

Hypertext '87

TR88-013

March 1988

Hypertext Planning Committee

John B. Smith, UNC

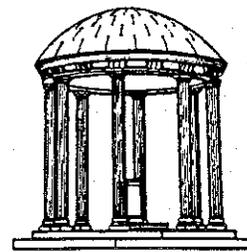
Frank Halasz, MCC

Nicole Yankelovich, Brown University

Mayer Schwartz, Tektronix

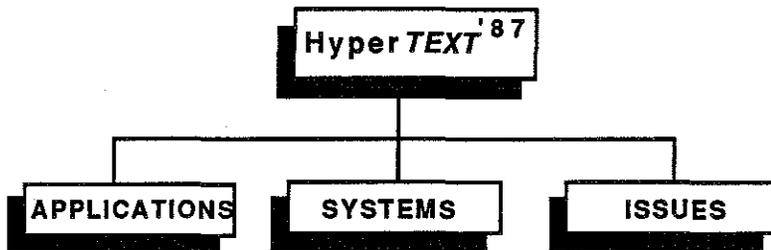
Stephen F. Weiss, UNC

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175



Copyright 1987 Hypertext Planning Committee.
A TextLab Report

All rights reserved.



HYPertext '87

PAPERS

NOVEMBER 13-15, 1987

THE UNIVERSITY OF NORTH CAROLINA
CHAPEL HILL, NORTH CAROLINA

Co-sponsored by ACM, CS of the IEEE, and UNC Department of Computer Science

With support from ONR, MCC, and NSF

In cooperation with ACM SIGCHI

Copyright 1987 by the Planning Committee for Hypertext '87.

**Copying for scholarly purposes is permitted provided that the copies are not
made or distributed for direct commercial advantage.**

These papers may be abstracted with credit to the source.

P A P E R S

FOREWORD

All for One and One for All	V
<i>Theodor H. Nelson, Project Xanadu</i>	

SESSION 1A: SYSTEMS I

KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations	1
<i>Robert Akscyn, Donald McCracken and Elise Yoder, Knowledge Systems Incorporated</i>	
HAM: A General - Purpose Hypertext Abstract Machine	21
<i>Brad Campbell and Joseph M. Goodman, Tektronix, Incorporated</i>	
Turning Ideas into Products: The Guide System	33
<i>P.J. Brown, Office Workstations Limited and The University of Canterbury</i>	

SESSION 1B: APPLICATIONS IN THE HUMANITIES AND MEDICINE

Hypertext and Creative Writing	41
<i>Jay David Bolter and Michael Joyce, University of North Carolina at Chapel Hill</i>	
From the Old to the New: Intergrating Hypertext into Traditional Scholarship	51
<i>Gregory Crane, Harvard University</i>	
Searching for Information in a Hypertext Medical Handbook	57
<i>Mark E. Frisse, M.D., Washington University School of Medicine</i>	
Hypertext and Pluralism: From Lineal to Non-lineal Thinking	67
<i>William O. Beeman, Kenneth T. Anderson, Gail Bader, James Larkin, Anne P. McClard, Patrick McQuillan and Mark Shields, Brown University</i>	

SESSION 2A: EXPERIENCES AND WRITING

Hypertext Habitats: Experiences of Writers in NoteCards	89
<i>Randall H. Trigg and Peggy M. Irish, Xerox Palo Alto Research Center</i>	
Comprehending Non-Linear Text: The Role of Discourse Cues and Reading Strategies	109
<i>David Charney, The Pennsylvania State University</i>	
The Notes Program: A Hypertext Application for Writing from Source Texts	121
<i>Christine Neuwirth, David Kaufer, Rick Chimera and Terilyn Gillespie, Carnegie Mellon University</i>	

SESSION 2 B: TRANSLATING TEXT INTO HYPERTEXT

Hypertext and the New Oxford English Dictionary	143
Darrell R. Raymond and Frank Wm. Tompa, <i>University of Waterloo</i>	
Content Oriented Relations Between Text Units – A Structural Model for Hypertexts	155
Rainer Hammwöhner and Ulrich Thiel, <i>University of Constance</i>	
SuperBook: An Automatic Tool for Information Exploration – Hypertext?	175
Joel R. Remde, Louis M. Gomez and Thomas K. Landauer, <i>Bell Communications Research</i>	

INVITED PANEL ON SYSTEMS

User Interface Design for the Hyperties Electronic Encyclopedia	189
Ben Schneiderman, <i>The University of Maryland</i>	
A Hypertext Writing Environment and its Cognitive Basis	195
John B. Smith, Stephen F. Weiss and Gordon J. Ferguson, <i>University of North Carolina at Chapel Hill</i>	

SESSION 3 A: ARGUMENTATION

Constraint – Based Hypertext for Argumentation	215
Paul Smolensky, Brigham Bell, Barbara Fox, Roger King and Clayton Lewis, <i>University of Colorado at Boulder</i>	
glBIS: A Hypertext Tool for Team Design Deliberation	247
Jeff Conklin and Michael L. Begeman, <i>Microelectronics and Computer Technology Corporation (MCC)</i>	
Exploring Representation Problems Using Hypertext	253
Catherine C. Marshall, <i>Xerox Special Information Systems</i>	

SESSION 3 B: SYSTEMS II

Thoth-II: Hypertext with Explicit Semantics	269
Georger H. Collier, <i>Bell Communications Research</i>	
The Architecture of Static Hypertexts	291
Tim Oren, <i>Apple Computer, Incorporated</i>	
Document Examiner: Delivery Interface for Hypertext Documents	307
Janet H. Walker, <i>Symbolics Incorporated</i>	

SESSION 4 A: ISSUES

The Hype in Hypertext: A Critique	325
<i>Jef Raskin, Information Appliance</i>	
Relationally Encoded Links and the Rhetoric of Hypertext	331
<i>George P. Landow, Brown University</i>	
Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems	345
<i>Frank G. Halasz, Microelectronics and Computing Technology Corporation (MCC)</i>	
*Developing and Distributing Hypertext Tools: Legal Inputs and Parameters	367
<i>Henry W. Jones, III, Esq., Morris, Manning and Martin</i>	

SESSION 4 B: SOFTWARE

Abstraction Mechanisms in Hypertext	375
<i>Pankaj K. Garg, University of Southern California</i>	
Manipulating Source Code in Dynamic Design	397
<i>James Bigelow and Victor Riley, Tektronix, Incorporated</i>	
On Designing Intelligent Hypertext Systems for Information Management in Software Engineering	409
<i>Pankaj K. Garg and Walt Scacchi, University of Southern California</i>	
AUTHOR'S INDEX	433

**Paper not presented.*



F O R E W O R D

All for One and One for All

Theodor H. Nelson

Invited Talk

Hypertext, or non-sequential writing with free user movement along links, is a simple and obvious idea. It is merely the electrification of literary connections as we already know them.

It took a great length of time to interest people in hypertext (1,2). However, researchers everywhere are becoming interested in the idea, and indeed we now see a heartening proliferation of hypertext systems in both prototype and commercial versions (3).

However, we stand in danger of recapitulating the ongoing disaster of the computer world: the incompatibilities, not just of equipment and storage media, but of software and data formats. These have made computers a terrible mess for users everywhere (4).

Hypertext offers vast new possibilities of access and clarity of information for everyone. If the availability of hypertext is to be conditional and incompatible across systems, this great hope will be dashed.

All for One: Everything for Any User

Everything should be available to everyone. Any user should be able to follow origins and links of material across boundaries of documents, servers and networks; and across boundaries of individual implementations.

One for All: A Single All-Encompassing Space for All Documents

There should be a unified environment available to everyone, providing access to this whole space.

This also means some single unifying mechanism or approach to support all different types of links, all different types of hypertext, and the appropriate labelling of typed data of any kind as it is served out to users.

Project Xanadu has developed a unique software mechanism to organize storage and network delivery of hypermedia of all types. This consists of a uniform servicing mechanism for documents, links, successor versions and material virtually copied from elsewhere. It maintains virtual documents under constant editorial change, and holds links in place on spans of bytes regardless of such changes. The Xanadu* storage and delivery program may be understood as a readdressing system for managing very large bodies of permanently stored material (whether on erasable disk, CD-ROM, WORM drives or other media).

Xanadu addressing methods may be used by anyone, and its address space may include nodes anywhere, owned and controlled by anyone, with appropriate assignment of compatible addresses. Thus we freely offer this addressing scheme to all users, with no arrangements necessary.

All for One: The Possibility of Uniform Service

This offers the possibility of uniform service to any client from all active servers of permanently-stored materials, under a single service discipline, with a single connection to a cooperative port.

1 for All: The Meaning of the Leading Tumbler "1"

The Xanadu addressing scheme (invented by Mark S. Miller, developed for this purpose by Mark S. Miller and Roger Gregory, and detailed in (2) and (5)) employs multipart *tumblers* for the addressing of servers, accounts, documents, versions and elements. A tumbler is a vector beginning with 1 and having up to four fields. Each field (node, user, document, element) is a succession of integers separated by points. Subnodes, subusers, subdocuments are referenced by additional integers within a field. Fields are separated by zero integer fields, which have arithmetic significance.

The leading digit "1" in tumbler addressing space has a special function: it represents the entire universe of documents, or docuverse. Its use permits a uniform arithmetic across the docuverse, and uniform reference to spans of materials. Metaphysically, the leading 1 refers to a mythical Ur-document from which all others are descended.

All for 1: The Meaning of the 1-1 Tumbler Span

Tumbler addressing permits us to refer not just to individual servers, accounts, documents and elements, but also to *spans* ranging from one byte or link up to the entire docuverse (not limited just to one document). Spans are referenced by tumbler arithmetic (detailed in (2) and (5)), designating a starting address tumbler and a difference tumbler, which specifies a span of the whole tumbler line.

Here again the number 1 has a mystical significance. The starting address of "1," and a tumbler difference of 1, stand for the entire docuverse. This furnishes a parsimonious way of referring to everything, especially in searches— when seeking links to anything from a given document, or from anything to a given document.

Such unity is often desirable, and here offers us the possibility of a single free world open to all, both as suppliers and as clients in tomorrow's hypertext world of knowledge and freedom.

BIBLIOGRAPHY

1. Theodor H. Nelson, "The Hypertext." Proceedings International Documentation Federation, 1965.
2. Theodor H. Nelson, *Literary Machines*, edition 87.1. Project Xanadu, 1987.
3. Jeff Conklin, "Hypertext: An Introduction and Survey," *Computer*, September 1987, 17-41.
4. Theodor H. Nelson, *Computer Lib*, second edition; Tempus Books (Microsoft Press), 1987. Note pages 20-23, "A Field of Rubble."
5. Theodor H. Nelson, "Immense Storage Management." *Byte*, January 1988 (forthcoming).

* "Xanadu" is a trade and service mark for hypertext and computer products and services offered by Project Xanadu, and licensed to XOC, Inc.

**However, the Xanadu program for maintaining these connections and inclusions through editorial operations is still proprietary.





Systems I

KMS: A Distributed Hypermedia System for Managing Knowledge In Organizations

Robert Akscyn, Donald McCracken, Elise Yoder

Knowledge Systems Incorporated
4750 Old William Penn Hwy, Murrysville, PA 15668

ABSTRACT

KMS is a commercial hypermedia system developed by Knowledge Systems for networks of heterogeneous workstations. It is designed to support organization-wide collaboration for a broad range of applications, such as electronic publishing, software engineering, project management, computer-aided design and on-line documentation. KMS is a successor to the ZOG system developed at Carnegie Mellon University from 1972 to 1985.

A KMS database consists of screen-sized WYSIWYG workspaces called frames that contain text, graphics and image items. Single items in frames can be linked to other frames. They may also be used to invoke programs. The database can be distributed across an indefinite number of file servers and be as large as available disk space permits. Independently developed KMS databases can be linked together.

The KMS user interface uses an extreme form of direct manipulation. A single browser/editor is used to traverse the database and manipulate its contents. Over 85% of the user's interaction is direct--a single point-and-click designates both object and operation. Running on Sun and Apollo workstations, KMS accesses and displays frames in less than one second, on average.

This paper describes KMS and how it addresses a number of hypermedia design issues.

INTRODUCTION

For the past six years, we have been developing a commercial hypermedia system (KMS) based on our previous research with the ZOG system at Carnegie Mellon University. This paper describes KMS and how it addresses a number of hypermedia design issues, particularly issues concerning what data model to use. Section 1 provides some historical background on ZOG and KMS. Section 2 gives an introductory description of KMS. Section 3 describes some hypermedia design issues and how KMS addresses them. Section 4 concludes by reiterating the importance of the KMS data model--how it permeates the overall design of the system.

1. BACKGROUND

We have been developing hypermedia systems for over a decade, first at Carnegie Mellon University with the ZOG Project, and now at Knowledge Systems with the commercial development of our Knowledge Management System (KMS).

We have been zealous users of hypermedia. While developing ZOG and KMS, we used them extensively for our work--logging over 10,000 person-hours as users, and creating over 50,000 frames (nodes). Throughout this period we have applied what we have learned to iterate the design of these systems, creating scores of intermediate versions.

Early ZOG efforts at CMU. Work on ZOG began at CMU in 1972. What we now call ZOG-1 was developed for a summer workshop for researchers in cognitive science. It allowed the participants to easily interact with one another's simulation programs by providing a uniform menu-selection interface. After the workshop, ZOG-1 was shelved because the technology used was inadequate (300 baud hard-copy terminals!). Work on ZOG was rekindled in 1975, after Allen Newell and George Robertson observed the PROMIS system at the University of Vermont. PROMIS was a menu system based on rapid-response touch-screen terminals, applied to the task of hospital management [Schu79]. Newell and Robertson were struck by the qualitative difference of the rapid-response PROMIS interface over traditional human-computer interfaces, and began an ONR-sponsored research project to study the general characteristics of large, rapid-response, menu-selection systems. From 1975 to 1980, the ZOG group developed a series of ZOG versions for PDP-10s, Vaxes, and even for an experimental multi-processor machine, C.mmp [Robe81b].

ZOG on the USS CARL VINSON. By 1980 we felt ZOG was sufficiently mature to be tested in the real world. So we embarked on a major ZOG application project--to build a computer-assisted management system for the Navy's newest nuclear-powered aircraft carrier, the USS CARL VINSON. This was a joint project between the ZOG Group at CMU and the officers of the CARL VINSON. The development phase of the project ended in March, 1983, when the CARL VINSON left on her first deployment with a distributed ZOG system running on a network of 28 PERQ workstations. ZOG supported four applications:

- On-line policy manual (Ship's Organization and Regulation Manual)
- Interactive task management system (for analyzing and tracking complex tasks)
- On-line maintenance manual with interface to videodisk (for weapons elevators)
- Interface to the AirPlan expert system (developed at CMU by McDermott, et.al.)

We continued to work with the crew of the CARL VINSON until the end of the ZOG project in December 1984. The project and some of the lessons we learned are described in [Newe81], [Aksc84b] and [Newe85].

Knowledge Systems and KMS. In 1981, at the request of Westinghouse, we formed a company (now called Knowledge Systems) to develop a commercial version of ZOG. Westinghouse was interested in applying ZOG technology to the problem of providing operators of nuclear power plants access to emergency operating procedures. This initial work led to our first commercial version of ZOG (called KMS) in 1983. Since then we have worked with a number of other organizations to apply KMS to various large-scale knowledge management tasks.

Applications we have explored. We have found ZOG and KMS to be useful in a surprising number of applications over the years. At Knowledge Systems we use KMS for almost everything we do with computers. Below, we list the applications we have explored. More information about these applications can be found in [Aksc84a], [McCr84], and [Newe81].

- Electronic publishing
- On-line technical manuals
- On-line instruction manuals
- On-line help for other software
- Project management
- Issue analysis
- On-line policy manuals
- Group presentations via large screen projectors
- Financial modelling
- User interface to videodisk-based materials
- User interface to expert systems
- Software engineering
- Computer-assisted foreign language translation
- Operating system shell

2. INTRODUCTION TO KMS

Our primary design goal for KMS is to create a general-purpose software environment that helps an organization manage its knowledge. We are concerned not only with the productivity of the individual, but also the productivity of groups--from small workgroups up to an entire organization. We are especially concerned about the problem of building and maintaining large databases, since this activity is often the principal bottleneck in many uses of computers.

We are shaping KMS to exploit what we believe will be the dominant architecture for organizational computing environments of the 1990's: wide-area networks of large-screen, diskless workstations. We believe networked workstations offer quantum leaps in productivity over what is possible with today's personal computers.

In this section, we give a brief overview of the two major components of KMS--its data model (how knowledge is represented in KMS), and its user interface. Additional details about KMS are woven into the following section on hypermedia design issues, where we describe how KMS addresses particular design issues.

KMS data model

A KMS database consists of a set of interlinked, screen-sized workspaces. These workspaces, called *frames*, may contain any arrangement of text, graphics, and image items. Each individual text item within a frame can be linked to any other frame. As with ZOG, text items may also activate programs. These programs may range from atomic KMS operations to lengthy KMS animations (written in the KMS Action Language), as well as conventional programs that normally run from the operating system.

Frame format. Strong conventions have evolved for the format of frames. These conventions are illustrated by the example frame in Figure 1.

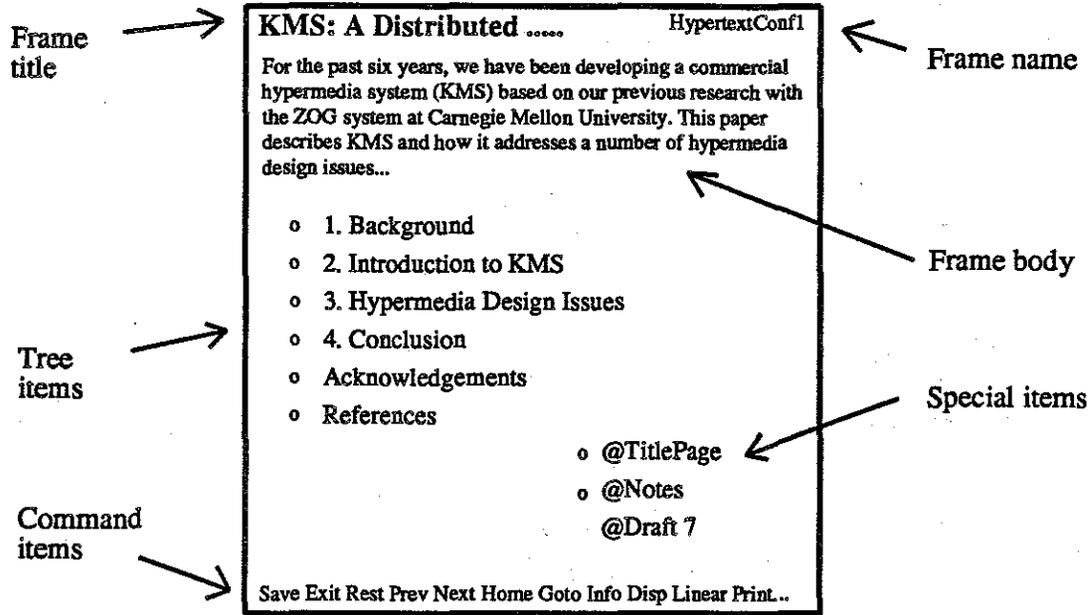


Figure 1: Example KMS frame

Every frame has a unique name. This *frame name* consists of two parts: an alphabetic part and a numerical part. The alphabetic part is the name of the *frameset* of which the frame is a member. (A frameset is the set of frames related to a specific topic as defined by the user, such as an individual document, software program, or project. Users are free to create a new frameset whenever they create a new frame.) The numerical part of the frame name is provided automatically. The frame name in the example above is "HypertextConf1".

The *frame title* is located in the upper left corner of the frame. It provides a short description of the knowledge contained in the frame. The frame title in the example is "KMS: A Distributed ...".

Next comes what we call the *frame body*. For upper-level frames in a hierarchy, the frame body is usually a short paragraph expanding on the topic described by the frame title. Lower-level frames usually contain more text, graphics, and images.

Below the frame body are text items called *tree items*, which are linked to lower-level frames. When a link is present, an item is displayed with a small circle to its left. The tree items in the example are:

- o 1. Background
- o 2. Introduction to KMS
- o 3. Hypermedia Design Issues
- o 4. Conclusion
- o Acknowledgements
- o References

On the lower right side of the frame are the *special items*, which begin with the character "@". Special items are used for miscellaneous purposes such as notes, comments, and document formatting keywords. As a result, special items have the connotation of being meta-level items. Special items are linked to other frames where appropriate. The special items in the example are:

- o @TitlePage
- o @Notes
- o @Draft 7

At the bottom of the KMS window (not actually part of the frame) is a customizable command menu containing *command items*. The default menu shown here contains 19 items. These items are used to invoke programs, from simple KMS operations to complex external programs. Invoking programs is discussed further in the following section on the "KMS User Interface."

Linked frames. KMS permits a frame to have an unlimited number of linked items, each of which may be linked to any frame (including itself). This flexibility permits KMS databases to have any structure the users desire, even a 'bowl of spaghetti' structure. In practice, however, KMS databases usually have a hierarchical backbone. This backbone is embellished with meta-level constructs in the form of *special items* such as user comments, formatting instructions, and cross-reference links. The use of hierarchy as the principal organizing paradigm is a strong factor in helping KMS users remain oriented.

Figure 2 illustrates a fragment of a KMS database. In this example we show part of the hierarchy of frames representing this paper.

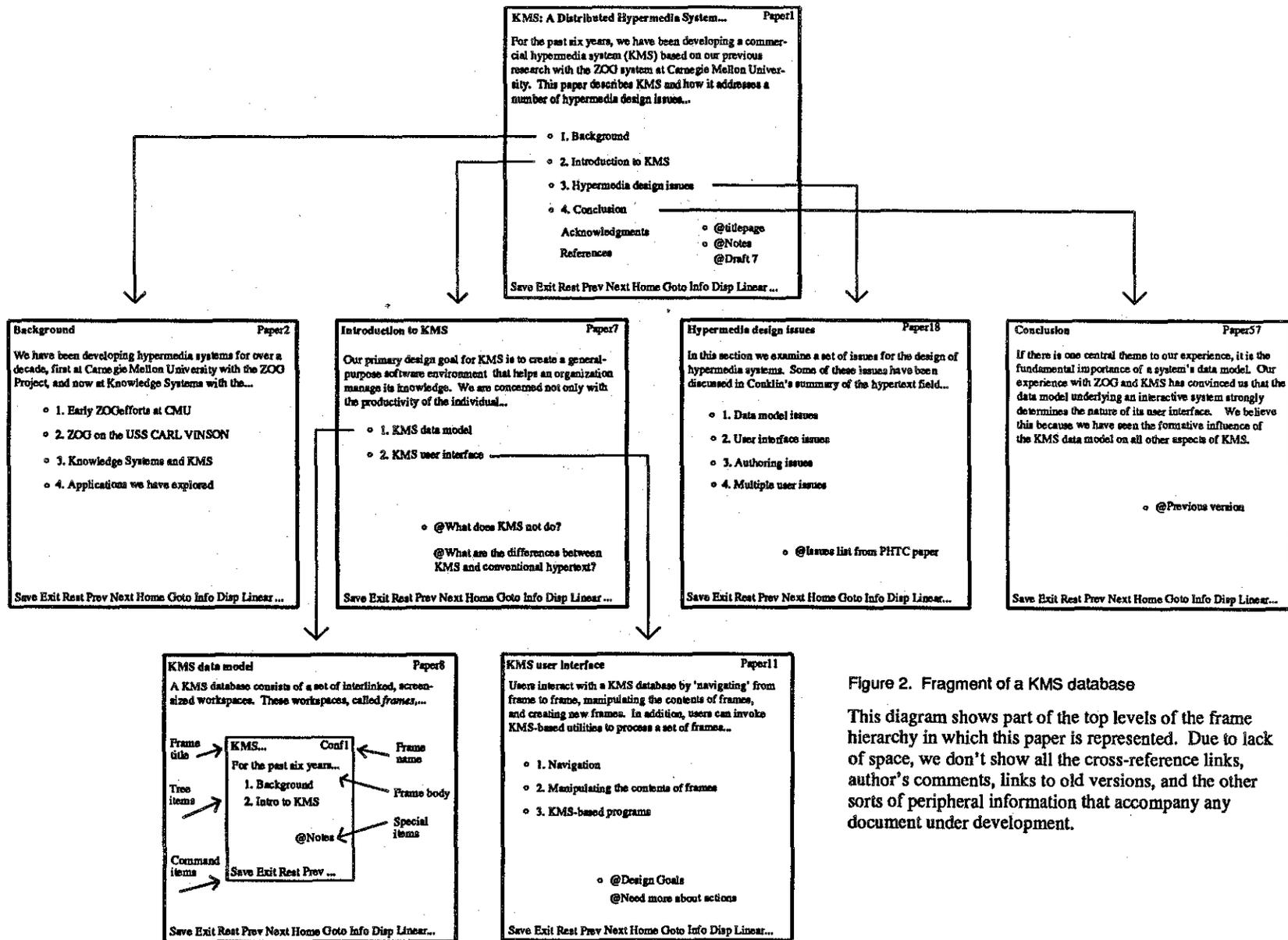


Figure 2. Fragment of a KMS database

This diagram shows part of the top levels of the frame hierarchy in which this paper is represented. Due to lack of space, we don't show all the cross-reference links, author's comments, links to old versions, and the other sorts of peripheral information that accompany any document under development.

