ferguson, G. J., Bolter, J. D., Lansman, M., Beard, D. V.: WE: A Writing Environment for Professionals, University of North Carolina at Chapel Hill, Department of Computer Science Technical Report 86-025. 1986.

[Spez86] Spezzano, Charles: Unconventional Outliners. PC World, 168. March 1986.

[Stal81] Stallman, Richard M.: EMACS: The Extensible, Customizable Self-Documenting Display Editor. Proc. ACM SIGPLAN/SIGOA Conference on Text Manipulation, Portland, Oregon, 147-156. June 1981.

[Trig83] Trigg, R.H.: A Network-Based Approach to Text Handling for the Online Scientific Community, Ph.D. Thesis, University of Maryland. 1983.

[Trig86b] Trigg, R.H., Suchman, L., Halasz, F.G.: Supporting Collaboration in NoteCards. Computer-Supported Cooperative Work; CSCW 86 Proceedings, Austin, Texas. 153 - 162. December 3-5, 1986.

[Ving81] Vinge, Vernor: True Names, Dell Books. 1981.

[Walk85] Walker, J.H.: The Document Examiner, SIGGRAPH Video Review, Edited Compilation from CHI'85: Human Factors in Computing System. l985.

[Weye82a] Weyer, S. A.: The Design of a Dynamic Book for Information Search. In Int. J. of Man-Machine Studies, Volume 17, 87-107. 1982.

[Wino86b] Winograd, T., Flores, F.: Understanding Computers and Cognition: A New Foundation for Design. Ablex Publishing Corp. 1986.

[Wood86] Woodhull, Albert, S.: Kamas. BYTE, 241. April 1986.

[Wood75] Woods, William A.: What's in a link: Foundations for semantic networks. In D.G. Bobrow and A. Collins (eds.) 'Representation and Understanding: Studies in Cognitive Science, New York: Academic Press. 1975.

[Yank85] Yankelovich, N., Meyrowitz, N., van Dam, A.: Reading and Writing the Electronic Book. Computer, 18(10). 15-30. 1985.

[Yank] Yankelovich, Nicole: Hypermedia Bibliography. Unpublished report from the Institute for Research in Information and Scholarship (IRIS), Box 1946, Brown University, Providence, Rhode Island 02912, or via electronic mail “ny@iris.bitnet” or “ny%iris.brown.edu@relay.cs.net”.

[Youn86] Young, Jeffrey, S.: Hypermedia. In Macworld. March 1986.