KMS User Interface

Users interact with a KMS database by 'navigating' from frame to frame, manipulating the contents of frames, creating new frames, and invoking programs.

Navigation. The central metaphor in KMS is that the database is like a universe of connected spaces through which users rapidly travel, like pilots navigating spacecraft in the real universe. Users navigate from frame to frame by pointing the mouse cursor at an item linked to another frame and clicking one of the mouse buttons (KMS uses a 3-button mouse). KMS accesses the designated frame and displays it in the same window in less than one second, on average. Thus, KMS is replacing the currently displayed frame as though the user had physically travelled to a new location in the real universe.

Manipulating the contents of frames. A user can directly manipulate the contents of a frame at any time. This is done by moving the mouse cursor to the desired location on the screen and clicking buttons on the mouse, or in the case of text input, typing keys on the keyboard. There is no mode boundary between navigation and editing.

KMS makes use of contextual distinctions so that users can invoke the most frequent operations with a single point-and-click of the mouse. The location of the cursor (for instance, whether it's in empty space or inside a text item) determines which functions are available via the mouse buttons. As an aid to users, the cursor images include text labels indicating which function is currently available on each button.

Three years of testing this approach shows that users have little trouble knowing what functions the mouse buttons perform--they can always read the labels. KMS novices rely heavily on these cursor labels to learn the system. KMS experts continue to rely on the labels, but their attention is sublimated to a perceptual level. Figure 3 illustrates several KMS cursors.

Reduced command set. The move command is an example of how we have tried to streamline the KMS user interface by unifying multiple operations into a small set of commands. Pointing the cursor at an object causes the "Move, Copy, Delete" cursor to appear. Clicking the Move button at this point causes the cursor to latch onto the object. The user can drag the object around--not only within the current frame, but also across the window boundary into the other frame. In addition, while in this dragging state, the user can still perform some top-level commands, such as typing text, moving to other frames, and even creating new frames. This unification eliminates the need for a clipboard construct and the operations of cutting and pasting.

This single Move operator can perform the KMS equivalents of the following functions in other computing environments:

- Rearranging text and graphics within a diagram or page
- Moving a text string to another location in text
- Rearranging the order of sections in a document
- Moving data from one file to another
- Moving a directory or file to another directory

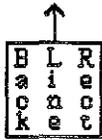
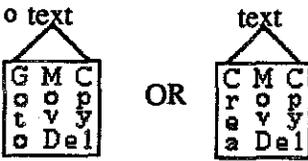
Cursor	Context	Available Functions
	When the cursor is not pointing at any item on the frame	Left: Go back to the previous frame Center: Create line or point Right: Create rectangle
	When cursor is inside of a text item (the cursor will only sit in between characters)	Left: Goto frame (if linked) Create frame (if not) Center: Move—latch onto the item(s) for movement Right: Copy the item(s) Center & Right: Delete the item(s)

Figure 3: Several KMS mouse cursor images

Invoking programs. Another category of user interaction is invoking programs by clicking on items linked to programs. These programs range from simple KMS operations, such as those provided by the customizable command menu at the bottom of a KMS window, to large conventional programs that normally run from the operating system.

A common function of KMS-based programs is to process a hierarchy of frames. For example, the program called **Linear** takes the contents of a hierarchy of frames and paginates it into a linear document, while automatically creating a table of contents, index, etc. Another program does a text search through a hierarchy of frames. With this program, a user can first narrow a search down, by going to the top of the appropriate hierarchy, before invoking the program.

3. HYPERMEDIA DESIGN ISSUES

In this section we examine a set of issues for the design of hypermedia systems. Some of these issues have been discussed in Conklin's summary of the hypertext field [Conk87], and in papers describing specific systems, such as Intermedia [Garr87], NoteCards [Hala87], Neptune [Deli86] and TIES [Shne86]. Other issues on our list haven't received as much discussion in the literature, but they have become important for the development of ZOG and KMS. We have concentrated on issues that highlight differences between KMS and other hypermedia systems. The issues are organized into four categories:

Hypermedia Design Issues

Data Model Issues

1. What is the appropriate data model for a node?
2. What is the best size for a node?
3. What types of nodes should there be?
4. What sort of data object should be used as the source for a link?
5. What sort of data object should be used as the destination of a link?
6. What types of links should there be?
7. Should a link have internal structure?
8. How can data be aggregated into large structures?

User Interface Issues

9. What style of user interface should be used?
10. How should the information in nodes be presented on the display?
11. How should a link source/destination be represented on the display?
12. How fast should the system respond when following a link?
13. How should the system support browsing?
14. Should graphical representations of the node linkage structure be provided?
15. How can disorientation be prevented or reduced?

Authoring Issues

16. How can authoring of large databases be facilitated?
17. How can material from a database be converted to paper form?

Multiple User Issues

18. How can information be jointly authored and shared by multiple users?
19. How can interference between multiple users be prevented?
20. How can access to sensitive data be restricted?

Data Model Issues

[1] What is the appropriate data model for a node?

You will recall that KMS uses a screen-sized, two-dimensional space for a node (called a frame), containing any arrangement of text, graphics and image items. This capability is flexible enough, when combined with the ability to link frames together, to allow users to represent a wide variety of knowledge structures (documents, drawings, programs, etc).

One source of the flexibility of KMS is the way it treats space. Like space in the real world, space in frames 'exists' whether or not anything occupies it. Thus a frame may be completely empty. This is distinct from the degenerate way space is represented by most text-oriented programs (e.g., word processors and mail systems). Space to the left of text is usually some mixture of space characters and tabs, while space to the right usually has no representation at all.

The spatial nature of frames is a fundamental feature of KMS that has important implications for the user:

It helps chunk items. White space provides a visual aid to perceiving the separate items on the frame. By convention, each individual text item is surrounded by white space, and therefore is easy to recognize as a separate chunk. In some hypermedia systems, a link is embedded in a larger piece of text, thus requiring some form of highlighting for the link.

It is easy to reposition items. The space in a frame provides a background on which objects can be positioned independently of one another, as is the case with graphics programs. Rearranging objects within a frame (for instance, linked text items representing a document's sections) is a moment's work.

It provides room for peripheral items. Empty space in a frame provides a handy place for peripheral items such as a note or a reviewer's comment about the contents of the frame. In KMS, creating such items is as simple as moving the cursor to an empty area in the frame and starting to type. As long as the note or comment is not unduly large, it can happily coexist with the other items in the frame. In the case of a lengthy note, the bulk of it can be placed on additional linked frames. (By convention, these items are prefaced with "@", which suppresses them from appearing in the hardcopy version of the material.)

It maps directly onto white space on paper. We know that white space on paper is a good thing, so why not in hypermedia? This is especially useful when we want to print out hypermedia-based material. If the desired white space for the printed version cannot be directly represented, how will it be supplied? (See also Issue [17].)

It provides a convenient command context. Since space in KMS is an actual construct, it provides another context for interaction. KMS exploits this context by using it as a means for creating new objects. Thus, when the cursor is in empty space, the user can directly create new points, lines, rectangles and text items. The navigation command "Back" is also available when the cursor is in empty space. In many systems, input in empty space is an error condition!

[2] What is the best size for a node?

KMS fixes the size of a frame to a width of 1140 pixels and a height of 820 pixels. These limits allow a whole frame to be displayed on most large screen displays, with some room left for window boundaries and a small message window.

The main reason we limit the size of a frame is to avoid scrolling, which we feel is an inefficient way to navigate in a database.

[3] What types of nodes should there be?

KMS has only one type of node--the frame. We have not found it necessary to have more than one type of node because of the generality of frames. Frames can contain any arrangement of text, graphics, and images. This generality plus the ability to link frames together (especially into hierarchies) makes it straightforward to represent a broad range of knowledge structures such as documents, programs, drawings, and conversations.

Implicitly, users can define frame types by placing distinctive data items in the frames and by developing distinctive frame formats. However, since KMS won't enforce these informal frame types, any processing of these frames that relies on this 'typing' is subject to error. We think it is a good tradeoff to accept the possibility of such errors, in return for the simplicity of a single system-supported node type.

[4] What sort of data object should be used as the source for a link?

The source for a KMS link is an individual text item in a frame. The text of the item describes what it's linked to. Although the text can range from a single character to a whole paragraph of text, it is most common to use a single line of text.

Links are not embedded within text as in traditional hypertext. Experimental work with TIES purports to show the superiority of embedded links over separate links [Shne87]. However, these results probably do not apply to ZOG and KMS, where the links are always visible on the same screen as the text, due to the small grain size of the node.

Using individual items as link sources decouples the contents of a frame from the links to other frames. In systems where links are embedded in text, the phrases in the text must fit into the context of the material as well as serve as links to other nodes. Also, if the material is to be transformed into a linear document, the linked phrases must appear in the order required in the document. These constraints make it more difficult to author the material. In KMS, the links can be treated separately, and can be given whatever text seems appropriate for them.

[5] What sort of data object should be used as the destination of a link?

The destination for a KMS link is a whole frame. Some hypermedia systems use an individual point within a node, or a 'region' within a node. We have never felt the need for such a capability within KMS.

[6] What types of links should there be?

KMS users do not think in terms of links per se, but rather in terms of *linked items* --that is, items that are linked to other frames. There are two types of linked items. *Tree items* have the connotation of being linked to lower-level frames in a hierarchy, such as a chapter of a book, or a procedure within a program. *Special items* are linked to peripheral material, such as comments and cross-references. These items are simply prefaced with the "@" character, which makes it easy to change the type of a linked item. The "@" is used by KMS utility programs to distinguish between the two types of links, especially for the common case of processing a hierarchy of frames.

[7] Should a link have internal structure?

In some systems, links are objects with internal structure which provides more information about the destination of the link. In KMS a link is not an object, but rather a property of a text item. Thus

links do not have any internal structure, other than the frame name representing the destination of the link. We have found that the text of the linked item can provide enough information about the destination of the link. This avoids the need for mechanisms to view and edit the internal structure of links. In addition, we feel the rapid response of KMS makes it just as practical to follow the link as it would be to see a 'preview' of the destination.

[8] How can data be aggregated into larger structures?

In KMS, aggregates can be built from regular frames. The primary way of aggregating data is to create hierarchies of frames by linking them together via *tree items*. Since frames can perform the indexing role normally provided by directories as well as the content-holding function of files, KMS users need not employ operating system directories as a mean of organizing their work. Users find this approach very natural. Many KMS utility programs are designed to work on hierarchies as input and create hierarchies as output.

User interface issues

User interface issues have always been a major focus of our work. In fact, we usually referred to ZOG as a "human-computer interface system." The ZOG Group created a User Studies Laboratory and conducted detailed studies of ZOG users. Some of this work is reported in [Robe81a], [Robe81b], and [Yode84]. Both ZOG and KMS are instrumented to collect low-level usage data. Over the years we have collected data on nearly 400,000 user sessions.

Below we discuss several important user interface issues that apply generally to hypermedia systems:

[9] What style of user interface should be used?

Because of the potential for innovation, we believe that the user interface for a hypermedia system should be designed from scratch. Consequently, we have attempted to leave behind most of our biases about user interfaces. Instead of adopting an existing style such as multiple, overlapping windows on a desktop with pull-down menus and icons, we have tried to completely open up the design of the user interface. This has proved extremely difficult.

Thus KMS today is the result of slowly unlearning many concepts and assumptions. Mostly, this has meant learning to do without things that seemed necessary before. We are trying to provide the KMS user with an environment in which there are few conceptual distinctions. Thus we dispensed with the distinction between files and directories, use a single node type, and restrict the explicit link types to just two. Also, we eliminated the mode boundary between navigating and editing, thereby dispensing with a separate "editor." Users may make changes to a frame at any time; when they leave the frame, the changes are saved automatically.

We have chosen to develop a user interface for KMS based on the direct manipulation paradigm. We have also chosen to develop the interface around the capabilities of the three button mouse. By exploiting every contextual distinction we thought natural, we have developed an interface in which over 85% of the user's interaction requires just a single point-and-click (i.e., no intermediate menu selection). As a result, KMS users can interact more than twice as efficiently as with interfaces dominated by menu selection.

[10] How should the information in nodes be presented on the display?

There are two approaches commonly used by other hypermedia systems: (1) Each node in a separate window, with multiple overlapping windows, perhaps of different sizes; and (2) A single, linear text display, where each node that is represented is expanded "in place."

KMS's choice is distinctly different: Two nodes, each taking up a full half of the display surface, or, at the user's option, one node taking up the entire display. There are no other possibilities. When a user selects an item linked to another frame, the currently displayed frame is replaced by the new frame. Because KMS can follow a link very quickly, you can think of it as using the time dimension to keep linked nodes close together, rather than trying to keep them visible on the display at the same time.

[11] How should a link source/destination be represented on the display?

Some hypermedia systems use various forms of highlighting to represent a link source on the display, e.g., italics, boldface, color, video-reversing, or a box. Unfortunately this usurps the normal use of such highlighting by the author.

Systems that use an embedded icon of some kind are prone to clutter. By themselves, icons often do not provide enough information to enable the user to make a good decision about whether or not to follow the link. In addition, these icons are often small targets, which require more time to select.

KMS uses whole text items as link sources. A linked item is displayed with a small circle to its left indicating the existence of a link. Since the text item is normally surrounded by a sea of empty space, the range or region of the link source is defined implicitly. The content of the text item can provide as much semantic information about the link as is needed. Also the average size of linked items makes them easy to point to them.

Since KMS links are one-way, and the destination of a link is a whole frame, there is no need to denote the destination of a link.

[12] How fast should the system respond when following a link?

We believe that fast system response to selecting a link is one of the most important parameters in a hypermedia system. Even though the average time a user spends at a node will usually be many seconds, there will be frequent bursts of rapid navigation, when response time becomes critical. Our experience with a variety of hypermedia systems has shown that the difference between one system with a response of several seconds and another with sub-second response is so great as to make them seem qualitatively different. Our design goal for KMS is to be able to access and display a random frame across a wide-area network in less than .25 seconds on average.

In the early 1970's, researchers at the PROMIS laboratory produced a hypermedia system capable of 0.25 second response 70% of the time, using specialized hardware [Schu79]. Our early versions of ZOG, created in 1976, ran on DEC time-sharing machines with 1200 baud terminal links and provided response times of 5 to 10 seconds. When we graduated to 9600 baud around 1979, response was improved to about 2 or 3 seconds, and it seemed like a major breakthrough for users. Our PERQ version of ZOG, completed in 1983, gave an average response of about 0.7 seconds for frames local

to the machine, and 1.5 seconds for frames accessed over the Ethernet. Users again experienced a dramatic improvement over the previous version, but they quickly adapted to the new speed and still hungered for more.

We have also had experience with response speeds at the very fast end of the scale--0.05 to 0.1 seconds. In 1978, as part of the ZOG effort at CMU, we built two special ZOG terminals using a high-speed vector graphics display, a touch screen, and a fast drum, attached to one of the DEC PDP-11 processors in the experimental multiprocessor called C.mmp. We were not able to study the use of this system in any detail, because it had no editor available, it was difficult to download material from our main working environment on a PDP-10, and the hardware was unreliable. But we did satisfy ourselves that we had bounded the optimal response time from below. In fact, without some explicit cue, 0.05 second response may be too fast--we had trouble noticing whether or not the screen had changed, especially if we blinked at the wrong time!

In making the initial leap from ZOG to KMS, we took a step backward in response speed. This happened because frames tended to become larger and more complex as we took advantage of larger bit-mapped displays. Also, we began using a separate file for each frame, for added flexibility. Fortunately, KMS has benefitted greatly from the faster hardware now available, so that KMS once again has sub-second average response times.

KMS's responsiveness is mostly a function of the amount of material in the average frame (1 Kbyte), the graphics performance of the window system, and the speed of the storage device. Interestingly, frames stored remotely on a file server with a fast disk can often be accessed more quickly than frames stored locally on a slower disk.

The larger memories now available in workstations (4 Mbytes being typical) have allowed us to implement a frame caching mechanism that further speeds the response by eliminating file accesses for frames already in the cache. In Figure 4 we show typical response times for KMS running on a Sun-3/50 with 4 Mbytes of memory, using a locally-attached small disk.

	Small frame (~ 0.4 Kbytes)	Med. frame (~ 1.6 Kbytes)	Large frame (~ 3.5 Kbytes)
From disk	0.34	1.02	2.60
From cache	0.20	0.28	0.30

Figure 4: Time for KMS to access and display a frame (in seconds). The average size for KMS frames is 1 Kbyte.

[13] How should the system support browsing?

We believe that the ability to browse quickly in a hypermedia system is critical to its usability. This is particularly true of larger-scale hypermedia databases, where it's necessary to 'travel' longer distances. Although system response time is an important factor for browsing, there are other aspects as well:

Standard frame layout. The conventions for the layout of a frame make it easier for the user to assimilate the information on the frame. As a result, it takes less time to decide what to do next.

Time user takes to select. On average, linked items are large in size (compared with embedded icons) and segmented spatially. This reduces the time it takes users to point the cursor at them.

No mode boundary between editing and navigation. The user need not cross a mode boundary in order to switch between editing and navigating. Navigation and editing commands are simultaneously available.

Fast backtrack command. Backtracking is a frequent activity--for every movement forward there tends to be a compensating move back. In KMS, the Back command is available as one of the buttons of the 'empty space' cursor. The user need only move the cursor to an empty area of the frame to get into the proper context and click the Back button. On average this takes .7 seconds to do, partly because the cursor often doesn't need to be moved. This compares with 1.5 seconds to click on a menu command (such as the command items at the bottom of a KMS window). This small difference adds up since the Back command may be used several hundred times per hour. From the user's perspective, it's not just the time savings, but the reduced mental and physical effort.

[14] Should graphical representations of the node linkage structure be provided?

Periodically we consider providing additional views in KMS such as a graph of a portion of the network. But each time we retreat. We believe such views are unnecessary, except perhaps for large, essentially non-hierarchical structures. Our own experience indicates that our 'mind's eye' sees KMS structures as time-travel through familiar frames, rather than as some graphical representation of the structure. This view is supported by our ZOG user studies, which revealed that users rarely made use of the multi-node views that were available. The 'overview-like' nature of frames, plus being able to travel in the database rapidly--seems to substantially reduce the need for such structures in KMS.

[15] How can disorientation be prevented or reduced?

The classical hypermedia problem is the "getting lost problem," which becomes more severe as the database grows larger. However, we have found that getting lost is not much of a problem for KMS users. KMS has characteristics that help users stay oriented, plus some features that help users re-orient themselves if they do get lost.

Hierarchical backbone. KMS strongly encourages a top-down, stagewise refinement approach to organizing material in the database. The resulting hierarchical "backbone" in the database helps users build a coherent mental model of the database. Also, multiple hierarchies can be constructed to provide alternative paths through the database.

Special navigation commands. KMS provides several commands that let users go directly to specific locations in the database. The Goto command lets a user go directly to any named frame. The Home command displays a user's home frame. The Info command displays a frame with links to KMS documentation and utilities.

Marking the item just returned from. KMS flags the item linking to the frame from which the user has just backtracked with a temporary asterisk.

Richer frames. The use of larger frames provides a richer context in which to assimilate knowledge. Also, since there are fewer frames, less travel is required.

Fast response. The ability to navigate quickly from frame to frame makes exploration less risky for users, since they can always backtrack quickly to return to a familiar frame.

Authoring Issues

Below we discuss a couple of issues dealing with how hypermedia databases can be created. These are important because database creation is a severe bottleneck. A user's ability to assimilate information far outstrips his ability to generate it.

[16] How can authoring of large databases be facilitated?

Small databases are of limited interest. This poses an economics problem for hypermedia system designers to solve. If it's too inconvenient to build a hypermedia database users will avoid doing it. Listed below are some of the approaches we have taken to encourage the development of large-scale databases:

Rapid navigation. Users need to be able to move around rapidly in order to get to where they wish to build. This is also important in the frequent case of moving objects to a different place in the database.

No editing/navigation mode transition. KMS does not have a mode transition between navigation and editing (see Issue [9]).

Rapid creation of new frames. To create a frame, all a user has to do is click on an unlinked item. Typically, the user can be editing a new frame less than two seconds after deciding to create it.

Default operand scope. Editing in KMS is dominated by manipulating individual items. Since the default scope for operations is the whole item pointed to by the cursor, the vast majority of operations can be invoked directly, without any explicit scope designation.

Implicit saving of changes. Tentative modifications are limited to the currently displayed frames. If there are any changes to a frame those changes will be automatically saved when the user moves to another frame. This default works well in practice. Not only does it eliminate most explicit save invocations, but it reduces the complexity of the user's model of the current state of the system.

Use of schemas. Schemas are chunks of data (e.g., a frame or tree of frames) that contain variable parts. Schemas can be used to build data objects that have some common parts, simply by copying the schemas and filling in the variable parts manually. Ramakrishna [Rama81] developed schema mechanisms for ZOG and studied their use experimentally.

Tools for importing external databases. KMS provides a number of tools for mapping in material from other sources (e.g. text files and bitmap files).

Support for multiple users. KMS is a distributed hypermedia system designed to support simultaneous building of a KMS database by multiple users. (Please see "Multiple User Issues")

No restriction on the size of the database. KMS databases may be as large as available secondary memory and may be distributed across any number of workstations and file servers.

Database merging. Independently developed KMS database are easily joined together to form a single database.

[17] How can material from a database be converted to paper form?

One of the major forces guiding our design efforts has been the desire to create well-formatted documents from material in the database. Since frames provide a local WYSIWYG view, there is a natural process for paginating the material: concatenating the contents of frames from a hierarchy in depth-first order. This default can be supplemented by additional formatting commands (e.g., "@NewPage," "@Figure," etc.) that are placed on frames to specify additional formatting constructs. Typically, these items, like other meta-level items such as notes and comments, are placed off in the corner to keep them out of the reader's way. This approach is a hybrid between pure WYSIWYG document systems (in which little structure is represented explicitly) and structured formatting systems (Scribe and T_EX). KMS also offers the flexibility of applying the document formatting process at any level of a hierarchy of frames, thereby enabling users to get just the portion of a document they want.

Multiple User Issues

Both ZOG and KMS have been designed from the beginning to support a community of communicating users, where users can jointly develop and share data, rather than simply exchange it. Below we discuss several of the relevant issues.

[18] How can information be jointly authored and shared by multiple users?

KMS provides to a community of users a single, logical database, physically distributed across multiple workstations and file servers on a network. The actual physical location of data can be completely transparent to the users--as if they were all users on a single time-sharing system, but with vastly improved response and display bandwidth.

Our first real implementation of a distributed system was the version of ZOG running on the PERQ network on board the USS CARL VINSON, which was completed in 1983. It implemented special ZOG network servers that managed the locking of individual ZOG frames for modification by one user at a time. It had a location database, managed by one of the machines designated as the *master*, with information about which machine each ZOG frameset was actually located on.

Our current version of KMS uses Sun's Network File System (NFS) to provide access to frames that reside on remote machines. As with ZOG, there is a *master* file server that holds the location of all frameset. (All file servers containing a portion of the KMS database have automatically-maintained copies of this location information, to be used in case the master is unavailable).

One of the major benefits of KMS is the ability of multiple users to work simultaneously on a common project such as proposal or conference paper. Working together on a single document (or single area of the database), users can easily see what others have done, make comments, print out any part of the material at any time, etc. There is no strict coordination concerning the evolution of the database similar to that required in conventional database systems.

This communal approach makes it possible to communicate electronically in a way quite different from conventional electronic mail. In KMS, conversations simply 'grow' in some area of the database, thus preserving their logical structure.

[19] How can interference between multiple users be prevented?

How can we prevent multiple users from losing changes due to interference, yet avoid the inefficiencies of locking users out from making changes for long periods of time? In ZOG, we provided for the locking and unlocking of a frame when the user entered and exited the editor, respectively. In KMS, we do not lock frames in this way. Instead, we use a weaker form of concurrency control, called 'optimistic concurrency control.' We make the optimistic assumption that since frames greatly outnumber users, a conflict between users editing the same frame is rare.

All 'optimistic concurrency' guarantees is that a KMS user who has successfully saved changes to a frame cannot subsequently have those changes revoked by another user who had been editing the same version of the frame. It does not guarantee that if you edit a frame, you will necessarily be able to save the changes without any problem. At the time you attempt to save your changes, you may be informed that someone else has already saved changes to the same frame. This means that your tentative changes cannot be saved, because they would revoke the other user's changes. What KMS does in this case is to temporarily save your changes in a newly created frame, so that you can then map them into the new version of the original frame.

Our experience shows this situation rarely occurs since users are normally working in different areas of the database, even when they are working on the same document (for example, this paper is represented by over 200 frames). Whenever users find they are 'bumping elbows,' they can cope by using informal frame 'locking' conventions--namely, by placing a text item on a frame that warns the other users who come to that frame that editing is in progress. Besides being more personal, this informal locking can be used to alert others you plan to do some work in this area of the database.

On the face of it, optimistic concurrency control may seem unwise, since it is not a foolproof mechanism for preventing interference between users. But adopting it allowed us some benefits that we feel well outweigh its drawbacks. The most important benefit was that it facilitated eliminating the mode boundary between navigating and editing (see Issue [9]).

[20] How can access to sensitive data be restricted?

To prevent access to sensitive data, KMS implements protection of individual frames. Every frame has an owner--originally, the person who created it. The owner can protect the frame so that others may access it but not make any modifications, or so that others can't even access the frame.

An intermediate kind of protection (called *annotation access* by the Intermedia researchers [Garr87]) seems like a useful capability to add to KMS. It would allow users to add new items to a frame without allowing them to modify any of the existing items. For now, we generally leave frames unprotected to allow free annotation, and simply rely on the good will of users to not delete the work of other users without permission.

4. CONCLUSION

If there is one central theme to our experience, it is the fundamental importance of a system's data model. Our experience with ZOG and KMS has convinced us that the data model underlying an interactive system strongly determines the nature of its user interface. We believe this because we have seen the formative influence of the KMS data model on all other aspects of KMS.

In the case of KMS, the properties of a node--its fixed size, its spatial nature, how links are represented within it, its standard format, etc.--contribute significantly to the global nature of the system and distinguish it strongly from other hypermedia systems.

Consequently we recommend that interactive systems be developed from the inside out--from the data model to the user interface--rather than the other way around. This view contrasts sharply with the philosophy that the user interface should be the dominant system component and thus standardized across programs. Perhaps hypermedia, with the structural richness it has to offer human-computer interaction, may eventually overshadow the reigning HCI paradigm, the desktop interface.

ACKNOWLEDGMENTS

We wish to acknowledge the contributions of many people over the years. Those who were involved with ZOG at CMU: Allen Newell, George Robertson, Kamila Robertson, Peter Lieu, Sandy Esch, Patty Nazarek, Marilyn Mantei, Kamesh Ramakrishna, Roy Taylor, Mark Fox and Andy Palay. Those officers from the USS CARL VINSON who worked with us at CMU: Mark Frost, Paul Fischbeck, Hal Powell, Russ Shoop, and Rich Anderson. Captain Richard Martin, Cdr. Ted Kral, Lt. Brian MacKay, and other officers and crew of the USS CARL VINSON. Finally, we would like to thank the Office of Naval Research for sponsoring the 10 years of the ZOG Project.

REFERENCES

- [Aksc84a] Akscyn, R. and D. McCracken, "The ZOG Approach to Database Management," *Proceedings of the Trends and Applications Conference: Making Database Work*, Gaithersburg, Maryland, May 1984.
- [Aksc84b] Akscyn, R. and D. McCracken, "ZOG and the USS CARL VINSON: Lessons in System Development," *Proceedings of the First IFIP Conference on Human-Computer Interaction (Interact '84)*, London, U.K., September 1984.

- [Conk87] Conklin, J., "A Survey of Hypertext," MCC Technical Report STP-356-86, Rev. 1, February 1987. To appear in *IEEE Computer*, September 1987.
- [Deli86] Delisle, N. and M. Schwartz, "Neptune: A Hypertext System for CAD Applications," *Proceedings of ACM SIGMOD International Conference on Management of Data*, Washington, D.C., May 1986, pp. 132-143.
- [Garr87] Garrett, L., K. Smith and N. Meyrowitz, "Intermedia: Issues, Strategies, and Tactics in the Design of a Hypermedia Document System," *Proceedings of the Conference on Computer-Supported Cooperative Work*, Austin, Texas, December 1986, pp. 163-174.
- [Hala87] Halasz, F., T. Moran and R. Trigg, "NoteCards in a Nutshell," *Proceedings of the ACM Conference on Human Factors in Computing Systems*, Toronto, Canada, April 1987.
- [Mant82] Mantei, M., *A study of Disorientation Behavior in ZOG*, PhD thesis, University of Southern California, 1982.
- [McCr84] McCracken, D. and R. Akscyn, "Experience with the ZOG Human-Computer Interface System," *International Journal of Man-Machine Studies*, Vol. 21, 1984, pp. 293-310.
- [Newe85] Newell, A., "An On-Going Case Study in Technological Innovation," in *Advances in Information Processing in Organizations*, Sproull, L. and P. Larkey (eds.), 1985.
- [Newe81] Newell, A., D. McCracken, G. Robertson and R. Akscyn, "ZOG and the USS CARL VINSON," *Computer Science Research Review*, Carnegie-Mellon University, 1981, pp. 95-118.
- [Rama81] Ramakrishna, K., *Schematization as an Aid to Organizing ZOG Information Nets*, PhD thesis, Computer Science Department, Carnegie-Mellon University, 1981.
- [Robe82] Robertson, C.K. and R. Akscyn, "Experimental Evaluation of Tools for Teaching the ZOG Frame Editor," *Proceedings of the International Conference on Man/Machine Systems*, Manchester, U.K., July 1982.
- [Robe81a] Robertson, C.K., D. McCracken and A. Newell, "Experimental Evaluation of the ZOG Frame Editor," *Proceedings of the 7th Canadian Man-Computer Communications Conference*, Waterloo, Ontario, June 1981, pp. 115-123.
- [Robe81b] Robertson, G., D. McCracken and A. Newell, "The ZOG Approach to Man-Machine Communication," *International Journal of Man-Machine Studies*, 1981.
- [Schu79] Schultz, J. and L. Davis, "The technology of PROMIS," *Proceedings of the IEEE*, September 1979, pp. 1237-1244.
- [Shne86] Shneiderman, B. and J. Morariu, "The Interactive Encyclopedia System (TIES)," Department of Computer Science, University of Maryland, College Park, MD, June 1986.
- [Shne87] Shneiderman, B., "User Interface Design and Evaluation for an Electronic Encyclopedia," Technical Report CS-TR-1819, Department of Computer Science, University of Maryland, March 1987.
- [Yode84] Yoder, E., McCracken, D., and R. Akscyn, "Instrumenting a Human-Computer Interface for Development and Evaluation," *Proceedings of the First IFIP Conference on Human-Computer Interaction (Interact '84)*, London, U.K., September 1984.

HAM: A General-Purpose Hypertext Abstract Machine

Brad Campbell
Joseph M. Goodman

Tektronix, Inc.
Computer-Aided Software Engineering Division
P.O. Box 4600, M.S. 94-480
Beaverton, Oregon 97076

ABSTRACT

The Hypertext Abstract Machine (HAM) is a general-purpose, transaction-based, server for a hypertext storage system. The server is designed to handle multiple users in a networked environment. The storage system consists of a collection of contexts, nodes, links, and attributes that make up a hypertext graph. This paper demonstrates the HAM's versatility by showing how Guide¹ buttons, Intermedia webs, and NoteCards FileBoxes can be implemented using the HAM's storage model.

INTRODUCTION

Tektronix' Hypertext Abstract Machine (HAM) is a general-purpose, transaction-based, multi-user server for a hypertext storage system. The HAM is based on the abstract machine Norm Delisle and Mayer Schwartz used in their Neptune system developed at Tektronix' Computer Research Laboratory [Deli86]. The HAM is an underlying component of the Tektronix CASE Division's Software Engineering Information System development effort. Because the HAM is a low-level storage engine, it provides a general and flexible model that can be used in several different hypertext applications.

The HAM stores all of the information it manages in graphs, or databases, on a host machine's file systems. Graphs are stored in a centralized area and can be accessed in a distributed environment. If a distributed file system is shared by a series of machines, the HAM does not reduce the file system's functionality.

Applications normally communicate with the outside world through a common user interface. This interface is window-based and highly interactive to provide a suitable environment for a hypertext system.

Figure 1 shows the typical organization of a system using the HAM.

1. Guide is a trademark of OWL International, Inc.

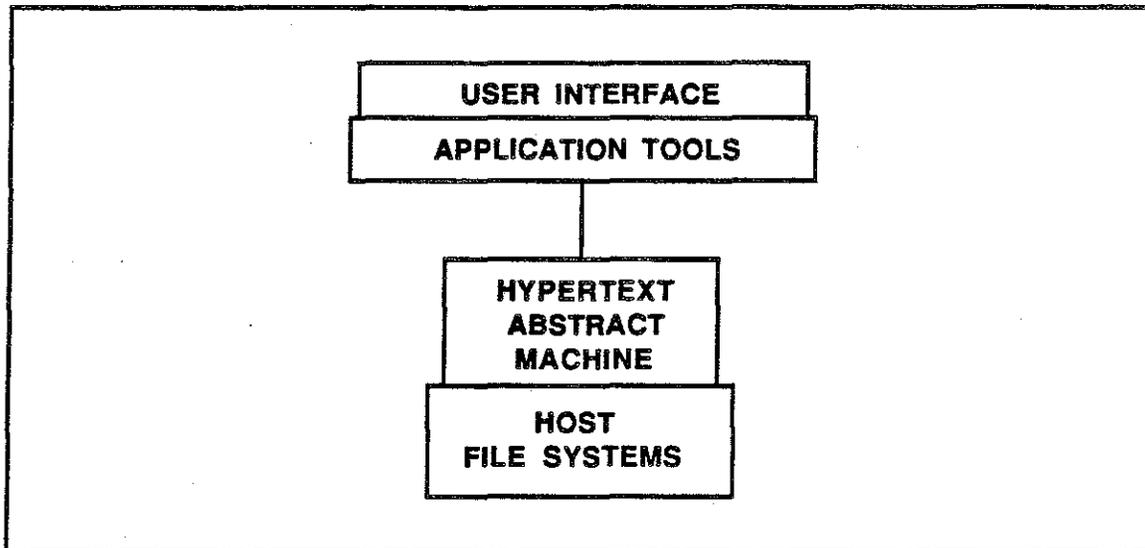


Figure 1. Generic hypertext system architecture.

In this paper, we describe the HAM's functionality and discuss how it can be used by potential applications. First, we describe the major features of the HAM and provide an overview of HAM operations. We then describe possible HAM representations for three hypertext data structures: Guide buttons, Intermedia webs, and NoteCards FileBoxes.

HAM FEATURES

The HAM storage model is based on five objects: graphs, contexts, nodes, links, and attributes. The HAM maintains history for these objects, allows selective access through a filtering mechanism, and can allow for access restrictions through a data security mechanism.

HAM Objects

A graph contains contexts, nodes, links, and attributes. These objects are organized hierarchically. The following paragraphs describe each of the objects.

Graphs. A *graph* is the highest-level HAM object. It normally contains all of the information regarding a general topic, such as the information for a software project. A graph contains one or more contexts.

Contexts. *Contexts* partition the data within a graph. Each context has one parent context and zero or more child contexts. When a graph is created, a root context begins the tree. A context does not depend on information contained in its parent context. A context contains zero or more nodes and links.

Nodes. A *node* contains arbitrary data. This data can be stored as text or as fixed-length binary blocks. When a node is updated, a new version is created by replacing the previous contents with the new contents. Previous versions of a node can be retrieved. A node can be an *append only* node. Updates to an append only node are appended to its contents. Append only nodes are useful for logging the actions performed by an application. A node's contents can also be searched for the occurrence of user specified regular expressions. Nodes are related by links.

Links. A *link* defines a relationship between a source node and a destination node and can be followed in either direction. A *cross-context* link relates two nodes in different contexts. Cross-context links are useful for sharing data between two contexts. The generality provided by link attributes allows application writers to define their own notions of link types or link end-point attachment schemes.

Attributes. *Attributes* can be attached to contexts, nodes, or links. Attribute values can be strings, integers, floating-point numbers, or user-defined types. Attribute/value pairs give semantics to HAM objects. They can represent application-specific properties of objects or contain information that further describes an object. Attributes are also used in the predicates that are part of the HAM filters.

Version History

The HAM provides an automatic version history mechanism. The version history for a HAM object is updated each time that object is modified. Because each access to an object contains a version time, previous versions of objects can be viewed. The HAM also provides operations to destroy undesired versions.

Filters

The HAM provides a filtering mechanism that allows subsets of HAM objects to be extracted from large graphs. Filters allow the user to specify visibility predicates, which are expressions relating attributes and their values. HAM filters only return objects that satisfy the predicates. Filters also allow the user to specify a version time so that earlier versions of a graph can be examined.

The HAM filters the following items:

- Contexts in a graph
- Nodes in a context
- Links in a context
- Instances of a node in specified contexts

- Instances of a link in specified contexts
- A set of nodes and links in specified contexts based on a specific link ordering

Data Security

The HAM provides security for the data contained in a graph through its access control list (ACL) mechanism. Attaching an ACL to an object is optional. An ACL entry consists of a user or group name and a set of permissions. A *user* is anyone who has access to the graph. A *group* is a list of users. The available *permissions* are access, annotate, update, and destroy.

The permissions associated with an ACL entry are additive. Access permission allows the user or group to view the data associated with the object. Annotate permission allows links to be attached to a node. Update permission allows the user or group to perform nondestructive updates on an object. Destroy permission allows the destruction of an object.

HAM OPERATIONS

To provide a consistent, simple interface, HAM operations are grouped into seven categories. Operations within a category behave similarly, regardless of the object on which they operate.

Create Operations

Create operations create new HAM objects. A create operation takes object-dependent data and returns an object index and a version time. The object index represents a unique identifier for the newly created object, and the version time denotes the time at which the object was created.

Delete Operations

Delete operations mark objects as deleted but retain historical information. A delete operation takes an object index and a version time, and returns a new version time. The object index specifies the unique identifier for the object being deleted. The returned version time represents the time the object was deleted.

Destroy Operations

Destroy operations free all space required for an object. The object does not have to be deleted to be destroyed. A destroy operation takes an object index and a version time, and returns a new version time. The object index specifies the unique identifier for the object being destroyed. The returned version time represents the time the object was destroyed.

Change Operations

Change operations modify data associated with an existing object. A change operation takes an object index, a version time, and object-dependent data. These operations return a version time. The object index specifies the unique identifier for the object being modified. The returned version time represents the time the object was modified.

Get Operations

Get operations retrieve data from existing objects. A get operation takes an object index and a version time, and returns the data that existed at the specified time. The object index specifies a unique identifier for the object from which data is being retrieved. The version time is a time range for the data retrieval.

Filter Operations

Filter (and linearize) operations selectively retrieve information from a graph. A filter operation takes a predicate, a version time, and a list of attributes. These operations return a list of objects that satisfy the predicate and a list of requested attributes attached to each object. The version time specifies the time at which the filter is to search for the information. Each filter operation also has unique parameters in addition to those already specified.

Special Operations

Operations that do not fit into any of these categories are considered special. They include functions such as searching for strings in node contents, merging contexts, and managing transactions.

EXAMPLE HAM APPLICATIONS

Because the HAM is a general-purpose hypertext engine, it can serve many types of hypertext systems. In this section, we will model three hypertext structures using the HAM's storage model: Guide buttons, Intermedia webs, and NoteCards FileBoxes.

Guide Buttons

Guide is a hypertext product developed for the Macintosh² by OWL International, Inc. of Bellevue, WA [Guid86]. It is a tool for writing and reading electronic documents. Guide uses buttons to represent links in a document between the information on the screen and related information. A button is a special area on

² Macintosh is a trademark of Apple Computer, Inc.

the screen. When a button is selected, by clicking the mouse, Guide follows the link to display the related information.

Replacement buttons replace the button icon displayed on the screen with the information associated with that button. *Inquiries* are sets of two or more mutually exclusive replacement buttons. *Reference buttons* display the information associated with the button in a new window. This window remains visible until the user returns to the document window. *Note buttons* display information associated with the button in a new window that disappears when the user releases the mouse button.

To model Guide, the HAM equates a document with a node. The various button relationships are modeled as links. Link attributes determine which type of button the link represents. The application uses these link attributes to determine which type of window to open when a button is selected.

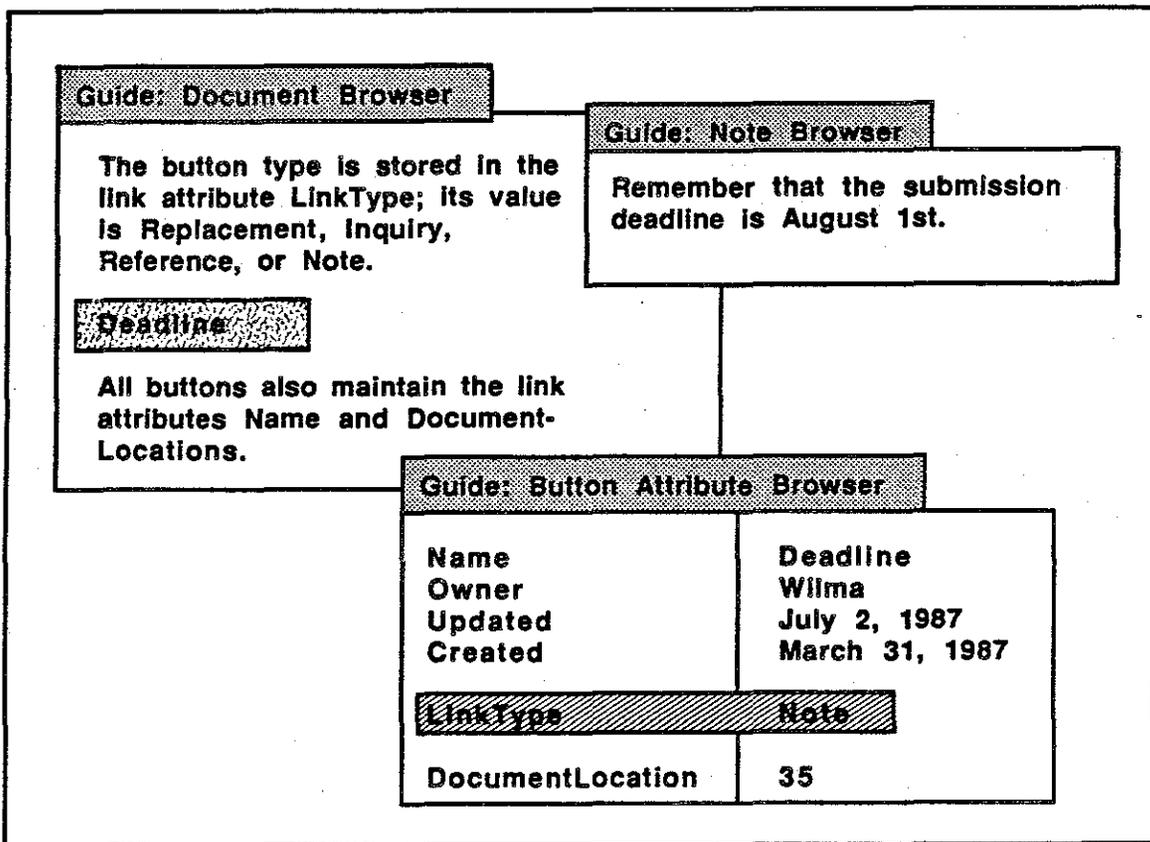


Figure 2. Possible representation for a Guide note button.

Figure 2 shows an example of a note button. The Document Browser contains the text being examined; the icon within the browser represents the note button. The Note Browser contains the note associated with the note button. The Button Attribute Browser shows the attributes associated with the link representing the note button, as well as the value of the *LinkType* attribute.

The button type is stored in the link attribute *LinkType*; its value is Replacement, Inquiry, Reference, or Note. All buttons also maintain the link attributes *Name* and *DocumentLocation*. *Name* represents the name associated with the button, and *DocumentLocation* defines the location relative to the beginning of the document where the button was created. The value of *DocumentLocation* corresponds to Guide's location of its button icon. Guide considers the information associated with a button to be an atomic entity. Therefore, the other end of the link representing the button can point to the entire node that contains the button's information.

If a replacement button is part of an inquiry, the value of *LinkType* is set to Inquiry. A link that represents part of an inquiry also has an attribute named *Grouping*, which contains the identification of a special node. This node contains the identification of all links (replacement buttons) that make up the inquiry.

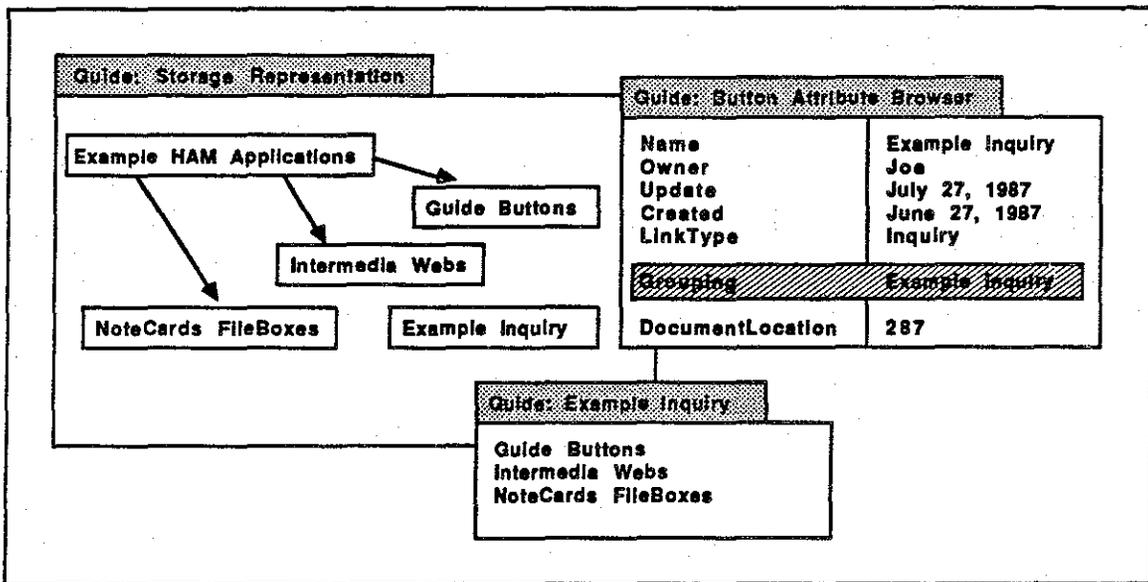


Figure 3. Inquiry storage representation.

Figure 3 shows the HAM storage model for an inquiry named Example Inquiry. The Storage Representation window shows the nodes and links involved in the inquiry. In this example, the links have the same name as their destination nodes. The node Example HAM Applications is the document node. The nodes Guide Buttons, Intermedia Webs, and NoteCards FileBoxes contain the information associated with the replacement buttons that make up the inquiry. The node Example Inquiry contains the names of the replacement buttons in the inquiry; its contents are shown in the Example Inquiry browser. The Button Attribute Browser displays the attributes attached to one of the links involved in the inquiry and shows that the value of the *Grouping* attribute is Example Inquiry.

Intermedia Webs

Intermedia, the system developed at the Institute for Research in Information and Scholarship at Brown University [Garr86, Yank85], is one of the newer and more innovative hypertext systems.

The basic hypertext concepts in Intermedia are very similar to those found in the HAM. Intermedia uses the term *web* to refer to a database that contains both references to a set of documents and the links associated with those documents [Meyr86]. A *block* is the piece of a document to which a link is anchored and can be any legitimate selection in the application. The attributes provided by the HAM allow the flexibility to efficiently model these relationships.

To model an Intermedia web, the HAM represents a web as a collection of nodes and links. A document is represented as a node. An Intermedia link is equivalent to a HAM link. Blocks are determined by using link attributes to define the anchor selections for both the source and destination ends of each link.

UNIX³ manual pages⁴ provide a convenient example of how the HAM can model Intermedia webs. The manual page for the mail command is used to create a small web of information.

Each document (manual page) is represented as a HAM node. The web is defined by attaching an attribute named *Web* to each link. The value of this attribute contains the name of the web to which the link belongs. A link filter is applied using the predicate "*Web = mail*" to let users view a map of the web. This filter returns only those nodes that are part of *mail*.

Figure 4 shows the mail web defined by creating links from the mail command to commands in the manual page's "SEE ALSO" section.

To define a block, the HAM uses the attribute pairs *SourceOffset/SourceExtent* and *DestinationOffset/DestinationExtent*. A block is determined by the value of the attribute pair attached to the link. For example, the source block of a link is represented by the attributes *SourceOffset* and *SourceExtent*. The values of these attributes are integers that contain the byte offset from the beginning of the node and the length of the block.

Each block is defined by the offset and extent attributes. The offset provides an insertion point for the block, and the extent determines the end point of the block.

Figure 5 shows the value of the *SourceOffset* and *SourceExtent* attributes attached to link *BinMail*. The highlighted area shows the block these attributes define.

3. UNIX is a registered trademark of AT&T Bell Laboratories.

4. Excerpts from the UNIX Programmers Manual, Berkeley Distribution, are used for purposes of illustration.

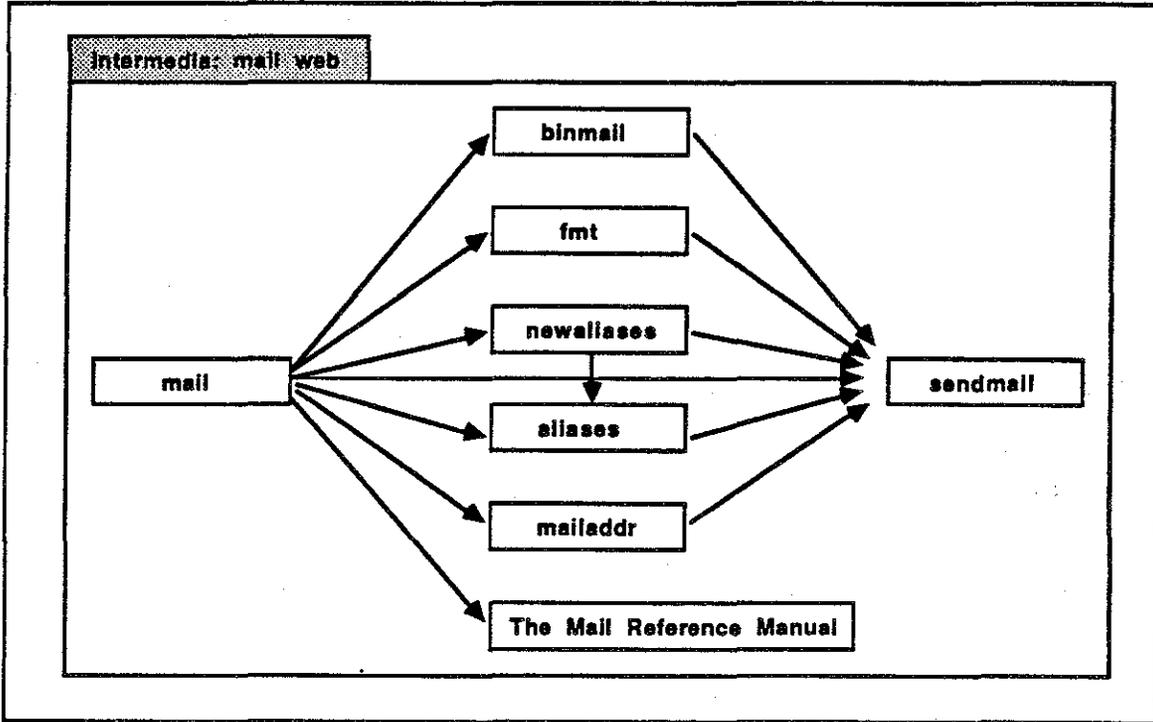


Figure 4. Mail web.

Intermedia: MAIL Document

FILES:

<pre> /usr/spool/mail/* ~/mbox ~/.mailrc /tmp/R# /usr/lib/Mail.help* /usr/lib/Mail.rc Message* </pre>	<pre> post office your old mail file giving initial mail commands temporary for editor escape help files system initialization file temporary for editing messages </pre>
---	---

SEE ALSO

Source Extent

[binmail\(1\)](#), [fmt\(1\)](#), [newaliases\(1\)](#), [aliases\(5\)](#),
[mailaddr\(7\)](#), [sendmail\(8\)](#)
 "The Mail Reference Manual"

Intermedia: Binmail Document

NAME

binmail - send or receive mail among users

SYNOPSIS

```

/bin/mail [ + ] [ - ] [ person ] ..
/bin/mail [ + ] [ - ] -f file
  
```

DESCRIPTION

Note: This is the old version 7 UNIX system mail program. The default mail command is described in mail(1), and its binary is in the directory /usr/ucb.

Intermedia: Link Attribute Browser

Owner	Bazings
Updated	July 20, 1987
Created	June 8, 1987
Source Offset	24423
Source Extent	11
Destination Offset	122
Destination Extent	7

Figure 5. Defining a block.

NoteCards FileBoxes

NoteCards is a general-purpose idea-processing hypertext system developed at Xerox PARC [Hala87]. NoteCards supports the concept of FileBoxes. Every notecard must be stored in one or more FileBoxes. A *FileBox* can contain notecards and other FileBoxes. The FileBox structure is arranged as a directed acyclic graph.

FileBoxes can be represented in the HAM using nodes, links, and attributes. Both FileBoxes and notecards are equivalent to nodes. The model uses a node attribute to determine whether a node is a FileBox or a notecard. Links show which notecards (or FileBoxes) are in a particular FileBox. Link attributes determine which links refer to other FileBoxes and notecards. This model allows nodes to reside in more than one FileBox. The example shown in Figure 6 helps to clarify the NoteCards FileBox model.

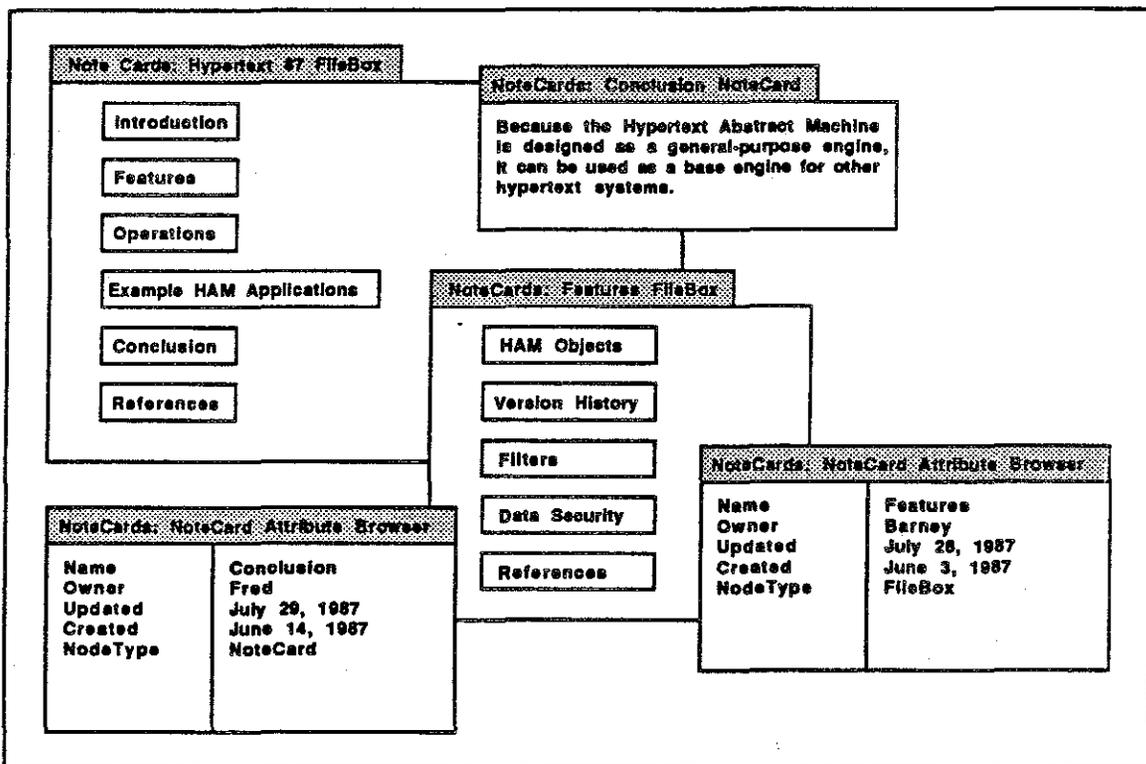


Figure 6. NoteCards representation.

The FileBox named *Hypertext 87* contains all of the FileBoxes and notecards that make up this paper. As shown in the *Features NoteCard Attribute Browser*, the *Features* node is a FileBox. When a user browses this node, the NoteCards-like application examines the *nodeType* attribute, determines that the node is a FileBox, and opens a new FileBox browser. The contents of the *Features* node are links to all of the FileBoxes and notecards that it contains. Note that *References* is contained in both FileBoxes.

The Conclusion NoteCard Attribute Browser shows that the Conclusion node is a NoteCard. When a user browses this node, the application examines the *nodeType* attribute, determines that the node is a NoteCard, and opens a NoteCard browser.

CONCLUSION

Because the Hypertext Abstract Machine is designed as a general-purpose hypertext engine, it can be used as a base engine for other hypertext systems. Most current hypertext systems emphasize the application and user interface layers. While these layers are very important an appropriate storage model is essential. We believe the HAM provides such a model.

Although the HAM is not a panacea for hypertext data storage problems, it is an important first step. As new hypertext applications are developed, we will learn more about the data representation problems hypertext presents. If a storage model standard develops from this work, it may lead to the development of a standard terminology and base engine that could improve immeasurably the progress of hypertext technology.

ACKNOWLEDGEMENTS

We would like to thank Amy Rivero and Rich Davenport for their editing and illustration assistance. We wish to thank Norm Delisle and Mayer Schwartz of the Tektronix Computer Research Laboratory for their helpful comments. We would also like to thank them for their patience during the past year as they helped us learn about hypertext.

REFERENCES

- [Deli86] Delisle, Norman and M. Schwartz. "Neptune: A Hypertext System for CAD Applications." *Proceedings of ACM SIGMOD '86*, Washington, D.C. (May 28-30, 1986): 132-142.
- [Garr86] Garrett, N., K. Smith, N. Meyrowitz. "Intermedia: Issues, Strategies, and Tactics in the Design of a Hypermedia Document System." *Proceedings of the Conference on Computer Supported Cooperative Work*, Austin, TX. (December 3-5, 1986): 163-174.
- [Guid86] *Guide: Hypertext for the Macintosh Manual*. Bellevue, WA: OWL International, Inc., 1986.
- [Hala87] Halasz, F., T. Moran, R. Trigg. "NoteCards in a Nutshell." *CHI + GI Conference Proceedings*, Toronto, Ontario, Canada. (April 5-9, 1987): 45-52.
- [Meyr86] Meyrowitz, N. "Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework." *OOPSLA '86 Proceedings*, Portland, Oregon.

(September 29 - October 2, 1986): 186-201.

[Schw86] Schwartz, Mayer and N. Delisle. "Contexts - A Partitioning Concept for Hypertext." *Proceedings of the Conference on Computer Supported Cooperative Work*, Austin, TX. (December 3-5, 1986): 147-152.

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

Turning Ideas into Products: The Guide System

P.J. Brown

Office Workstations Ltd., 5 Abbeymount Techbase
2 Easter Road, Edinburgh EH7 5AN

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

ABSTRACT

The Guide system is a successful commercial product that originally came out of some ideas of a research project. Unlike many other hypertext systems, Guide is aimed at naive users and authors in the personal computer market. This paper evaluates the basic principles of Guide, and describes the interplay between the product and the continuing hypertext research programme.

INTRODUCTION

Many hypertext systems today are research prototypes that are starting to have their first experimental uses on real documents. Although some systems are in the field — including Augment, following Engelbart's pioneering ideas (Engelbart, 1963) — most require some sophistication either on the part of users or, more particularly, authors. In general systems have not reached, and often have not been designed to reach, a mass market. The Guide system is an exception to this: it has been marketed as a product since August 1986. Members of the public who buy Guide not only have to learn how to use the system; they also need to author their own material.

There is a danger of popular research areas becoming a bandwagon that is driven by its own momentum rather than by the needs of real users. A particularly sad example of this was the 'extensible languages' bandwagon of the early seventies, which attracted large numbers of able researchers but, after a few years, had disappeared without trace. Hypertext is currently a popular research area, and we are all keen to make it lead, in due time, to systems that gain wide acceptance with users. This paper tries to make a contribution towards this by describing research ideas that have fed into a product that has had success in a mass market, and showing how the product has fed back into research.

HISTORY

I will start with an outline of the history of Guide, and its intended usage.

Guide began as a research project at the University of Kent at Canterbury in 1982. The aim was simple: to present documents on computer screens. At the time — and it is still largely true — most documents presented on computer screens were simply reproductions of paper documents. Such documents had all the disadvantages of paper, and some extra disadvantages too, such as an inferior image and a physical lack of flexibility. Not surprisingly, the users preferred paper. The aim of Guide was to break away from the constraint of paper and rethink from scratch how best to display documents on screens. In particular it was desired to take advantage of the high bandwidth of communication between user and computer offered by a graphics workstation. As a result, the user should be able to interact closely with a document and tailor it to what he wanted to read.

Note that Guide did not start as a 'hypertext' project. The focus was on an application rather than a mechanism. In the event the needs of the application have led Guide into including increasingly more hypertext features.

The first prototype of Guide ran under Unix on the ICL Perq in 1983, and development at the University of Kent has continued to focus on UNIX workstations.

A COMMERCIAL PRODUCT

In 1984, Office Workstations Ltd. (OWL) became interested in Guide, and decided to implement Guide as a commercial product on the Apple Macintosh. Turning a research prototype into a product requires a good deal of work, but in essence OWL's changes to Guide were:

- tailoring the user interface to the Macintosh house style.
- adapting Guide to fit in a smaller environment than the expensive workstation for which it had been developed.
- adding some features and cutting a lot out.

The co-operation between OWL and the University has continued successfully since. The purpose of the University's work, on its UNIX implementation of Guide, is to try out new ideas, and that of OWL to exploit these ideas — and indeed to enhance them with OWL's own ideas — and to respond to the demands of the marketplace.

In this paper I will refer to the two strands of development as *UNIX Guide* and *OWL Guide*. The OWL product is, incidentally, now available on an IBM PC, and is not confined to the Macintosh.

There has been a tendency for hypertext work to be based on expensive hardware, that precludes its use in any mass market for at least five years. This was true of the original UNIX Guide and a great achievement of the OWL work has been to show what can be achieved with more modest hardware. The Guide effort has been re-focused accordingly, and the resulting discipline has been wholly positive. Features that are profligate with resources have to be strongly justified.

USERS

A striking impression that comes out of Conklin's (1986) excellent survey of hypertext is the huge diversity of, on the one hand, hypertext systems and, on the other, potential application areas. Generally we have yet to see how systems and application areas match up, and in considering how this may happen it is useful to evaluate how programming languages match up with their application areas. The market for programming languages, being at least thirty years old, is a mature one compared with the hypertext market, and thus there may be lessons to be learned.

For example, attempts to build the ultimate programming language to cover all applications have failed to produce attractive products. The successful programming languages cover a market sector, which may vary from a wide sector like Fortran's 'scientific computing' to a narrower niche like SNOBOL's 'string manipulation'. You can, if you are sufficiently determined, use a language outside its intended market: you could use FORTRAN to write a payroll program or a string manipulation program. A few of these strays have been surprisingly successful and have pioneered applications that were never in the mind of the original language designers. The majority have, however, been disasters.

Some programming languages have succeeded because they were there at the right time to exploit new hardware/software advances. A good example of this is BASIC, which exploited interaction and was sufficiently small to run well on hardware that people could afford. As a result of this head start (and of successful design, even though computer scientists may not like it) BASIC has come to dominate a large market sector.

In the field of hypertext systems, different systems began with different applications in mind. Guide started out with the application of displaying documents ('browsing' if you like), whereas others systems such as Intermedia (Yankelovich *et al*, 1985) and Textnet (Trigg and Weiser, 1986) had critiquing applications in mind. Systems are now extending their horizons, and, as with programming languages, you can bend any system to any application if you try hard enough; we are now learning where the practical boundaries lie. Certainly Guide has had its surprises — mainly but not exclusively positive — from the marketplace. When there is a failure, a voice says: 'With these extra two features, Guide would become suitable for a new set of applications'. If the parallel with programming languages is correct, this is a siren voice. Presumably the original designers of BASIC resisted such siren voices.

CATERING FOR NAIVE USERS

As well as finding its application area, a hypertext system needs to determine how much sophistication its users need. There are, indeed, two classes of user: the end-user, and the author who prepares material for the end-user.

Guide is aimed at naive users, both authors and end-users, as any product aimed at a mass market must be. The number of systems that claim to be suitable for naive users is probably ten times greater than the number that really are. However Guide can point to some success in this area, and I shall now outline the design principles that led to this.

FUNDAMENTAL MECHANISMS

One way Guide has set about catering for naive end-users is by disguising the nature of the underlying data structures. In many applications the user can be totally unaware that the document he sees on the screen is made up of a lot of interlinked substructures. In particular the Guide user sees the document as a single scroll, rather than as lots of separate pieces of material scattered about the screen in separate windows, or in separate frames which appear one at a time on the screen.

The most important mechanism in Guide for exploring documents is the *replacement-button*. The replacement-button is a button within the document. It is an example of the 'embedded menu' described by Koved and Schneiderman (1985), and implemented in their TIES system. When selected with the mouse, a Guide button is replaced in-line by the material linked with that button (as distinct from TIES, which causes a new frame to replace the current one). Typically the author will present a document initially in summary form, with replacement-buttons to allow the user to expand the parts of the document that interest him. For readers of this paper who are unfamiliar with Guide, Figure 1 shows how a document may initially be presented. The replacement-buttons are shown in a bold font — the same font as the menu. In Figure 1 all the replacement-buttons have the label More, though the author could, if he chose, have given them different labels. Figure 2 shows the result of selecting the More replacement-button below 'Malaysia's MMC Metals' in Figure 1 (a close look at Figure 1 will show the cursor pointing at this).

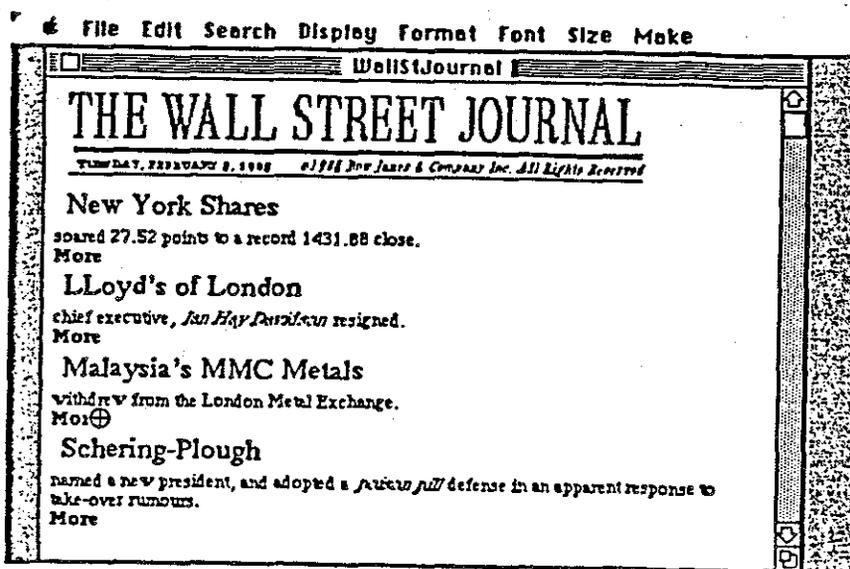


Figure 1: an initial view of a document

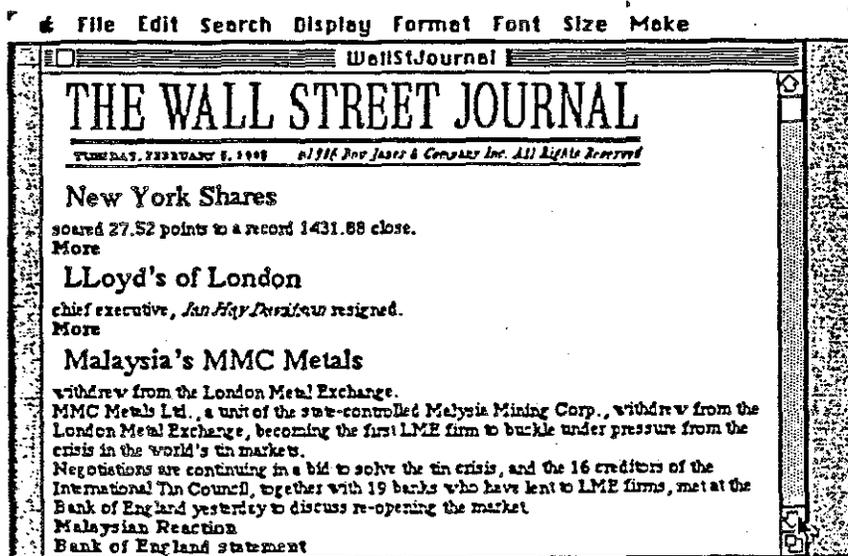


Figure 2: outcome from Figure 1 when the third More is selected

Typically the replacement of a button itself contains further buttons. A reader explores a document by successively expanding buttons, until he reaches the level of detail he wants. He thus tailors the document on the screen to his reading needs.

Sometimes the reader will wish to go back to a lesser level of detail. He can at any time 'undo' the replacement of any button previously selected, thus folding the replacement back under its original button. This saves screen space and generally makes the document more manageable and understandable. As the user sees it, this folding mechanism is particularly simple: if anything you see is at too great a level of detail you just point at it and click the mouse-button; the offending material is then folded back under a button. (This does not work, of course, if the offending material is at the top level of the document, and thus not part of a button's replacement. However if the reader cannot understand the initial top level description there really is something wrong — either with reader or author.)

Before folding occurs, the user gets feedback on what will be folded. Indeed feedback before the event is a feature of most Guide operations, as it is with much highly interactive software.

By using the replacement-button mechanism, documents can be presented in a form suitable for a wide range of readers. It would be wrong to claim, however, that by using these mechanisms it would be possible to produce, say, a single description of an aero-engine that was understandable by a Ph.D in aeronautics and a ten-year old. It is nevertheless true that authors can cover a reasonable spectrum of readers with one document, and that each reader can get what he needs out of it.

Other advantages of this in-line replacement mechanism are:

- it helps, as I have said, to disguise underlying data structures. The reader sees a document in terms of 'magic buttons' which can be expanded and contracted. He does not need to understand the computer scientist's concept of a tree.
- reversing actions is easy, and need not be done in the same order in which they were performed.
- all material is seen in context since it is replaced in-line.

ENHANCING REPLACEMENT-BUTTONS

Obviously replacement-buttons are not by themselves suitable for presenting all types of document. As described so far, they only cover tree structures, though a later section of the paper describes extensions to this. Guide therefore contains two further facilities concerned with buttons embedded within documents. One is the *inquiry*, which enhances the use of replacement-buttons, and the other is buttons that complement replacement-buttons by offering out-of-line replacement. Examples of the latter are *note-buttons* in OWL Guide (which cause the replacement to pop up in a separate window that remains in view only while the mouse-button is held down) and *glossary-buttons* in UNIX Guide (which display the replacement in a separate sub-window using a split-screen approach). Such buttons are shown in a different font from replacement-buttons since their behaviour, as the user sees them, is different. The term *poison pill* in Figure 1 is an example of this: when this note-button is selected, an explanation of the jargon term 'poison pill' pops up.

An inquiry consists of one or more replacement-buttons embedded in other text (and/or graphics), e.g.

Is it Red, Green or Blue?

If any of the buttons within an inquiry is selected, the whole inquiry is replaced by the corresponding replacement. This provides a mechanism for multi-way replacement, as the above example suggests, and more generally permits links to have regions rather than points as their source (following the terminology of Conklin's survey). The use of regions allows the author to explore different presentation styles. In particular if the inquiry is a screenful of information then Guide can be made to simulate frame-based systems.

A big advantage of the inquiry mechanism is that it gives the author a great deal more flexibility without adding any significant extra complication for readers: replacement-buttons within inquiries are used in exactly the same way as other replacement-buttons, and the reader can be largely unaware of the inquiry mechanism.

FURTHER MECHANISMS

I have argued that replacement-buttons (together with inquiries that contain them) contribute towards making Guide suitable for naive end-users. There are three other factors, which I believe, also make major contributions.

The first factor is the use of graphics and their close integration with text. Names of buttons can, for example, be graphical rather than textual. This allows many types of information to be presented in a much more attractive and readable way. (Though this paper has my name as author, many of the facilities described are as much the work of the OWL staff as me. In the case of graphics, the work exclusively came from OWL and I can claim no credit at all.)

The second factor is minimisation of facilities. If a Guide user needed to have to learn more than (say) five facilities before using it, then most users would give up. Guide end-users only need to master four facilities (though, to be exact, there is a bit of play in this figure since it depends on what you count). These are:

- travelling round the document using the scroll-bar. (On the Macintosh this mechanism will already be familiar to most users.)
- replacement-buttons.
- note-buttons (or glossary-buttons in UNIX Guide).
- reference-buttons. These, which only apply to OWL Guide, will be described later.

The greatest thrills in the development of Guide occurred when a single simple mechanism was found to replace several more elaborate ones. The four mechanisms described above have numerous ancestors.

The third major factor in making Guide acceptable to naive users is its authorship facilities. A fundamental principle of Guide is that *the author is the reader and the reader is the author*. The reader (i.e. end-user) can therefore freely edit the document, thus acting as author, and the *only* way an author can see a document is to see it as the reader does. Further details of this can be found elsewhere (Brown, 1986), but I certainly believe that close integration of authors with readers — a feature certainly not unique to Guide — is a key to success of

hypertext systems.

Lastly, it must be said that those of us who are technicians always attribute software's success — if it happens — to the excellent technical features. The truth is, of course, that marketing plays a large part, and it is even possible that success can be in spite of rather than because of technical features: a strong caveat to many of the comments in this paper.

RELATIONSHIP BETWEEN RESEARCH AND PRODUCT

Up to now, I have described and evaluated those design characteristics of Guide which, I believe, help to make it successful as a product as well as an interesting research exercise. In the rest of the paper I will explore the relationship between the research at the University and the product.

Researchers like systems built on pure concepts: a single simple approach that covers a variety of needs. Indeed I have already described instances of this. Products that are to succeed must temper this intellectual purity by adding some logically redundant features that users think they need or are familiar with. There are, for example, a lot of features that every user would expect to find in a Macintosh application, and would be upset if they were not there.

A particular instance of this occurs in the different linking mechanisms found in UNIX Guide and OWL Guide. The latter, to cater for non-hierarchical links, provides a conventional hypertext linking mechanism. This is the *reference-button*. When a reference-button is selected, it causes a jump to a different point of the document (or to a point in a new document). Such links are the essence of hypertext, but nevertheless have their dangers. Again taking a parallel with programming languages, a link is a 'goto' instruction. As the reader will know, 'goto' instructions flourished during the first fifteen years of the development of programming languages. They were then revealed to be criminals, and, unlike criminals in the real world, have now been successfully eliminated, at least in professional programs. Their sin was that they made programs structurally unmanageable; the sin was only revealed during the maintenance stage of very large programs.

Although it is wrong to base arguments solely on analogy, there is a real worry that uncontrolled linkages in hypertext documents may have a similar effect to gotos. Indeed they *are* gotos. Documents, like programs, need to be maintained, and doubtless for large documents, as for large programs, maintenance costs will dwarf all others. It is therefore worth looking ahead to try to find alternative, more structured, approaches to random gotos.

UNIX Guide has therefore (so far!) eschewed the goto and has instead extended the replacement-button mechanism to achieve the same purpose. This extended mechanism, the *copied-definition* mechanism, also attacks another problem with gotos in documents: if a link says 'see XXX' the interested reader should peruse XXX and then resume at the current point. The problem is that he has, in general, no way of knowing where to stop reading about XXX and return to the original. This is because the goto does not specify the scope of the destination. (In a hypertext system where the units are small, this problem is lessened.)

The copied-definition mechanism requires the author to mark as a *definition* any material that is gone to. A definition is like a replacement-button but has the added property that the button's replacement can also be used elsewhere. In any place where the author wished to go to the material associated with the definition, he inserts a *usage-button* of the definition. A usage-button, like a definition, behaves to the user as an ordinary replacement-button. Thus the reader might see:

... thus follows (see Lemma 2).

When the user selects the Lemma 2 usage-button, it is replaced in-line by a copy of the definition of Lemma 2. To him, the usage-button is absolutely identical to a normal replacement-button; he is unaware of the machinations going on behind the scenes. The single idea of the 'magic button' is made to encompass both tree-like hierarchical expansions, and cross-reference mechanisms associated with directed graphs. Moreover the usage-button has the further merit that the user reads its definition in the context in which it is used; the user is not required to move to a different part of the document or to another window,

This copied-definition mechanism, though both powerful and usable, is perhaps still not suitable for a mass-market product. There are problems for authors in distinguishing copies from originals, and there are performance problems with extremely large documents. Nevertheless, it might well lead to valuable results in the medium-term future, and provide a further example of seeds of research fruiting in the Guide product.

PURITY LOST

The above has described potential feeding of research into product. On the other side, the product has fed strongly into the research, mainly through the evidence of real users with real applications. I will take one somewhat lightweight example since it illustrates the clash between reality and the intellectual purity desired by researchers.

One of the original design principles of UNIX Guide was that, not only should there be no `gotos` within the document, but there should also be no 'Find' command (i.e. a command that searches for a given string). The Find command is, after all, a user-controlled `goto`. More seriously, there is a lot of wisdom in the statement 'You can only extract information from a database if you know that it is there' (quoted in the ZOG papers (Akscyn and McCracken, 1984, Robertson *et al*, 1981)). Thus users can only use the Find command effectively if they know the name of the word to search for. Given that they will not, in general, know this, it is better to encourage them to follow the discipline of exploring the document hierarchically through the use of replacement-buttons. With this discipline, assuming the author has done a good job with the button names, the user should be able to find the path to the information he needs. That, then, is the intellectual argument, and it is certainly not without merit.

OWL Guide includes a Find command, and there is no doubt that in a real world rather than an ideal one, it is valuable. An author may for example feed a Guide document to a spelling or style checker, and find he has made some mistakes. He then needs to find the errant words in the document. He may wish to change his terminology, replacing all occurrences of one word by another, or he may wish to find if a document contains any references to a particular term. UNIX Guide has therefore quietly introduced a Find command. The enemy, the `goto`, has gained a foot in the camp.

WORK FOR THE FUTURE

Of the software tools constructed by researchers, perhaps 98% never find any serious users outside the research group that created the tool. The most common reason for failure is that a tool is too complicated to use. Hypertext tools have no exemption to this rule, and UNIX Guide would have failed without the feedback from the OWL product. The biggest problem in hypertext systems, which most of us admit in footnotes towards the end of papers extolling the virtues of our systems, is of getting lost. This applies both to readers who follow links set by others, and, worse, to authors who need to create and modify links. There is thus a pervading need for navigation aids and also for checking aids that verify the validity of links. The need for such aids probably rises proportionally to the square of the document size.

Guide can certainly suffer from problems of getting lost. We like to think that some of the disciplines imposed by Guide, such as its linear scroll and its replacement-buttons, help to alleviate the problem, but nevertheless the problem remains. The solution may lie in extending the current scroll-bar to act as a simple map of the current document, and perhaps in providing more contextual feedback as the user explores a document. This is the subject of current research.

A second reason for failure is that a tool is an island to itself and cannot be combined with other tools. Those of us that expect the whole world to rewrite its documentation to fit the needs of our new hypertext system are unlikely to have our expectations fulfilled. Instead we must capture existing documents and have some way — even if crude — of automatically imparting structure to it. We must also work with existing tools such as spelling checkers or encryption programs. This is to some extent a research area but more, I expect, a question of curbing some of our wilder aspirations so that, following a recurrent theme of this paper, we fit the world as it is rather than the world as we would like it to be.

CONCLUSIONS

This paper has described a hypertext system built on a few simple, perhaps even simplistic, principles. The biggest advantage of this is that Guide is now widely used by real users with real applications. Their feedback will make it more likely that subsequent, more elaborate, features added to Guide are the right ones.

REFERENCES

- Akscyn, R.M. and D.L. McCracken (1984). *ZOG and the USS CARL VINSON: lessons in system development*, Carnegie-Mellon Technical Report CMU-CS-84-127.
- Brown, P.J. (1986). 'A simple mechanism for authorship of dynamic documents' in J.C. von Vliet (Ed.), *Text processing and document manipulation*, Cambridge University Press, pp.34-42.
- Engelbart, D.C. (1963). 'A conceptual framework for the augmentation of man's intellect', in Howerton and Weeks (Eds.) *Vistas in information handling*, Vol. 1, Spartan Books, London.
- Koved, L. and B. Schneiderman (1986). 'Embedded menus: selecting items in context', *Comm. ACM* 29, 4, pp.312-318.
- Robertson, G, D. McCracken and A. Newell (1981). 'The ZOG approach to man-machine communication', *Int. J. Man-Machine Studies*, 14, pp.461-488.
- Trigg, R.H. and M. Weiser (1986). 'TEXTNET: A network-based approach to text handling', *ACM Trans. on Office Systems*, 4, 1, pp.1-23.
- Yankelovich, N., N. Meyrowitz and A. van Dam (1985). 'Reading and writing the electronic book', *IEEE Computer*, 18, 10, pp.15-30.

**Applications
in the
Humanities
and Medicine**

Hypertext and Creative Writing

Jay David Bolter
Michael Joyce

University of North Carolina
CB# 3145 Murphey Hall
Chapel Hill, North Carolina 27599-3145

ABSTRACT

Among its many uses, hypertext can serve as a medium for a new kind of flexible, interactive fiction. Storyspace™ is a hypertext system we have created for authoring and reading such fiction. Interactive fiction in the computer medium is a continuation of the modern "tradition" of experimental literature in print. However, the computer frees both author and reader from restrictions imposed by the printed medium and therefore allows new experiments in literary structure.

THE IDEA OF INTERACTIVE FICTION

The idea of hypertext, which seemed daring only a few years, is now emerging as a serious and sensible way to use the computer for reading and writing. Technical writing and pedagogy (interactive communication between teachers and students) are obvious and important applications for hypertext systems. But hypertext may in fact apply to the whole range of human literacy, including the writing and reading of fiction. Using hypertext as a vehicle for fiction is both more and less daring than using it for technical writing or education. It is more daring because fiction seems frivolous in the pragmatic world of data processing. It is less daring because fiction, at least modern fiction, is by nature open to experiment, and being open or open-ended is precisely the quality that hypertext fosters in writing. The point of a hypertext is that it can change for each reader and for each act of reading. This flexibility can be exploited to make fiction interactive.

Interactive fiction has already existed for some time in the form of computerized adventure games. In an adventure game the player has a mythical world to explore -- a dungeon or an enchanted forest or valley. The computer describes the scene, and the player issues simple commands such as "go ahead", "enter the room", "pick up the dagger", "get gold," and the like. The goal is to amass treasure and dispatch monsters, although sometimes the game is more sophisticated, casting the player in the role of a detective who must

solve a murder or other mystery. Even the simplest of these games is a fictional hypertext. For the computer is presenting the player with a text, and the player's job is to understand and respond to that text. Depending upon his response, the computer presents more text and awaits a further response. The player, then, is an unusually powerful reader, whose decisions determine what text he will next see. Admittedly the text of the current games is simple-minded, but the method of presentation is not.

This method of presentation can now be applied to serious fiction. A printed novel presents its episodes in one order, but the computer removes that restriction. Instead of a single string of paragraphs, the author lays out a textual space within which his fiction operates. The reader joins in actively constructing the text by selecting a particular order of episodes at the time of reading. Within each episode, the reader is still compelled to read what the author has written. But the movement between episodes is determined by the responses of the reader, his interactions with or intrusions into the text, and the reader's experience of the fiction depends upon these interactions.

In its simplest form, interactive fiction requires only two elements: episodes and decision points (links) between episodes. The episodes may be paragraphs of prose or poetry, they may include graphic designs or pictures as well, and they may be of any length. Their length will establish the rhythm of the story -- how long the reader remains a conventional reader before he is called on to participate in the selection of the next episode. At the end of each episode, the author inserts his decision points -- a set of links to other episodes together with a procedure for choosing which link to follow. Each link may require a different response from the reader or a different condition in the computer system. The reader may answer a question posed in the text, and there will be one link for each possible response. The computer can also keep track of the previous episodes the reader has visited, so that he may be barred from visiting one episode before he visits another. Artificial intelligence experts would not consider such a simple scheme for interactive fiction worth pursuing. They would argue that we have to store knowledge representations in the computer and write a program that can generate new sentences in response to the reader's replies. In other words, the program itself would be the author, not simply the medium for delivering what the human author has written. While this AI strategy is interesting, it is not feasible at present or in the near future. No AI specialists can tell us how to store a world of knowledge in the computer; nor can their programs generate sophisticated English prose in response to queries by human users. For the foreseeable future, interactive fiction can only be a hypertext of prose written by human beings. There is in any case no need to wait for such breakthroughs in artificial intelligence. Even with the simple matching technique and the tracking of previously visited episodes, the author can create a fictional space of great flexibility.

Such electronic fiction is not automatic fiction. Since the computer does not create the verbal text, the locus of creativity remains with the author and the reader. Nor is electronic fiction necessarily random, for the author may put any number of restrictions on the reading order. The extent of the reader's choices and

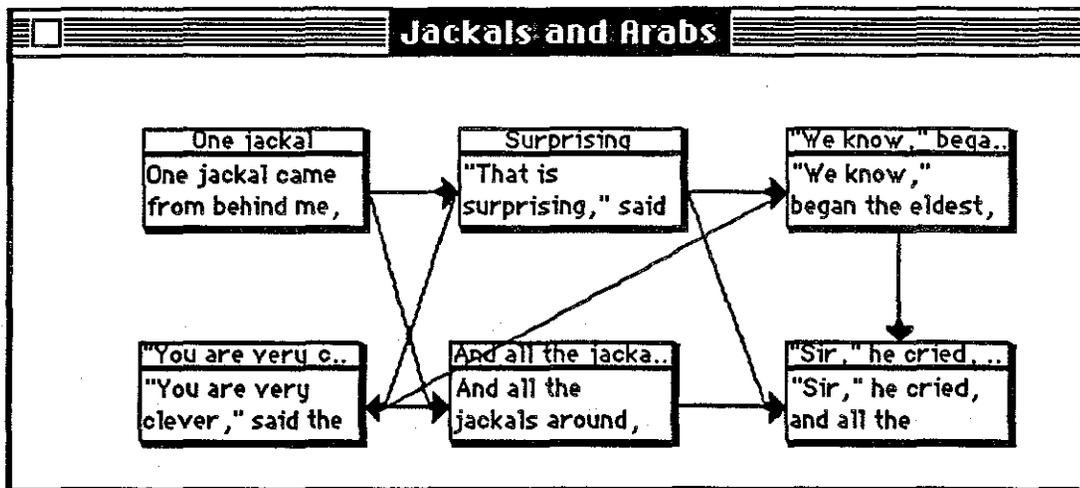
therefore his freedom in examining the literary space depend upon the links that the author creates between episodes. The reader may have to choose among a few alternatives or may range widely through the work. The author can relinquish as much or as little control as he chooses; he has a new literary dimension with which to work.

STORYSPACE

In collaboration with Professor John B. Smith, we have created a system for interactive fiction called Storyspace™, which implements the scheme of episodes and links mentioned above. (The system is by no means limited to fiction, although fiction will be the focus of the discussion here.) Storyspace has two modes: one for the author and one for the reader. The author creates his fiction as a series of textual episodes, using what we call a structural editor. This editor gives him a graphic or diagrammatic view of the hypertext he is creating. The reader has a different and more limited view: he sees the contents of each episode and may then reply by typing a string or pressing a button in order to branch to the next episode. Storyspace is implemented on the Apple Macintosh computer. This machine provides good graphics for the editing process. It is also popular and inexpensive, so that fictions created with Storyspace can be widely distributed.

The author constructs his fiction as a network of units. In the editor, the units appear as boxes on the screen. The author manipulates the structure by moving, adding, or deleting these boxes. He can also open each box to type in the text of his fiction. Each episode may be as long or as short as the author wishes: one word or several paragraphs. The end of each episode should, however, indicate how the reader is to respond: whether he should answer a question, make a choice, or simply hit return.

To indicate the relationships among these episodes, the author draws links, which appear as arrows on the screen. These links indicate possible orders of reading for the episodes, in that each arrow points to a possible next episode.



Each link carries with it a condition statement (specified by the author), which must be satisfied in order for that link to be followed. At present Storyspace recognizes two kinds of conditions. The link may require that the reader match a string (answer a question) before proceeding. The link may also require that the reader already have visited a particular episode before proceeding. The author can also specify Boolean combinations of these conditions.

While the author sees and manipulates a diagram of the evolving structure of his fiction, the reader sees only the text, not the structure of connecting links. The reader begins in one episode designated by the author as the starting place. He reads the text of that episode and may respond by typing text or pressing an appropriate button. The system processes the reader's reply simply by checking all the links that lead out of the current episode. It checks these links in the order of creation and takes the first one whose condition is satisfied. It then displays the text of the destination episode and awaits a further reader response.

INTERACTIVE FICTION AND THE EXPERIMENTAL TRADITION

Storyspace, then, is a simple system for hypertextual fiction, but even this system permits significant structural experiments. Its use of episodic branching challenges many conventional ideas about literature. If the reader is allowed to choose his path through the narrative, then the stability and certainty inherent in a printed text disappear. There may no longer be one plot, but several, and characters may no longer develop in a consistent fashion. The structure and rhythm of the text will be different for each reading. Every element of fiction is subject to electronic fragmentation and recombination. At the same time, by disrupting the stability of the text, interactive fiction belongs in the tradition of experimental literature (if I may use this oxymoron) that has marked the twentieth century -- the era of modernism, futurism, Dada, surrealism, letterism, the nouveau roman, concrete poetry, and other movements of greater or lesser influence. The experiments of Dada, for example, were aimed at breaking down all structures of established

art and literature, and in that breakdown some of the Dadists worked in the same spirit as writers and readers may now work in the electronic medium. Jean Arp wrote that in his poems: "I tore apart sentences, words, syllables. I tried to break down the language into atoms, in order to approach the creative." [Gros71, p.136] Tristan Tzara proposed a poetics of destruction, when he gave this advice for creating a Dada poem: "To make a dadist poem. Take a newspaper. Take a pair of scissors. Choose an article as long as you are planning to make your poem. Cut out the article. Then cut out each of the words that make up this article and put them in a bag. Shake it gently. Then take out the scraps one after the other in the order in which they left the bag. Copy conscientiously. The poem will be like you..." [Gros71, p. 125]

Dada is an early and influential example of the modern will to experiment. The modern attack has often been aimed at the conventions of the realistic novel, the nineteenth-century novel that told a story with a clear and cogent rhythm of events, and in the course of their attack modern authors have often found themselves straining at the conventions and limitations of the printed page. Because the linear-hierarchical presentation of the printed book was so well suited to the conventions of plot and characters of the realistic novel, to attack the form of the novel was also to attack the technology of print that helped to shape that form. The French often led the way with the *nouveau roman* and Philip Sollers and the *Tel Quel* group. From France and elsewhere, we have had programmed novels and aleatory novels. All these efforts were instances of subversion: they worked from within, attempting to undercut the conventions of printed literature while themselves remaining printed books. Subversion is an effective mode of attack precisely because of this irony -- because in this way the printed novel is made to contain the seeds of its own destruction, or perhaps deconstruction.

Indeed, much important twentieth century literature may be, and has been, accused of subversion. The avant-garde movements like Dada were never so radical as they claimed to be; they were instead extensions or perhaps caricatures of the mainstream. Joyce, Virginia Woolf, Pound, Eliot, and others all participated in the breakdown of traditions of narrative prose and poetry; breaking with such traditions was the definition of being modern. Pound and Eliot set about to replace the narrative element in poetry with fragmented anecdotes or mythical paradigms. Joyce and Woolf called into question the strategy of the novel as a linear and objective narrative. They devised new ways of structuring their works based upon stream of consciousness (Woolf) or upon multiple layers of topical and mythical organization (Joyce). All of these writers were trying to set up new relationships between the moment-by-moment experience of reading a text and our perception of the organizing and controlling structures of the text. In this sense, hypertextual fiction is a natural extension of their work, redefining the tradition of modernism for a new medium.

BORGES

One contemporary writer whose work is very suggestive of interactive fiction is the Argentinian Jorge Luis

Borges, whose short stories called Ficciones are a series of meditations upon writing or more broadly upon the human capacity to create and comprehend symbols. Borges writes tiny pieces without much plot or characterization -- pieces that are utterly insignificant by the standards of the nineteenth-century novel. Often these pieces concern the problem of time and writing: Borges is intrigued by the fact that a frozen text cannot change to reflect possibilities that unfold in time. "An Examination of the Work of Herbert Quain" is the literary obituary of a writer who tried to liberate his texts from linear reading and static interpretation. Quain's work *April March* is nothing less than an interactive fiction. It consists of thirteen chapters or sections representing nine permutations of the events of three evenings. The novel is therefore nine novels in one, each with a different tone. Borges tells us the work is a game. He adds that "[w]hoever reads the sections in chronological order... will lose the peculiar savor of this strange book." [Borg62, p. 76] He even gives us a tree diagram of the ternary structure of the work, and another diagram of the binary structure that Quain later says he should have written.

Borges' longer and more elaborate "Garden of Forking Paths" is a detective story. At its center the story contains a description of a Chinese novel, a novel that seeks to explain and in its way to defy time. It was thought that the author Ts'ui Pên had retired from public life with two objects: to write a book and to build a labyrinthine garden. In fact Ts'ui Pên had only one goal, for the book was the labyrinth. The manuscript Ts'ui Pên left behind was not, as it seemed, "a shapeless mass of contradictory rough drafts," [Borg62, p. 96] but instead a ramifying tree of all possible events. "The Garden of Forking Paths is an enormous game, or parable, in which the subject is time." [Borg62, p. 99] Ts'ui Pên "believed in an infinite series of times, in a dizzily growing, ever spreading network of diverging, converging and parallel times." [Borg62, p. 100]. In the end "The Garden of Forking Paths" is no more serious than any of Borges' fantasies. Borges' point is not to sketch a new philosophy of time, but a critique of writing itself -- to imagine a book that calls into question the finite, fixed character of writing or print. Of course Borges' fiction is also a book, as he realizes, and therefore subject to the same limitations that he is describing. His understanding of these limitations may explain why Borges himself only envisions experiments rather than undertaking them. The Ficciones are themselves conventional pieces of prose, essays or stories meant to be read page by page. Yet the works he describes, the novels of Quain or the "Garden of Forking Paths," belong in another writing space altogether. Borges never had available to him an electronic writing space, in which the text can constitute a network of diverging, converging and parallel times. The literature of exhaustion in print by no means exhausts the electronic medium. In fact, a number of Borges' pieces suggest themselves for translation into the computer's writing space. The "Garden of Forking Paths" has been converted into an interactive fiction by Stuart Moulthrop of Yale University. This Storyspace module contains a web of over 100 units and 300 connections. Moulthrop has added his own meditations to those of the Borges' story.

THE STRUCTURE OF INTERACTIVE FICTION

Like Borges, many experimental writers have concerned themselves with the problem of writing or printing: their concern is shown by the difficult relationship between the narrator and text and between the text and its reader. Hypertextual fiction has much in common with these experiments in print. It too will have to introduce new procedures of reading that violate the reader's expectations of a linear narrative. (Perhaps the first indication that electronic fiction has matured beyond the adventure game will be the appearance of fictions that are about writing, about capturing experience in writing, and therefore about themselves.) Electronic writers therefore need conventions, genres, traditions by which their medium can be governed. They must find new ways to maintain a tension between the reader and the text. The source of that tension will surely be the participation of the reader in making the text. In electronic fiction, the struggle between author and reader to appropriate the writing space can become visible, as the reader admits or tries to avoid admitting particular elements into his particular reading of the text. Moreover, electronic authors will need a new concept of structure. The structure of an electronic fiction will change with each reading, because the order and number of episodes will change. Authors must therefore learn to conceive of their text as a structure of structures, and this is a concept that is new in the history of literature.

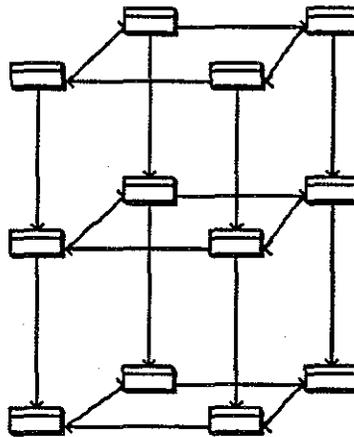
The temporal character of interactive fiction is also something new. In printed fiction the author is free to manipulate the time in which his story takes place, and every good author does so. However, the plot, the author's manipulation of story time, is itself static. Printed fiction is one-dimensional in the sense that we need only one dimension in order to represent the experience of reading it. The episodes (chapters, sections, cantos, books, volumes) are realized through time as we read. The links between the episodes are fixed in the course of writing, and the reader has no obvious and effective way to alter the order of reading. In electronic fiction multiple links among episodes allow our temporal experience of the plot itself to vary. Time may be fluid in a printed novel, but the presentation of time is fixed, as the fixed pages of the book mark the progressive stages of the narrative. The author manipulates words to create a single narrative structure. The author writes with words, not with structures. The electronic medium permits writing of a second order, a writing with narrative units, in which the structure of the text becomes truly fluid and indeed geometric. The author becomes a geometrician or architect of computerized "space" (as computer memory is in fact called by programmers); he fills his space with a special pattern of episodes and links that define a kaleidoscope of possible structures. The success of his work will depend upon the poetic rightness of the way in which the pattern is realized in the act of reading.

Storyspace provides the author with the opportunity to see the structure of his tale and therefore to use the structural geometry as an aesthetic principle. Because the reader does not see the diagrammed structure of the text, he is left to gain an intuitive sense of the structure by reading the episodes themselves. He might have to read the tale many times to understand a structure that changes, in a controlled fashion, with each

reading. Such a reader is like a mathematician who attempts to envision a four-dimensional object by looking at several projections in three dimensions: each projection is a snapshot, and all the snapshots must be synthesized to win a sense of the whole, if indeed such a sense is possible. The synthesis of many readings will be the ultimate experience that the electronic writing offers its readers.

Multiple reading does not necessarily mean multiple plots, although the most obvious way to construct a hypertextual fiction is by presenting the reader with choices that affect the plot. Hypertext could borrow from modern literature the techniques of stream of consciousness or multiple points of view as methods of organization. An electronic author could, for example, create rings of episodes representing the same events as told by several characters. The technique of telling the same events from different points of view is familiar from Faulkner's The Sound and the Fury and Darrell's Alexandria Quartet. In exploring these multiple points of view, the reader of a hypertext can enjoy far more freedom than the reader of a printed text. He can move back and forth through the episodes, comparing one narrator's version with another. Even visiting the same episode twice is not mere repetition, because his experience of the narrative will be affected by the other episodes he has recently read.

Here is an example of a narrative hierarchy composed from the ancient myth of Oedipus, as elaborated by the Greek playwright Sophocles. The three levels correspond to three points of view from which the story might be told. The lowest and most confused version takes the perspective of the shepherd who gave the baby Oedipus away rather than obey the order to kill him. Years later, this kindly shepherd is brought before Oedipus the king, threatened with torture, and made to reveal what he did with the baby. Then he is released and left to wonder at the extent of the horror that he has revealed. Oedipus himself provides an heroic perspective; he is of course most directly affected by the horror of the events. The third and clearest perspective is the divine view of these view: the story as the god Apollo might tell it. This third perspective is one that Sophocles himself would never have permitted, for Sophocles' gods, unlike those of Euripides, had to remain dignified and remote, their ways inexplicable and terrifying.



The reader begins at the shepherd's level, and his task is to break into the higher levels representing greater understand of the events. The divine level is of course harder to reach than the heroic. It is also a detached view the reader himself may find repellent. Unlike the electronic version of Dungeons and Dragons, the electronic Oedipus is a game in which winning is not clearly defined. (One of us, Jay David Bolter, has discussed the Oedipus example, other possible geometries, and the idea of interactive fiction as a game in greater detail elsewhere [Bolt85]).

All electronic literature takes the form of a game, a contest between author and reader. Unlike the static and monumental character of printed fiction, hypertextual fiction is characterized by impermanence and a lack of monumentality. A playful attitude prevails, as it should in any computing task. An interactive fiction is, after all, a program that the author creates and the reader executes, and any computer program can go unexpectedly and ridiculously off track. Fortunately, with most programs, it is possible to restart the system and try again. The reader of an interactive fiction can return to the starting episode and take another path. This flexibility of interactive fiction stands in sharp contrast to the solemn rigidity of the printed book, which cannot change in response to its readers' previous experience. No matter how often the reader returns to the first chapter of a printed book, he still has only one path to choose. Printed fiction is proud of its rigidity, and hypertextual fiction, which cannot rival this quality of the printed book, must therefore draw its aesthetic strength from its capacity for change.

[Storyspace is available to creative writers on an experimental basis. One of us (novelist Michael Joyce) is currently working on interactive fiction using this system, and his first effort, "Afternoon, a Story" is available to interested readers. The same methods of editing and presentation can be used for nonfiction and particularly for teaching. A group working at the University of San Diego under the direction of Professor Bart Thurber is presently considering an interactive course using Storyspace as an interdisciplinary introduction to the humanities.]

REFERENCES

[Bolt85] Bolter, Jay David. "The Idea of Literature in the Electronic Age," Topic: A Journal of the Liberal Arts, 39 (Fall, 1985), 23-34.

[Borg62] Borges, Jorge Luis. Ficciones, edited with an introduction by Anthony Kerrigan. New York: Grove Press, 1962.

[Gros71] Grossman, Manuel L. Dada: Paradox, Mystification, and Ambiguity in European Literature. New York: Bobbs-Merrill, 1971.

From the Old to the New: Integrating Hypertext into Traditional Scholarship

Gregory Crane

Co-Director, Perseus Project
Assistant Professor of Classics
Harvard University

ABSTRACT

Hypertext allows academics to structure and manipulate their ideas in a radically new way, but it should also reinforce traditional scholarly activity. Those designing Hypertext systems that are intended for the general academic market must be careful to support not only new possibilities, but those functions with which academics are already familiar. Further, many scholars hope that their documents will be useful for decades to come. We need standard document architectures that will separate a particular Hypertext from the system in which it was designed.

PERSPECTIVE AND PURPOSE OF TALK

This paper differs from that of many in that it is only secondarily interested in the issues of software design. This talk will outline some of the things that a hypertext system should be able to do if it is going to exert the widest possible appeal among scholars in the humanities. Rather than addressing those problems that are most challenging or appealing from a technical point of view, it will pursue those features that many academics would like to see implemented.

I am, therefore, going to outline a set of priorities which may differ from those of you who are software designers might assign yourselves. These priorities may seem to you to be distractions, or to pose problems intellectually less appealing than those which you would normally approach. But if you can reconcile at least some of these priorities with your own interests, you will expand the basis of your support, increase the resources at your disposal, and ultimately make greater progress than if you devote all of your attention to those problems which seem most challenging to you and your immediate colleagues.

My own field bears the elegant name "Classical Philology." I attempt, primarily through the study of texts, to reconstruct the ancient Greek and Roman world. I happen to be a moderately conservative scholar, but the materials that I use and the methods that I apply often do not differ markedly from those of my more avant-garde colleagues. We work inductively, examining in considerable detail, a wide variety of texts. Any statement that we make should derive from a close examination of the primary material. Some literary critics will discuss a single work or a group of works with little reference to texts by other authors. Others, more historically minded, will incorporate evidence from a wide variety of sources in order to discuss a single play or poem. Historians will discuss inscriptions as well as texts, and students of Greek religion will dwell heavily upon archaeological material. Whether we are deconstructing a fragment of Sappho, or writing a commentary on the seventh book of Herodotus, we build our arguments upon textual and visual materials. Primary sources are precisely that, the material to which all other conclusions are secondary. This holds true not only for most Classicists, but also for many, if not most, Historians, Literary Critics, or general students of any period or culture.

Hypertext systems offer substantial promise to such traditional scholars, but software alone is not enough. An empty hypertext may become a tool for composition or collaboration, but hypertext will not have its full impact within the academic world until it becomes a medium for publication. Unless contributions to a given hypertext carry the same weight on a scholarly CV as a traditional paper publication, few scholars are

going to make such contributions, and we will consequently have little material available to us in the new medium.

On the other hand, a hypertext will only count as a traditional publication when members of a particular discipline feel that, in order to do their own work, they must use material within that hypertext. Even those who have worked extensively with hypertext systems (such as Professor George Landow of Brown University) caution that such work is time consuming, and should be left to those who have already established themselves. Relatively few will, therefore, invest money in hardware and software, or put time into learning how to use that hardware and software, until they have some compelling reason to do so. We have, then, a "chicken and egg" problem: we won't develop hypertexts as enthusiastically as we should until considerable information is already available in some hypertext, but we aren't going to have very much information until we are already using hypertext systems.

The problem is particularly acute for many traditional scholars. Some disciplines (such as medicine) change so rapidly that information has a very short useful life. In a relatively few years, all current medical textbooks will be replaced, and new articles will become old. If the medical world begins to place its information in a hypertext, all crucial information will, within a few years, be stored in this format.

With many traditional disciplines, however, the rate of change is not nearly so quick. Basic classical texts and commentaries may be edited once a generation, and this is true for source works in many disciplines. How often do scholars reread the wartime papers of Robert E. Lee, or the correspondence of Thomas Paine? Humanists in particular, even as they argue fiercely among themselves over their methodologies or conclusions, often assume that the basic material with which they work is relatively stable. Classicists are in an especially enviable situation, because the *Thesaurus Linguae Graecae* at Irvine has put much of Greek literature online. But even if all Classicists began using some standard hypertext today, they would have to wait quite a while before scholars had recreated all of their commentaries and standard reference works in an electronic form.

The Perseus Project is attempting to break through this barrier. We hope to enter into an electronic format a critical mass of information about the Classical Greek world. Ultimately we plan to enter about 100 megabytes of textual information and 10,000 images. This material, disseminated cheaply on optical disk, will be accessible to individual students, and will serve as the basis for courses on Greek Art and Civilization.

Some of the material (such as a 50,000 word dictionary and reference grammar, maps and various images) will, however, also be useful to professional scholars. If the Perseus databases can become a standard reference tool, then subsequent contributions to Perseus will become a new means of publication. We thus plan that Perseus should become, rather than a static resource, a dynamic entity that attracts further scholarly work and will be updated on perhaps a yearly basis.

The task is, however, forbidding. We must somehow take a heterogeneous selection of material and make it useable in an electronic format. We must smooth the path from the printed to the electronic medium, so that documents created in a print environment can be efficiently entered and integrated into a larger hypertext. Upward compatibility is the real issue, but it is as important for the humanist as for the software engineer. A trail needs to be blazed which traditional scholars can follow as they carry their information from the printed text to the electronic medium. This trail will ultimately lead to new tools for storing scholarly information (such as hypermedia, with resources such as animation and sound supplementing printed text). But few will follow this path until humbler, but more immediate issues have been addressed. Few indeed will embrace the new, if they cannot take the old along with them.

TRADITIONAL LINKS

Hypertext appeals to many not because it is so radically new, but because it promises to allow them to do exactly what they always have done, only better. Scholars have always pursued links that carry them from text to text to image and back again to text. Even the print-based tools currently available have evolved over a long period of time to facilitate this process.

Dictionaries in antiquity were, for example, sorted only according to the first letter. With an equivalent English dictionary, you would know that the word "streetcar" appeared somewhere in the list of words beginning with "s", but you would have no precise idea of where in that rather lengthy list the word appeared. Even well into the nineteenth century, when German scholars had developed "modern scholarship" as we know it, references were often vague or imprecise. Author A might refer in only a general way to some statement made by author B, without even bothering to list the particular work of author B in which this statement was made. Or a scholar might cite an ancient source, but give no line number, leaving the reader to thumb through sixty pages of a Greek play for a single quote. The bibliographic and citation schemes that we, all of us, from Classicist to American Historian, represent a long refined system developed by generations of academics.

Simple text strings such as "Herodotus 7.132" or "Euripides *Medea* 335-337" are, therefore, self-standing links, and have certain advantages. First, they contain valuable information lacking in a more generic kind of link (e.g., a button that identifies a cross reference, but does not indicate what the cross reference points to). The object "Herodotus 7", for example, means "the seventh book of the history of Herodotus". Anyone already familiar with Herodotus knows that this covers the first portion of Xerxes' invasion, so the scholar has some general idea of the context before he or she pursues this link. Likewise, the mere fact that a link point to Euripides (as opposed to Aeschylus or Plato) may determine whether that link is worth pursuing or not.

Second, and far more important, links such as "Herodotus 7.132" or "Euripides *Medea* 335-337" are standard. Whether I am using Gilbert Murray's text of Euripides or the new edition of Diggle, the phrase "Euripides *Medea* 335-337" will designate the same passage. Citation systems differ slightly—someone may abbreviate the reference as "Eur. *Med.* 335" or "Euripid. *Medea* 335." Any system that analyzes such references would have to recognize many slight variations, and standardizing material will require substantial editorial effort. Overall, however, standard citation schemes offer a firm foundation on which to build.

Now I realize that there are problems with references of this type, or, perhaps more importantly, that references of this type lack problems. Most Greek texts are, in fact, quite stable. It is computationally straightforward to build tools that will call up passages according to conventional chapter and verse schemes. Archival indices can be built up, and the systems that maintain these archives, though they will have to accommodate the periodic addition of new texts, will not have to manage a complex stream of real time editorial changes.

Someone building up a document of his or her own, or several people collaborating on a project, will have to create links dynamically. Nor will they easily be able to create pointers to "line six of paragraph fifteen," since the original "paragraph sixteen" could be radically changed or deleted altogether. A system that could manage X links as Y users performed Z editorial operations on a Hypertext would be extremely powerful, and effective work has been done in this direction, particularly by the Intermedia Project at Brown.

Still, the humble archival lookup problem confronts many of us every single day. Suppose I read in Walter Burkert's *History of Greek Religion* [Burk85:163] the following statements:

As god of wine, Dionysos is a delight to mortals and a giver of joy, *polygethes*. He stills all cares and brings sleep and oblivion of daily ills.

I can, of course, simply take these statements at face value, and in some circumstances I would. But if I am particularly interested in Dionysus or in these aspects of Dionysus, I would examine the three texts cited in the notes: Il. 14.325; Hes. *Erga* 614; Eur. *Bacch.* 280-2. I may, of course, find that these passages simply backed up Professor Burkert's statements. But often enough I will find that my view of the primary source differs from that of the scholarly authority commenting on it: I may decide that Burkert had read more into these passages than I feel proper, or that there is more in these passages than he had seen, or these passages might suggest something about Dionysus that was entirely new. Such questions lead to answers and more questions. They provide much of the excitement that many of us find in pursuing scholarship. But such questions are not limited by purely intellectual concerns. Logistics often play far more of a role than we like to admit.

A single page of Walter Burkert's *Greek Religion*, for example, contains pointers to more than two dozen source texts of Greek literature. If I am to assess his conclusions with any thoroughness, I should look at most, and ideally all, of these texts. The pure mechanics of this task are depressing. Even if I have the relevant book on a shelf in my office six feet away from my chair, it still takes me about 15 - 20 seconds to get up, find the book, and open to the proper reference. It then takes me another 15 or so seconds to put the book back on the shelf. Of course, I may spend some time on this passage, but most times I will probably glance at it briefly, see that it has nothing of any particular interest and move on to the next. Thus, to check the primary references of a single page, I must, under optimal conditions, spend perhaps ten or fifteen minutes walking back and forth in my office. Not only is this relatively time consuming, but visually and intellectually disruptive as I must distract my concentration from the material in front of me so that I can keep from tripping over the furniture as I cross the room.

Optimal conditions, however, rarely prevail. Less than half of the references on a given page of Burkert's book, for example, are in my office. Pursuing much of the critical material will force me to walk over to the library, and then hunt back and forth around the stacks. Here, optimal performance degrades from seconds to minutes for each lookup, and the overall time required to ponder the evidence for a single page may swell from minutes to hours. And again, optimal performance is unlikely, as at least some of the books are liable to be absent from the stacks.

Purely mechanical factors of this type may perhaps not shape the kind of research that I could conduct, but they certainly shape the kind of work that I actually do conduct. The more time it takes to poke into the background of a statement, the less I shall do such poking, and the weaker my understanding of the issues involved will be. My scholarly training urges me to pursue the evidence as far as I possibly can, but the dull limitations of time and space hold me back and prevent me from pursuing this goal as fully as I would like.

A fairly simple system could dramatically improve this situation. It takes about five seconds for me to select a passage with a mouse and then to pull down and select a menu option. If I could reduce the time required to pursue any particular link, then not only can I pursue more links, but I will even derive more pleasure from pursuing links. Thus, in the world of paper, when a scholar works in a major research library, he or she finds that work becomes not only more efficient, but more stimulating as well. Generally speaking, the easier it is to answer a question, the more different questions can be asked, and the easier it is to become intellectually engaged. This is one reason why the expert derives so much more satisfaction than the novice.

A hypertext system could automate much of the work, allowing the scholar to have flexible access to vast amounts of data stored on optical media. Clearly, we need the ability to take existing source and reference material, and to convert this as mechanically as possible into an electronic format. But to do this, we first need a standard form in which we may, as efficiently as possible, store this material. Nor is it enough to observe that a given hypertext system does support a document architecture flexible enough to perform such traditional functions. This has implications for the way hypertext systems are designed. No system is going to do a good job managing several hundred megabytes of data stored on a compact disk, unless someone has made sure from early in the design phase that the system would do just that.

A scholarly hypertext system should, therefore, be built on top of an efficient document management system. This document management system should be able to address texts using canonical citation systems (e.g. "Herod. 1.38", "Shakespeare *Macbeth* 3.2.30", "*Greek Roman and Byzantine Studies* 1987 33-35"). While a hypertext system should be able to manage a certain amount of dynamic information, much of the source material within the hypertext will be static data, and can thus be stored on inexpensive optical media.

LONG TERM STANDARDS

Those of us who work with fairly stable source materials look not only backwards, but far into the future as well. Any text that we edit or any commentary that we write should normally be useful for a generation.

But how are we to develop academic hypertexts, if the tools in the hypertext system tie that information to a particular software (or even hardware) platform? If we build our commentary around system X, what

happens when system X becomes obsolescent ten years from now, and the new systems that have taken its place do not quite provide upward compatibility? Most people who have spent five years building a major new piece of scholarship do not want to have to keep rewriting it every few years just so that it will fit some new system or another.

So, even those of us who preach to our colleagues about the bold new electronic world, are reluctant to design our work for this year's leading hypertext system. We need standards in which we can trust and around which we can build our work. Until we have defined the basic kinds of links that will be supported over the long term, hypertext will be an intellectual siren, luring the curious to an environment that will ultimately destroy their work.

Of course, we will want to develop sophisticated kinds of links that take full advantage of the electronic medium. But traditional methods for linking texts, being independent of any particular hardware or software base, could provide a starting point. Of equal importance, standards built upon such links would be easier for many scholars to understand. Familiar problems will provide a gradual introduction into the more general possibilities of hypertext. More than one scholar, initially attracted by conventional problems, will also begin to understand some of the more radical opportunities presented by hypertext.

EXTENSIBILITY: PLATFORMS RATHER THAN CEILINGS

A final suggestion only affects the average scholar indirectly, but its importance cannot be overemphasized. Some of the links in a hypertext will be implicit and domain specific. These links will require specialized programming that can only come from someone who understands the discipline in question. A new kind of scholar will have to meet this particular need, one who is at home with issues of both programming and of his or her discipline.

Consider, for example, a problem that we face in building a hypertext for Classical Greece. The user will be able to call up Greek texts, and will have an online Greek-English Dictionary. But Greek morphology can be very complex, and the intermediate student of Greek will not always realize that the form "*enenkontes*" comes from the verb "*fero*". Likewise, if a specialist wants to study the underlying idea of the verb "*fero*," he or she will not want to type in the several thousand odd forms that this verb could generate. The user will want a system that can cope with the idiosyncracies of Greek morphology automatically.

Greek morphology is, however, a non-trivial subject. Not only can a single verb generate thousands of forms, but each form must obey complicated, but precise rules of accentuation. Furthermore, Greek contains various dialects, each of which uses different endings and stems. Only someone proficient in the language could design a system that could cope with all the problems that this language posed. We are, therefore, building software tools that can cope efficiently with Greek morphology.

On the other hand, there is no reason to build a separate hypertext system just to deal with Greek. We should be able to write a module for Greek morphology in some general programming language (e.g., C, Lisp), and then be able to add this function to an existing hypertext system.

Greek morphology may only disturb a narrow group of users, but many users will find that their discipline requires specialized functions not included in any generalized hypertext system. The businessman will need to incorporate a spread-sheet, while the medical doctor may need access to a decision support system to help with his or her diagnoses. We, therefore, need a platform on which we can build, rather than a completely self-contained application. A hypertext is, or should be, a beginning rather than an end.

REFERENCES

- [Burk85] Burkert, Walter, *History of Greek Religion*, Harvard University Press, 1985.

Searching for Information in a Hypertext Medical Handbook

Mark Edwin Frisse, M.D.

Washington U. School of Medicine
660 S. Euclid Avenue
St. Louis, MO. 63110

ABSTRACT

Effective information retrieval from large medical hypertext systems will require a combination of browsing and full-text document retrieval techniques. Using a prototype hypertext medical therapeutics handbook, I discuss one approach to information retrieval problems in hypertext. This approach responds to a query by initially treating each hypertext card as a full-text document. It then utilizes information about document structure to propagate weights to neighboring cards and produces a ranked list of potential starting points for graphical browsing.

A DYNAMIC HANDBOOK OF MEDICAL THERAPEUTICS

It is early evening in a busy hospital emergency room. A young insulin-dependent diabetic man awaits treatment. His wife states that for the past 3 days he has been experiencing fevers and a productive cough. He has been "too sick" to manage his diabetes properly. He is allergic to penicillin. On examination, the patient is comatose, has a fever, and is breathing rapidly. His mouth is dry and blood glucose concentration is dangerously high. His chest x-ray shows a pattern characteristic of pneumonia, and microscopic examination of his sputum shows a bacterial cause. "Diabetic coma precipitated by pneumococcal pneumonia," his physician mutters. Immediate action is required. A number of obvious diagnostic tests come to mind: arterial blood gases, serum electrolytes, sputum cultures. Therapeutic issues quickly follow; intravenous hydration, electrolyte management, insulin therapy, antimicrobial administration. Having studied extensively, and having treated several patients with similar conditions in the past, the physician is well aware of the issues regarding the patient's care, but details are lacking. How much intravenous fluid should be administered within the next hour? The patient's coma can be explained by his diabetes, but should the physician order a radiographic study of the patient's brain in order to eliminate the possibility of a stroke or tumor? How much insulin should be administered? Wasn't there a recent review on intravenous insulin therapy in diabetic ketoacidosis? How did the physician treat the last patient she saw with this disorder? What is the antibiotic of choice in this penicillin-allergic patient?

This scenario exemplifies the information-management dilemma faced by medical students and practitioners. Their repertoire of knowledge and skills allows much of their daily professional lives to go by uneventfully. All too frequently, however, they are confronted with problems for which they have the expertise to interpret observations, but they lack the detail required for action. The information they require usually resides somewhere in our vast corpus of biomedical literature, but it usually is difficult to locate and, when found, it generally is not structured in a manner that facilitates solving the specific problem confronted by a medical practitioner.

We view the physician's search for expertise and detail as one example of a conversation for action [Wino86a]. Traditionally, medical conversations have been constrained by the written word and by oral communication. Medical students, for example, continually record valuable clinical heuristics that they obtain through conversations with their teachers and colleagues. They often call the notebooks in which they record these conversations their "peripheral brains." Although peripheral brains and other conventional media facilitate medical communication, their limitations impede medical research, teaching, and practice [Warr81,Aamc86, Cove85].

The fundamental limitations of "peripheral brain" methods of information-management led Vannevar Bush to conclude that technical fields required a new medium in which to represent their discourse. He called this new medium a "memex", and defined it as "a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory [Bush45]." Bush's memex is a generalized notion of the medical student's peripheral brain.

At about the time Bush was writing his landmark paper, the Department of Internal Medicine at Washington University published its first therapeutics primer for medical students and interns. Unlike conventional medical textbooks, this primer emphasized specific guidelines for medical diagnosis and therapy. Over the last several decades, the Washington University Manual of Medical Therapeutics has evolved from a small, mimeographed document to a 500-page institutional benchmark for patient care standards in internal medicine [Orla85]. The Manual has become one of medicine's most popular books. It has been translated into many languages and is used on a daily basis by tens of thousands of students and physicians throughout the world.

As occurs in other medical peripheral brains, physical limitations force the Manual's editors to compromise between generalization and detail. As a result, the book often lacks the specific information required to address unique medical problems.

The Dynamic Medical Handbook Project applies the memex concept to medical information-management problems. First, we are studying how computer tools can facilitate the work of the Manual's authors and editors. Our concerns in these areas focus on the use of electronic mail, document-processing programs, outline organizers, hypertext systems, and bibliographic-retrieval programs. Second, we are studying how users can retrieve information from dynamic medical books. To explore this issue, we have examined how to retrieve information from a prototype hypertext Manual. The remainder of this paper will discuss our work on facilitating information retrieval from a medical hypertext system.

FROM PAPER TO HYPERTEXT

Prior to writing a hypertext therapeutics handbook de novo, we created a prototype handbook from existing chapters of the current Manual. We performed this process in four steps. First, we had to decide how to partition the conventional document into individual hypertext cards. The hierarchical format of the Manual simplified partitioning. Each node in the hierarchy is marked by a clearly defined identifier¹. The text residing between any two identifiers easily fit on a single hypertext card. Second, we had to assign a label to each card. This card label (or equivalently, card title) would be displayed as text in the icons that represented links to the card. We used the first sentence as a label. Third, we had to determine how each new card fit into the hierarchical structure of the rest of the hypertext handbook. When the immediate ancestor of a new card was identified, our parser created a "structural" link between the two cards. After these three steps we completed, we faced the fourth, and most difficult, step -- indexing.

¹ The identifiers are, in order, chapters, subchapters, roman numerals, upper case letters, integers, and lower case letters.

Indexing and Information Retrieval

How people conceptualize hypertext will affect how they design indexes and information retrieval-methods for these systems. If they emphasize the relative autonomy of the hypertext card over the semantic relationships among cards, they will view retrieving information from a hypertext system as similar to retrieving it from a collection of small documents. If they emphasize the semantic links among cards, they will view retrieving information from a hypertext system as similar to traversing a directed graph. The small-document approach emphasizes pattern matching; the graph-traversal approach emphasizes browsing [Xero85, McCr84, Kove85, Mona87]. We will consider each approach in turn.

The Small-Document Approach

Although a wide variety of techniques are available for indexing and retrieving full-text documents, the applicability of these techniques to smaller quanta of text is not well-understood [Fox86]. Full-text document-retrieval methods are quite useful when there are numerous, nonrandomly distributed document descriptors, but their performance is likely to degrade when they are applied to the small number of strings present in the typical hypertext card.

In addition to the theoretical limitations, full-text document-retrieval techniques are costly in terms of storage and computation time, and most available hypertext systems limit their small-document retrieval capabilities to simple pattern-matching capabilities. NoteCards, for example, allows the user to search for strings both in card titles and in card property lists. Intermedia employs a variety of filters to search for keywords associated with individual cards or links.

The Graph-Traversal Approach

The graph-traversal approach is instantiated through active icons on individual cards [Kove86]. A hypertext user traverses the card network either by directly manipulating card icons or by creating graphical browser cards containing the same icons. The decision to display a new card is based on the icon labels in the card currently under examination. The icons reflect either the title of a card or a label representing the semantics of the link. If an icon is activated, procedures associated with that icon display the card denoted by it. The new card may contain yet other link icons that can lead to activation of yet other cards.

FINDING INFORMATION IN A DYNAMIC MEDICAL TEXTBOOK

In some situations, the limited information-retrieval capabilities of current hypertext systems are quite powerful. When the number of cards is small, the domain is restricted, and the semantics of links is well understood, a combination of simple pattern matching and graphical browsing can be quite effective. Pattern-matching queries will, on average, return to the user only a limited number of candidate cards (*Figure 1*). To assess the relevance of these cards, the user must quickly read each of the candidate cards. In restricted domains, card summary information represented by the link icon is likely to make the contents of the card quite apparent to the user. In this case, most cards activated by the user will be relevant to the query and browsing generally will be quite efficient.

There are many situations in which simple pattern matching and graphical browsing are not as effective. First, if the number of cards is large, a pattern-matching query will, on average, return a number of candidate cards too large for sequential examination. Second, if the subject-matter domain is less restricted or if the semantics of the links are not clear, the user will activate many cards that are not relevant to the query.

Third, if the structural and contextual relationships among cards are not maintained, potentially useful cards will not be discovered by a pattern-matching search. Often, cards do not even mention what they are "about," but assume that the reader understands the context because he has read "earlier" cards. (For example, a card on disease manifestations might not even mention the disease name, but instead might assume that the reader wouldn't read the card unless he had just read an ancestor card containing the disease name and other contextual information.) In examining the Manual, we noted that important keyword terms are distributed throughout the card hierarchy. For example, a query about the treatment of digitalis-induced heart-beat irregularities (ventricular arrhythmias) can be accomplished only by traversing the following hierarchy:

- Chapter 6 (Heart Failure)
 - V. (Digitalis)
 - D. (Digitalis Toxicity...)
 - 3. (Treatment)
 - c. (Ventricular Arrhythmias)

Search begins at the chapter level, where the subject matter is declared to be treatment of heart failure. Digitalis is assigned an entire roman numeral section (V.) because of its importance in treatment of this disorder. A part of the digitalis discussion is the section on toxicity (D.). This is the setting in which digitalis-induced ventricular arrhythmias most commonly occur. Beneath this section is a discussion of treatment of digitalis toxicity (3.), and a part of this treatment is the lowercase section relating specifically to ventricular arrhythmias (c.).

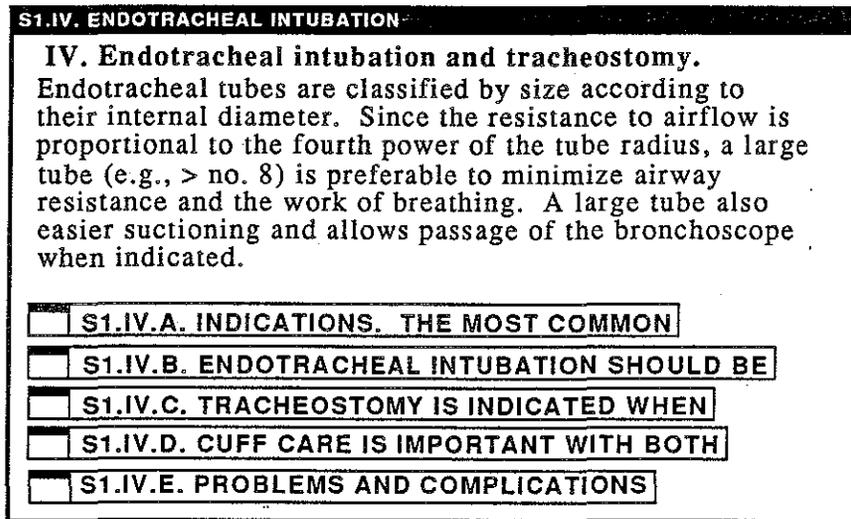


Figure 1: A Representative Hypertext Card. The upper portion of the card displays text. Five link icons are located beneath the text. These link icons show the titles of immediate descendant cards.

We believe that card size, subject-matter domain, link semantics, and card context all must be incorporated into a set of principles for information from hypertext systems. Considering these concepts, we proposed a series of heuristic principles about card utilities, and we implemented these principles in an enhanced version of our hypertext Manual prototype.

PRINCIPLES OF HYPERTEXT QUERY PROCESSING

Hypertext systems should respond to text-string queries by providing the user with one or more best-guess starting points for graphical browsing. We believe the following principles increase the probability of selecting useful starting-point cards:

1. The utility of a card can be approximated by a computed numeric weight consisting of two components. The intrinsic component is the value computed from the number and identity of the query terms contained within the card. The extrinsic component is the value computed from the weight of immediate descendant cards.
2. The intrinsic card weight should be proportional to the number of times each query term occurs in the card and inversely proportional to the number of cards containing each query term.
3. The extrinsic card-weight component should be inversely proportional to the number of immediate descendant cards. A card with many immediate descendant cards but only one query term on one immediate descendant card, should have a lower weight than does a card with fewer immediate descendant cards and only one query term on one immediate descendant card.
4. The optimal starting-point for graphical browsing is the card with the highest weight. The next most "optimal" starting card is the card with the next highest weight that is not a descendent of any card with higher weights. If the next card is an immediate ancestor of any previously identified starting-point card, the ancestor card should assume the descendant's role as a starting-point card.

The application of these principles led us to develop two enhancements that improved the information-retrieval capabilities of our prototype hypertext medical handbook. First, we added an inverted index, a stopword list, and a suffix-stripping algorithm -- methods commonly associated with full-text document-retrieval systems. This allowed us to associate with each query term some notion of its distribution and frequency relative to other query terms. Second, we represented the syntactic hierarchical organization of our document with a specific semantic link type. We used these syntactic relationships to propagate query-generated card weights through the hypertext document. This explicit representation of syntactic relationships among cards allows us to identify parent cards that might not contain any search query terms, but which might nevertheless be ideal starting points because many of their immediate descendant cards contain the query terms.

IMPLEMENTATION OF A HYPERTEXT QUERY PROCESSOR

We used chapters from the current edition of the Manual of Medical Therapeutics as a substrate for our hypertext prototype [Orla86]. We calculated the intrinsic card weights using a modification of a simple, well-known algorithm [Salt83]. This algorithm assigns term weights to cards as a function both of the frequency of occurrence of the term in the entire search space and of the number of cards containing the term. The algorithm assigns a higher weight both to cards containing infrequently used terms and to cards containing several occurrences of a term not found in many other cards. The formula is as follows:

$$\text{WEIGHT}_{ij} = k \cdot \text{FREQ}_{ij} \cdot (\log(n) - \log(\text{DOCFREQ}_j) + 1)$$

where WEIGHT_{ij} is the weight component of card i due to term j , k is a constant, FREQ_{ij} is the number of occurrences of term j in card i , n is the number of cards in the collection, and DOCFREQ_j is the number of cards containing the term j .

The intrinsic card-weight component is combined with the extrinsic component due to weights of immediate descendant cards using the following formula:

$$\text{TOTALWEIGHT}_i = \sum_j \text{WEIGHT}_{ij} + 1/y \sum_d \text{TOTALWEIGHT}_d$$

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

This propagation function is called recursively from the leaf cards to the root card. A graphical display of the search subtree and card weights (Figure 2) serves as a road map for browsing.

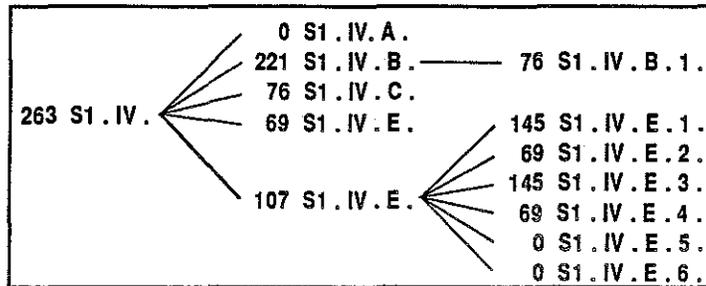


Figure 2: A portion of the search subtree. Card S1.IV. has the highest card weight, because of the presence of the search query terms both within its own substance and within the substance of 4 descendant cards. This graphical display can serve as a road map for browsing.

After card weights are calculated and propagated, the program creates a general search card (Figure 3). Links of type search connect the search card to each card containing one or more instances of any search query terms. The weights of each card are displayed on the search card in front of the link icons.

Tracheostomy

(96) S1.VI.F.2.b. THERAPY SHOULD INCLUDE APPROPRIATE

(96) S1.V.G.2.c. IF MANUAL VENTILATION IS DIFFICULT, CHECK

(96) S1.IV.E.6. EROSION INTO THE INNOMINATE ARTERY WITH

(96) S1.IV.E.3. TRACHEOSTOMY TUBE DISLODGE MENT OR CUFF

(96) S1.IV.D. CUFF CARE IS IMPORTANT WITH BOTH

(96) S1.IV.C. TRACHEOSTOMY IS INDICATED WHEN

(336) S1.IV.E. ENDOTRACHEAL INTUBATION AND TRACHEOSTOMY.

Figure 3: A search card. The search for the term "tracheostomy" has yields seven cards. The last card has a higher weight because of weights propagated from immediate descendant cards. Cards that do not contain the search query terms but that might have high weights because of card-weight propagation are not displayed on the search card.

If the retrieved cards are distributed throughout a large part of the hierarchy, a search summary card is created (Figure 4). This card contains the four highest-weighted cards that can serve as roots for independent graphical browsers.

Summary --> Tracheostomy	
(336)	<input type="checkbox"/> S1.IV.E. ENDOTRACHEAL INTUBATION AND TRACHEOSTOMY.
(189)	<input type="checkbox"/> S1. ACUTE RESPIRATORY FAILURE THE MAJOR FUNCTION
(144)	<input type="checkbox"/> S1.IV.E. PROBLEMS AND COMPLICATIONS
(96)	<input type="checkbox"/> S1.VI.F.2.b. THERAPY SHOULD INCLUDE APPROPRIATE

Figure 4: A search summary card. The cards containing the term "tracheostomy" are distributed throughout the hierarchy. After propagation of weights, the most "reasonable" starting points for browsing are displayed.

This procedure gives the user three options for pursuing an answer to a query. First, the user can browse from a card denoted by an icon on the search summary card. She can browse either by sequentially activating link icons on successive cards or by creating a graphical browser starting at a selected card. Second, the user can scan all the icons from the search card. This affords a thorough but less efficient way of examining every card that contains an instance of any query term. Finally, at any time during a session, the user can reformulate her initial query on the basis of what she has learned from browsing.

HYPertext AS BELIEF NETWORKS

The placement of a link between two cards is an assertion of belief that the value of information contained in one card is conditionally dependent upon another. Propagating weights across syntactic links asserts this conditional dependency. Returning to our heart failure example, the placement of a structural link between "3. Treatment" and "c. Ventricular Arrhythmias" states if there exists a structural link between "3." and "c.", the utility of card "3." will in part depend upon the utility of card "c.". In this sense, hypertext systems can be conceptualized as belief networks [Pear86a], where links serve as constraints to limit the amount of search and reasoning we must perform to obtain information. This conceptualization might provide a valuable key to the integration of hypertext systems and artificial intelligence.

FURTHER ENHANCEMENTS TO SEARCH - THE USER INTERFACE

An extensive body of research suggests that computers can surpass conventional texts in many educational and information-retrieval tasks. Perhaps the richest tradition evolves from the SmallTalk experiments in the 1970s [Gold83]. This work led to Kay's implementation of his Dynabook concept [Kay75], and later to Weyer's formal evaluation of this paradigm as a teaching tool [Weyer82]. In a subsequent publication, Weyer and Borning envision an interaction between an electronic encyclopedia and a reader as a "conversation with a guide or tutor who accompanies us during our learning adventure" [Weyer85a] -- the reader embarks with a query, gains knowledge along the tour, and returns with an answer.

A number of commercial concerns are attempting to create devices that will function as "electronic books" [Micr86, Owl86]. But reading is just one of many ways that people interact with their books; They write in them, they fold pages, they remove pages, and they insert bookmarks. We are constructing a medical hypermedia system that changes our way of reading, but do we address all of the ways in which a book is really used? What aids do readers employ while reading a medical book? We suspect that an analysis of reader behavior would show that some of the following features would prove useful:

- Medical readers like to use highlighters to emphasize important concepts and keywords. Every medical school bookstore keeps an ample supply of yellow, orange, and green marking pens. Some readers highlight only rare terms, some highlight almost every sentence they read, and some highlight only those words already set in boldface or italics! Some highlight in multiple colors, suggesting they are following some implicit semantic rules. The common characteristic can be generalized: People like to superimpose their own values on print.

- Readers like to annotate their reading. These annotations may be the explication of an implicit statement (e.g., a definition of "class IV heart failure"), a supplement to published information (e.g., a trade name where only a generic is listed), or a correction that makes the text more appropriate to the user's setting (e.g., order a "Chem Panel 9" when the text says "check the serum electrolytes"). Each annotation is meant to convey additional information about the passage to which it refers.
- Readers frequently turn pages back and forth as they read. Medical hypermedia will require several types of page turners. First, these systems will need a text-to-text page turner to explore references made to other portions of a document. Second, they will need a text-to-figure/table page turner to examine figures and tables while reading text. Third, they will require a text-to-dictionary/glossary page turner to examine formal definitions of terms. Fourth, readers often must refer to other articles while reading. For this task readers need a text-to-text page turner. Finally, they will require a text-to-reference page turner to examine citation headings referenced in the text.
- At times, the path from query to answer takes the reader through many references full of relevant points. Readers need some mechanism to mark the trail of an enjoyable path so that they can return to it in the future. This requires that the system be able to save a trace of the query and reading sequence. Brown University's Intermedia refers to these traces as webs [Meyr86, Yank85].
- Medical life is characterized by interruption. Readers need a "smart bookmark" that keeps their place within the text when their reading is interrupted. This bookmark should help to simplify the reader's task when he continues reading. If he has paused for only a few minutes, the bookmark should just mark the portion of the text where reading stopped. But if the reader has paused for several hours or days, the bookmark should help to restore the context by providing a summary of the previous reading session.
- Medical readers often photocopy citations, pertinent phrases, or entire pages. They need a clipboard for recording such notes. This clipboard also should record the source or context of the copied material so that they rarely have to ask "Now where did I read that?"
- The pursuit of an medical problem creates a continuously growing list of new ideas and knowledge sources. Readers engaged in this intellectual pursuit need an agenda-keeper to retain lists of future readings and tasks. This device could request the appropriate information automatically, so that information will be available when desired.

All these tools require some knowledge (or assumptions) about the content of the hypertext document and the reader's goals, knowledge, and preferences. At one extreme lies the common user profile used by operating systems, text editors, and electronic mail systems. At the other extreme lies the metaphor of dynamic book as conversation: The goal of the reader is to obtain information; the goal of the program (the speaker) is to understand what the user wants. Implementing a program that understands what the reader wants remains one of the most challenging problems facing artificial intelligence.

CONCLUSIONS

Our work suggests that both the small-document and the graph-traversal approach can provide useful perspectives toward information retrieval from hypertext. Our prototype demonstrates the power of combining these two paradigms, and suggests the need for further work in two areas. First, extending the notion of card weight propagation using belief network techniques might provide one way to incorporate artificial intelligence techniques into hypertext systems. Second, although effective information retrieval mechanisms will be essential for robust hypertext systems, the ability to personalize these systems also will greatly enhance their utility within the medical community.

Acknowledgements

This work has been supported by grant LM-07033 from the National Library of Medicine and a grant from the New Medical Foundation. Computer facilities were provided by the SUMEX-AIM resource under NIH grant RR-00785. I thank Steve Weyer, Gio Wiederhold, Edward H. Shortliffe, Christopher D. Lane, and Lyn Dupre for their advice and support.

REFERENCES

- [Aamc86] Association of American Medical Colleges, "Medical Information in the Information Age". Proceedings of the Symposium on Medical Informatics, Association of American Medical Colleges, Washington DC, 1986.
- [Bush45] V. Bush, "As We May Think," *The Atlantic Monthly* Vol. 176, July, 1945, p. 101.
- [Cove85] D. Covell, G. Uman, and P. Manning. "Information Needs in Office Practice: Are They Being Met?," *Annals of Internal Medicine* Vol. 103, October, 1985, pp. 596-599.
- [Fox86] E. Fox. "Information Retrieval: Research into New Capabilities," in CD-ROM, *The New Papyrus*. Microsoft Press, Bellevue, Washington, 1986, pp. 143-174, .
- [Gold83] Adele Goldberg and David Robson, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley Publishing Company, Reading, Massachusetts 1983.
- [Kay75] Alan Kay, "Personal Dynamic Media," Technical Report, Xerox Palo Alto Research Center, Palo Alto, 1975.
- [Kove86] L. Koved B. Shneiderman, "Embedded Menus: Selecting Items in Context," *Communications of the ACM*, Vol. 29, No. 4, April, 1986, pp. 312-318.
- [McCr84] D. McCracken and R. Akscyn, "Experience with the ZOG Human-Computer Interface System," Technical Report CS-34-113, Computer Science Department, Carnegie-Mellon University, Pittsburgh.
- [McCr84] D. McCracken and R. Akscyn, "The ZOG Approach to Database Management," Technical Report CS-34-113, Computer Science Department, Carnegie-Mellon University, Pittsburgh.
- [Meyr86] Norman Meyrowitz, "Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework," in *Proceedings OOPSLA 86*. 1986.
- [Micr86] Microsoft Press, CD-ROM, *The New Papyrus*. Microsoft Corporation, Bellevue, Washington, 1986.

- [Mona87] I. Monarch and J. Carbonell, "CoalSORT: A Knowledge-Based Interface," IEEE Expert Vol. 2, No. 1, Jan. 1987, pp. 39-53.
- [Orla86] M. Orland and R. Saltman eds., Manual of Medical Therapeutics, 25th Edition. Little, Brown, and Company, Boston, 1986.
- [Owl86] Owl Technology, Guide User's Manual, Owl Technology, 1986.
- [Pear86] J. Pearl, "Fusion, Propagation, and Structuring in Belief Networks," Artificial Intelligence Vol. 29 No. 3, Sept. 1986, pp. 241-288.
- [Salt83] Gerard Salton and Michael J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill Book Company, New York, 1983, p. 63.
- [Warr81] K. Warren ed., Coping With the Biomedical Literature. Praeger, New York 1981.
- [Weyer85] S. Weyer and A. Borning, "A Prototype Electronic Encyclopedia," ACM Trans. Office Information Vol. 3, No. 1, Jan, 1985, pp. 63-88.
- [Weyer82] S. Weyer, "Searching for Information in a Dynamic Book," PhD thesis, Department of Education, Stanford University, February, 1982.
- [Wino86] Terry Winograd, "A Language/Action Perspective on the Design of Cooperative Work," in Conference on Computer-Supported Cooperative Work, MCC Software Technology Program, December 3-6, 1986, pp. 203-220.
- [Xero85] Xerox Special Information Systems. Notecards Release 1.2i Reference Manual. Xerox Corp., Pasadena, California 1985.
- [Yank85] N. Yankelovich, N. Meyrowitz, and A. van Dam, "Reading and Writing the Electronic Book," IEEE Computer, Vol. 18, No. 10, October, 1985, pp. 15-30.

Hypertext and Pluralism: From Lineal to Non-linear Thinking¹

William O. Beeman, Kenneth T. Anderson, Gail Bader, James Larkin, Anne P. McClard, Patrick McQuillan, and Mark Shields

Office of Program Analysis
Institute for Research in Information and Scholarship
Brown University
P.O. Box 1946
Providence, Rhode Island 02912

ABSTRACT

One goal of American and Northern European higher education is to promote acquisition of a pluralistic cognitive style, which has as an important property—non-lineality. This paper investigates the effects of using of an advanced hypertext/hypermedia system, Intermedia, to develop instructional materials for two university courses in English and Biology intended to promote acquisition of non-linear thinking. Use of Intermedia is shown to produce significant learning effects, which are somewhat more pronounced for persons involved in developing materials than for students using the system.

INTRODUCTION

A central problem in the study human cognition is whether human cognitive styles—basic modes of apprehension and codification of reality—are universal or variable across cultures and life stages. A related problem is the question of whether human cognitive styles can be changed through education and training. This paper addresses these questions, and poses a more specific one: Can complex human cognitive styles be changed more effectively using computer-aided learning processes, specifically those using hypertext applications, than through traditional post-secondary school instructional methods?

Many American educators believe that student learning in post-secondary education is tied to a change in cognitive processes. Students are thought not only to acquire knowledge, but also to learn new ways of thinking. Essentially they are thought to acquire a new cognitive style through the educational process. As we will show below, most American and many Northern European educators view a cognitive style referred to variously as pluralistic, relativistic, or critical thinking as the most desirable goal of the educational process. We maintain that all of these cognitive styles, though they may differ in emphasis, share an important feature: an emphasis on *movement toward non-linear thinking*. Throughout this discussion we will refer to this cognitive style as pluralistic, non-linear thinking, though some of the sources which we quote and to which we refer use some of the other terms cited above.

In the past decade it has also been widely accepted that student learning can be improved through the use of computer-based instruction (CBI). While a vast literature already exists on CBI and its effects (Kulik, Kulik and Cohen 1980; Kulik, Kulik and Bangert-Drowns 1985), the

jury is still out on exactly *what* accounts for improved learning through CBI (Clark 1983, 1985a, 1985b). Moreover, until recently most instructional software was designed to promote essentially lineal thinking through such techniques as programmed instruction and drill-and-practice. The potential value of CBI for promoting non-lineal thinking has been unexamined. At Brown University we are now completing an assessment of a three year project, the Educational Software Project (ESP), which directly addresses the development of pluralistic thinking through the use of sophisticated educational software. The discussion which follows derives from this assessment

Cognitive Styles and Western Educational Processes

There is considerable research suggesting that humans everywhere may not think alike, nor may they think in the same way at all stages of their lives. Cognitive styles can be contrasted in many ways. Anthropologists Edward Sapir and Benjamin Whorf conducted research on language and cognition leading to the well-known "Sapir-Whorf Hypothesis" (Sapir 1924, 1949; Whorf 1956). This suggested to many researchers that there may be a linkage between the language one speaks and one's patterns of thought. ² An extreme formulation of the hypothesis would suggest that there are as many styles of thought as there are varieties of language.

Cognitive psychologists have also developed typologies of thinking styles based on environmental conditioning and ontogenetic development. Bernstein's "elaborated vs. restrictive codes,"³ Piaget's stages of early childhood cognitive development ⁴ and William Perry's scale of development in college students from "dualistic thinking" to "contextual relativistic reasoning" to "commitment,"⁵ which we will consider at greater length below, are examples of these.

Educational systems in the United States, Canada, Great Britain and Australia (seen in part in other Northern European nations) seem unique in their tacit acceptance of the notion that cognitive styles can be changed through directed instruction. In many countries, the basic mental functions of students are assumed to be in place and intact early in a child's school career. The basic function of education is then to exercise the brain through rote memorization and standard drill exercises with less attention to integration of that information. The French and Japanese education systems are notable for this.

In the United States higher education has traditionally been seen as contributing to an ongoing human ontogenetic process. Through education the student is seen not only to acquire information, but also to develop new capacities as a human being. Styles of thinking are seen to be different at different stages of education, and much energy is devoted to seeing that students learn "how to think." It is also felt that styles of thinking are coupled with ethical development, and that ethical training cannot be divorced from learning itself.⁶

The cognitive style most widely fostered in American and Northern European post-secondary education today is the promotion of pluralistic, integrated thinking, whereby students are encouraged to see the phenomena of the world in interrelated relativistic terms rather than as isolated bits of information.⁷ William Perry, in a comprehensive study of Harvard final examinations from 1900 to 1960, documents a rise in the percentage of examinations requiring "considerations in more than one frame of reference, that is, relativism (Perry 1968: 4)" from a high of 20% in 1900 to 70% in 1960. He has referred to this as the "intellectual revolution of this century." (Perry 1976:30) Most research on the development of this cognitive style has taken place at single insti-

tutions of higher education. Perry's work (1970) took place exclusively at Harvard, and Ray Heath's similar work (1964) at Princeton. However, research in Lancaster, England over the past decade focusing on university lecturers' aims and objectives showed a widespread, clear pattern of expectations at many institutions.

While knowledge and technical skills were expected, students had to be able to use these effectively—to combine and interrelate ideas . . . The unifying theme both in the interviews and in the general literature on the aims of university education is that of 'critical thinking' or as Ashby has described it—'post-conventional thinking' (Entwhistle and Ramsden 1983: 7-8).⁸

As stated in the introduction, we gloss this kind of thinking as pluralistic cognitive style.

The nature of non-linear thinking

Pluralistic cognitive style in its extreme ideal form is non-linear in nature. The ideal scholar is one whose insights and perceptions are automatic, correct and spontaneous, proceeding from a total integration of knowledge and the principles which govern its meaning and use across a broad spectrum of contexts. Total pluralistic integration of knowledge is an asymptotic ideal in complex societies, approached only by Zen masters and mystics. It is seen in more limited ways in the expertise and facility of the most revered experts in any discipline, who are able to encounter new phenomena and interpret them correctly, seemingly without thinking.

The burning question for educators is how to move novice learners to the point where they apprehend and develop this style of thinking (Kohlberg 1972). Every instructor knows students who work very hard, read everything assigned, and perform well on tests, but whose understanding of subject material is somehow mechanical—they just don't "get it," the "it" being the cognitive style instructors feel appropriate for the material.

The paradox in post-secondary education is that the ideal of non-linear knowledge is approached through lineal communication, presentation and instruction. Lineal knowledge implies lines, fixed structures and definitive processes. Anthropologist Dorothy Lee describes academic lineal structures in this manner:

In our academic work, we are constantly acting in terms of an implied line. When we speak of *applying* an attribute, for example, we visualize the process as lineal, coming from the outside. . . . When I organize my data, I *draw* conclusions *from* them. I *trace* a relationship between my facts. I describe a pattern as a *web* of relationships. Look at a lecturer who makes use of gestures; he is constantly making lineal connections in the air. And a teacher with chalk in hand will be drawing lines on the board whether he be a psychologist, a historian, or a paleontologist. . . . The line is found or presupposed in most of our scientific work. It is present in the *induction* and the *deduction* of science and logic. It is present in the philosopher's phrasing of means and ends as lineally connected. Our statistical facts are presented lineally as a *graph* or reduced to a normal *curve*. And all of us are lost without our *diagrams*. We *trace* a historical development; we *follow the course* of history and evolution *down* to the present and *up from* the ape; and it is interesting to note, in passing, that whereas both evolution and history are lineal, the first goes up the blackboard, the second goes down (Lee 1959:110).⁹

Lee's description of academic presentation of information is accurate, but she makes the assumption that Westerners are incapable of thinking in any but this exclusively lineal manner. She contrasts this way of thinking with the non-linear thought of the Trobriand Islanders who she claims not only *think* non-lineally, but also have no grammatical expressions which imply linkage, such as the word "and."

Her characterization of thinking in the West is, we believe, unwarranted. Westerners are not only capable of non-linear thinking, they strive to achieve it. But representing its nature to others is often difficult, if not impossible. Descriptions by nature require language, presented in a time-stream, and thus scholars resort to the diagrams and inductive-deductive models Lee describes, but other experts reading these materials understand them for what they are: mere skeletal suggestions of the *real* thought lying below.¹⁰ It is for this reason that "expert" systems designed for the computer cannot represent true expertise. As Dreyfus and Dreyfus (1986a, 1986b) point out, true expertise is indeed non-linear—the expert simply *knows* without consciously transversing a complicated decision matrix.¹¹ The closest we may get to public representation of this state is in non-discursive and semi-discursive forms such as music, performance and poetic imagery.

Nevertheless college instructors do not shirk from trying to represent this view to students despite its difficulty. Viewing non-linear knowledge from a lineal perspective is somewhat like viewing three-dimensional objects from a two-dimensional perspective. The non-linear world seems to consist of conflicting ideas, multiple explanations and contradictory interpretations of phenomena. For the pluralistic knowledge worker such conflicts do not exist; all items of information have a natural place within a patterned cognitive universe.¹²

Teaching non-linear knowledge

In a university setting the pursuit of non-linear knowledge is a life-long occupation. It is an occupation chosen by those who are able to live with such an open-ended work activity, with the full knowledge that one's work will never be complete, and that it will always be superseded by subsequent generations of workers. Nevertheless, even partial integration of a large body of knowledge provides enormous personal satisfaction, and is extremely practical as a basis for conducting one's life.

An instructor in a college classroom cannot hope to imbue students with this vision from the very start. He or she can, however, attempt to start students on the path to viewing knowledge in this way. The first step is often to disabuse students of the notion that there exist single truths in the world. The second is to introduce them to the notion of pluralistic thinking and multi-causal explanation. The third is to bring them to a level of sophistication which enables them to examine and question the parameters by which knowledge itself is defined,¹³ thereby allowing them to make personal judgments and choices about correctness and validity.

Many logistical difficulties face the instructor in this pursuit. First, students typically lack the knowledge base that serves as the critical foundation for pluralistic thinking. Second, students and instructors meet at disjointed, infrequent intervals. Third, the material in a course of instruction must be laid out in a lineal fashion, piecemeal over the course of several months. Fourth, the material is disjunct: lectures, readings, exercises and examinations all sprinkled throughout the time of the course. Finally, students themselves are left to interrelate this disjunct material the best they can. The proof of their success usually lies in their performance on examinations, which often become the most meaningful integrative exercise engaged in by students. Every instructor feels mixed happiness and sadness when, as often happens, he or she hears from students: "I didn't really know what this course was about until I wrote the final exam."

Conscientious instructors do the best they can to get beyond the lineal properties of the material they are presenting to students, but in direct instruction non-linear knowledge can only be

conveyed by implication. Just as in mathematics two points determine a line and two lines a plane, so can instructors suggest the pluralistic shape of knowledge in a given course of study through systematic presentation of fragmentary pieces. Nevertheless, mere presentation of material does not guarantee that students will actually make the transition from lineal to non-lineal thinking. Rhetorical strategies must be utilized to guide students to the correct use of information.

Intermedia—Hypertext tool for promoting non-lineal thought

The Institute for Research in Information and Scholarship (IRIS) at Brown University undertook a three year research project (under support from the Annenberg/CPB project) aimed at creating software designed to address two difficult problems in instruction, the *connectivity* of materials and the *visualization* of concepts and ideas. The software developed for the project, called Intermedia, consisted of a number of applications or "tools" such as a text editor, graphics editor, timeline editor, a three dimensional viewer, etc., within a hypermedia framework allowing the materials created by the applications to be linked, annotated and navigated. Utilizing a hypertext, hypermedia model, it was hoped that new capabilities for visualization and connectivity embodied in Intermedia would enable students to synthesize course materials and facilitate instruction.¹⁴

Connectivity addresses the question of the relations between material. One of the key aims of Intermedia was to provide the ability to link different materials creating semantically meaningful relationships between different information and ideas. Using this software environment, "authors" (who may be students or instructors depending on the guidelines determined by the instructor) could link and annotate materials to create "semantic webs" or trails of meaning. In the course of the Educational Software Project, connectivity early became defined as the need and ability to show that any single concept, problem or idea must be understood from a variety of perspectives. The materials created attempted to demonstrate this interconnected juxtaposition of ideas. Connectivity had another meaning within the project, in that the workstations on which Intermedia ran were connected via ethernet, thus allowing all users to share files.

Visualization referred to the use of "dynamic, interactive graphics" to build and interact with visual representations of concepts and processes. The representation of processes and concepts are often laborious (drawings on chalkboards) or cumbersome and lack the flexibility to be manipulated by instructors and students. Interactive graphics however, allow instructors to represent dynamic concepts to their students and to manipulate these concepts to illustrate process.

The Educational Software Project developed Intermedia for, and examined its use in two Brown University courses, English 32, an English literature survey course, and Biology and Medicine (hereafter Biomed) 106, a plant cell biology course. Each of the two instructors were asked to use Intermedia to develop educational materials to meet their specific instructional needs. The materials to be created, the way in which they would be utilized in the course and the form of presentation were decided by the instructors and were not pre-determined by project criteria.¹⁵

Each of the courses was closely observed by a team of social scientists, once as the instructor taught it prior to the introduction of the Intermedia materials and again when the course was taught using the materials. During this time instructors and students were interviewed several times, and a select group of students was asked to keep time and task diaries of their activities during the semesters in which the courses were taught. A computer laboratory was set up for use by the project classes, and student and instructor use of this facility was also observed.¹⁶

Pluralistic thinking in an English literature course

For the students in his English literature course, the instructor, George Landow, has one main goal—to develop advanced scholarly skills in the field of English literature in a self-directed fashion. To attain this level of intellectual sophistication, Professor Landow believes that students need to progress through two interdependent stages of development. They first have to acquire extensive familiarity with the literature through comprehensive reading and study. The second stage requires critically applying this knowledge to issues raised in the literature. Beyond mere literal understandings of the works, students should be able to draw on a range of contextual sources (e.g., authors' biographies, political, religious, economic, and social developments of the time, etc.) to support or refute literary interpretations, as well as to understand why one analysis constitutes a valid assessment where another fails.

Professor Landow refers to the ability to interrelate contextual influences and literary works as “multi-causal” reasoning. His course syllabus contains an elaborate explanation of this process:

If this course has one central idea, it is that no literary phenomenon—no work, part of a work, or idea about one—can ever be explained by a single fact. All literature. . . is multi-determined, by which we mean that multiple causes impinge upon each fact. Dickens may write a particularly great novel in a certain way because, 1) he needed money, 2) he worked out his own psychological problems in writing it, 3) he confronted and challenged past and contemporary novels and novelists, 4) he wished to convince his readers to think about the world in a certain way—and so on. All are explanations, and they don't conflict with one another.

This approach, emphasizing the integration of a range of factors to create pluralistic understandings, underscores his interest in having students “learn how to synthesize information rather than parrot back concepts,” a general skill, he argues, that “one can use in any course you care to take at Brown or elsewhere.”

In his teaching, Professor Landow found it difficult to bring students to this point of intellectual sophistication due, most often, to one of two obstacles: “Either students don't have enough background or they don't know what to do with their knowledge if they have read a lot.” The traditional response to this dilemma, he felt, was for instructors to “spoonfeed” students with some background knowledge and literary analysis—what he called, “the consumer theory of education,” where instructors produce and students consume. Rather than spoonfeeding students, Professor Landow sought to nurture a “community of discourse” in which students could refine their analytic skills by devising their own literary interpretations of assigned readings and criticizing those of their classmates. In this process, class discussion in a seminar format was extremely important. Professor Landow, in fact, used quality of class discussion as an important index of the success of the instructional process.

Pluralistic Thinking in a Biology Course

Peter Heywood teaches Biomed 106, a plant cell biology course, one of the courses in which Intermedia was implemented. His primary goal for the students in Biology 106 has always been to teach them to think like biologists. This requires that the students acquire a broad base of knowledge from which they should be able to draw upon to integrate materials from many different

areas of cell biology. Also, they should be able to apply this knowledge to new information and experimental problems creatively.

Thinking like a cell biologist means thinking experimentally. This involves the application of pluralistic biological knowledge to experimental problems. Learning through experimentation is one of the ways in which students have of connecting information, integrating and applying the knowledge which they acquire from other contexts (i.e., lectures, textbooks, journal articles, discussion, etc.). Because Biomed 106 is taught without a laboratory, as are over fifty percent of the biology courses at Brown, the students have more difficulty integrating the knowledge which they acquire from the class through their own experience. Peter Heywood's goal, then, is to give the students the opportunity to learn experimentally without actually participating in a "hands on" experiment.

In the past, Peter Heywood has had difficulty in conveying the connectivity of all of the course materials for Biomed 106. Aside from not having access to laboratory facilities, difficulties in teaching students to think of biology pluralistically arise because the course has been taught in a traditional lineal fashion. By traditional lineal fashion we mean that the course has been geared around lectures, textbook readings, and journal article readings, all of which have been presented in a sequential manner.

Textbooks segment instructional material into topics. The topics are usually dealt with in full in separate chapters. The material is not usually integrated between chapters. A student will sometimes come across a footnote which tries to integrate material in one chapter with material in another, but in general the student is left to his or her own connective abilities. The author(s) often hope that the reader will piece the material together after they have finished reading the book.

Lectures also suffer from the segmental topic phenomenon. The instructor often has a particular topic area to cover in a limited amount of time. Usually this subject is marked out in a syllabus so that the student also knows what to expect in lecture for a particular day. However, Peter Heywood tries to integrate material from various parts of the course into his lectures. This approach is not wholly satisfactory. A partial explanation is that because the students are aware of what is to be covered, they tend to focus on that specific material. Also, words in a lecture seem ephemeral to the student who is busily taking notes. Often the best a student can do is to jot down the main ideas, and in doing so he or she often misses some of the more subtle and important connections that the professor makes about the course material.

Nevertheless, Professor Heywood relentlessly attempts to guide the students to learn to think in a nonlinear experimental fashion during the course of his lectures by posing questions which are intended to spark an integrative train of thought in the students' minds.

Using journal articles also has its pitfalls in a lineally taught course. Ideally, for the biology student, journal articles should serve as "springboards" for thought. Students should be able to take the central ideas of the articles they read and incorporate them into a larger body of working knowledge from which they can generate new and testable hypotheses. However, students often only attend to issues relating directly to the particular topic being focused on for that particular week, or for the upcoming exam.

In the past, Professor Heywood has tried to resolve some of the difficulties of demonstrating the connectivity of material by requiring students to write "a professional paper". Each year that

he has taught Biomed 106 he has assigned a twenty to twenty-five page research paper to his class. This is the usual publishable length biology paper in the profession. Aside from giving the students practice in writing professionally, the paper gives the students the opportunity to write on a topic that is of immediate interest to them, integrating and synthesizing some of the material that they learned in the class with information they researched in the library. Like professional biologists, they are expected to look critically at their topic and to use up-to-date material. It is described in the course handbook as follows:

The term paper is an opportunity to become an expert in a subject that interests you, and so it is important that the topic should be your choice. Each year approximately half of the class chooses to write on a topic which is different from those suggested and/or write on organisms other than plants (medical issues are a frequent alternative).

The term paper will require reading and critical discussion of original publications (such as scientific papers and review articles). The paper should be written on the cellular level and should be detailed and specific. Please reference your sources in the text and include a bibliography at the end of the paper. . .

The paper also tests the students abilities to think experimentally, because part of thinking critically about their topic is to suggest alternative ways of seeing their particular problem, and suggesting new experimental ways of approaching it. Professor Heywood said,

. . . when the students write their term papers they have to emphasize an experimental approach. They not only have to review the literature in a critical way, but say what they themselves would do in terms of designing an experiment.

Tests also serve the purpose of helping the students to synthesize and interrelate the vast quantities of information. Unfortunately, students too frequently learn the material for exams only well enough to pass, and do not retain information any longer than is necessary. Unlike the paper where the students learn through integrative thinking, exams may only test the students' short term recall—especially the short answer exam questions. The essay exam, on the other hand, comes closer to testing the students' abilities to make connections and to think creatively about a problem related to the course. Professor Heywood uses exams which are a combination of short answer and essay because he feels it is important to test two levels: first, whether the student is able to access basic materials which are fundamental for the course and second, whether they are able to synthesize information and formulate an argument based upon the material. Professor Heywood has always preferred the essay questions on exams to the short answer questions for judging whether the student is truly understanding the course material.

Ultimately, what Peter Heywood wants, as most instructors do, is to have his students really KNOW the material. He doesn't want them to simply memorize volumes. Rather, he wants them to take information and ideas in such away that they generate new information and ideas. Basic knowledge is necessary for this to occur, along with a certain amount of memorization, but he feels that the material is not truly learned until it can be easily translated from one context to another by the student.

How Intermedia worked in promoting pluralistic thinking

For George Landow, Intermedia offered a method for mediating between student preparation and the classroom discussion process he considered so important for developing and demonstrating

pluralistic thinking. Primarily, Intermedia would serve as an information source, an "electronic encyclopedia" that provided social, historical, and other contextual information, previously available only through "spoonfeeding," which students would later interweave with their understanding of the texts to foster multi-causal analysis in class discussion and on exams and assignments.

In addition to functioning as an electronic encyclopedia, Intermedia offered a model for understanding the interrelatedness of information central to Professor Landow's pedagogical aims. As he explained, "Intermedia is going to allow students to develop habits of linking things . . . of making connections, of making comparisons, and drawing discriminations and formulating problems. Most important for us . . . formulating multiple explanations of complex phenomena." For instance, browsing through the Intermedia materials roughly paralleled the critical thinking characteristic of multi-causal reasoning. By following a path of their choice through a web of interconnected documents, students begin to see how writers, works, and their literary periods connected, to weigh the value of different sources, and to realize that they could personally construct valid literary interpretations. Ideally, rehearsing these skills during Intermedia sessions would prepare students for carrying through similar analytic operations during class discussion and while writing papers and exams.

The Intermedia materials themselves contained a second model for understanding multi-causal analysis, the "concept map." Designed by Professor Landow and his graduate assistants to represent interrelationships graphically, these heuristic devices presented authors and other relevant topics encircled by such contextual and literary influences as "religion," "technology and science," "literary relations," "biography," and "works." Differing from a standard table of contents, the topics were arranged to suggest more than lineal relations. As Professor Landow explained

You see things arranged around a central concept or entity and that is establishing in you the habit of thinking of entities surrounded by interconnecting ideas. . . [leading one] to think in complicated and sophisticated ways. . . . [This] is a perfect model for education. It can be used in anything, whether you're reading the newspaper, working mathematically, or anything else. It is a way of teaching you that education is a thinking, active procedure.

Given this tool for interrelating extensive background materials, Professor Landow was able to remove himself from dominating class discussion, especially limiting his need to lecture. Observation in Professor Landow's class showed that he adopted more of a "coaching" role and relied far less on didactic presentations. Chart 1 (See Appendix) shows a dramatic difference in his classroom role before and after Intermedia use. Using Intermedia Professor Landow prompted the class with many fewer questions, and student questions and observations increased. This, had the additional effect of allowing more time for discussion. Students, now primed with contextual understandings, could determine the course class would follow rather than following the instructor.

Professor Landow also felt that quality discussion had improved. Noting factors which may have contributed to this development, he explained:

From the very first, discussions have been better than they have ever been. . . . I have heard occasionally people actually quote Intermedia or introduce information without saying this, but that is the only place they could have gotten it. . . . [also] it seemed to me a lot more people had very specific correct things to say about individual poems. Now when someone would say something, [others] would respond as though they had read . . . and had thought about it. . . . they felt more at home with the materials. . . . and when I chime in with something, they don't let me control the situation. . . . they keep talking. The ability . . . for

me to interrupt and have them immediately take over discussion again is very different from last year. Last year, I had to be extremely careful or they would just stop talking. So those are a whole series of improvements . . . they are willing to ask a questions . . . they will try to answer mine and will go on and ask their own.

As a consequence of using Intermedia Professor Landow felt that formal, graded student performance also improved in a variety of ways. For instance, he felt that student writing assignments "proved markedly superior to those in previous years." The grader for the mid-term exam, a woman who had taught at Brown for six years and who did not know the students, described the exam as the "most difficult exam, let alone mid-term" that she had seen. As for student performance, the "high quality of the essays" which demanded that students "not just spit back facts," especially impressed her. Since students had to synthesize and analyze literature, much of which came from the eighteenth century, she was "really amazed at how well the students understood the period." In her experience, students generally found it hard to realize the complex interrelationship between 18th century literature and the social environment in which it was written—yet, the English 32 students were, in her judgment, able to grasp this aspect of the literature well. She felt compelled to assign five students grades in excess of the theoretical maximum for the examination. The students receiving these high grades were among the highest users of Intermedia.

Many students in English 32 agreed with Professor Landow's assessment of their performance. Commenting on her development of multi-causal analysis, one student said:

. . . you can get an idea of the author, put him in his historical background, the time period, and other author he relates to. . . . So, you get an idea of the person within the whole course. . . a way to place authors in some kind of context . . . something to refer to when . . . trying to remember things about them.

Another student outlined the ways Intermedia allowed him to make integrated, pluralistic analyses: "I can look at 'science and technology' (a file on Intermedia), then I can look at 'Victorianism,' and then I can look a biography and put them all together myself."

In English 32 the corpus was designed to expand students' contextual understanding of English literature and so to provide a basis for further investigation. One student explained how using Intermedia in this fashion affected discussion:

I know more about the materials than I am going to garner from simply reading [the text] and therefore I can go into discussion with a better background, more knowledgeable, and I can say more intelligent things.

By creating a "common denominator" among students, another student felt Intermedia boosted the quality and complexity of class discussion:

Intermedia allows me to get past the basic need for rudimentary information—biographies, history—outside of class. In class then, one has a 'common denominator' among the students and you can go on past it.... [O]nce you are in the classroom...you are sensing how [other people] respond to the literature...and you can integrate all of this information that would otherwise have to be given in a straight, boring lecture.... [Y]ou can say, 'OK, Browning was such and such. I think this influenced his poetry in this area and this paragraph specifically because....' You are getting at the heart of the [author], of the literature, and how it applies to what you are doing....

Students also saw how using Intermedia in this fashion affected the instructor's role in class:

Professor Landow doesn't have to sit and talk forever and ever to give us all this information about the background;...we can get a lot of that ourselves from Intermedia.... [S]o that helps our discussion...get into more specific things and [Lan-

dow does] not have to give us this general stuff so we can all have better discussions.

One component implicit in multi-causal analysis is the acceptance of varying perspectives—that, rather than having one “right” answer, multiple pieces fit together to form a coherent whole. One student provided a telling instance of her appreciation for this approach when she copied some Intermedia notes on Romanticism for a friend in a different, non-Intermedia section of English 32. She gave her these notes because “it is just another viewpoint, and the more viewpoints you have, the better anything is. . . . And she just had the viewpoint of one professor.” Summing up how Intermedia and Professor Landow’s instruction fit together to create an integrated understanding of course materials, a student remarked, “[This approach] ties everything together and makes a survey course cohesive. It means something, rather than skimming across the top of things that you can’t get into.”

The sense of developing an pluralistic, non-linear appreciation for English 32 came across when students tried to separate out and assess various course components. Reflecting on her preparation for the mid-term, one student said, “I couldn’t really tell what I had gotten from lecture or [class discussion] section or the introductions in the book or from Intermedia,” even though these were the materials she relied on for completing the exam. Another student, when asked what Intermedia had contributed to the course this semester, remarked, “I do think it adds to [the course]. . . it is just hard to define how much.”

The final bit of evidence suggesting that students broke from a lineal understanding of English literature is the students’ sense that process had value in other courses—a transfer of learning. One student compared English 32 to other literature classes he had taken:

I can place things [in history] whereas in other classes, I would read something but I didn’t have an understanding of the influences surrounding it . . . or [know] how to relate it to other . . . pieces of literature. . . . I feel like I am doing that now and I will be doing it in the future too—trying to relate things.

A second student reflected on what he had taken from the class saying, “Learning to learn. The power of learning . . . it doesn’t matter in what area because you can use the process of learning . . . in any area.”

Professor Landow sought to bring students to a level of intellectual sophistication at which they would personally devise literary interpretations based upon non-linear, multi-causal synthesis of text with extraliterary, contextual information. Viewing student performance in light of these goals, their appreciation for multi-causal reasoning, their sense that knowledge sources had become so entwined as to blur their origins, and their realization of this skill as applicable to “learning” in a broad sense of the word, all suggest that many students in English 32 did, at least, begin to achieve true non-linear, pluralistic thinking.

Peter Heywood feels that Intermedia was a success in a number of ways. His primary goal for his class is to teach his students how to think like cell biologists. This requires that they learn to think less lineally; that they come to understand that there are many explanations for the same phenomenon, and many different ways of arriving at those explanations. Intermedia has provided him with an effective way to supplement his classroom teaching to accomplish this particular goal.

Additionally, Intermedia has helped him teach the experimental approach because it in some ways simulates the lab experience. The second assignment that he gave to his students, in which

the students were asked to attempt an explanation of the role of microtubules at anaphase using Interdraw, Intertext, and information drawn from Intermedia as well as any other pertinent sources, exemplifies the way in which the lab idea was realized. Drawing upon their general knowledge and other resources, the students were able to come up with solutions to the problem which were possible, and indeed some of which had been postulated formally by scientists who had come to their conclusion through extensive experimentation in the laboratory. Although the students had not been exposed to this particular area, yet, they were able through Intermedia to make connections in the material that they would not have otherwise been able to make based upon the course material they had up to that point. Doing the assignment was yet another way for Peter Heywood to get his students to practice thinking in a nonlinear fashion.

An added benefit of using Intermedia this year for Peter Heywood is that it made lecturing easier for him. He was able to cover more material more thoroughly than he had in previous years simply because the students had access to the lecture notes on Intermedia. He did not have to re-explain things as frequently, and he could spend less time talking about some of the simpler ideas, and more time on the difficult ones. Using Intermedia also had the benefit that it allowed students the opportunity to pick up on the interconnections in the lecture instead of just focusing on the general topic. This in fact reinforced many of the same kinds of connections that the instructor was making.

Professor Heywood feels that Intermedia helped his student to see biology and his class more pluralistically; that it was easier for them to understand how all of the course material was interrelated. That this was so was reflected in the exams and the papers that were turned in this year as compared with previous years. Last year 34% of professor Heywood's class received "A's", as compared to 44% in this year's class. The students were graded both times on the basis of their performance on three exams and the term paper which were all equally weighted at twenty-five percent of their grade. Indeed, for both courses there was a positive correlation between high Intermedia use and high grades ($r=.29$ at .05 level of confidence).

Students apparently felt that they had learned more in the course using Intermedia. On a survey administered to students at the end of the course, 100% of Professor Heywood's students felt that they learned either "more" or "much more" in his course than in the average course at Brown. In the previous year, without Intermedia, only 77% felt this way. Nevertheless the nature of learning was different. **When asked if, given a choice, they would choose a section of a course which used Intermedia over one which did not, 73% of Biomed 106 students and 76% of English 32 students stated that they would choose the section with Intermedia.

Real pluralistic learning?

We began this discussion asking whether cognitive styles could change through education and training, and whether computer instruction could aid or effect this change. In English 32 and Biomed 106 there seems to be convincing evidence that the instructors had clear goals of developing pluralistic thinking in their students, that they had difficulty in achieving their goals using traditional teaching methods, and that use of Intermedia in their opinion significantly improved student performance in areas which they felt indicated movement toward that style of thinking.

We are left with the question of whether use of Intermedia actually helped students to acquire a new, non-linear cognitive style, or merely helped them in their performance in two courses.

Hypertext in general, and Intermedia in particular, can be thought as a model for presenting information, made practical through a particular computer application. One unintended consequence of our research, discovered when Professor Landow was forced to teach English 32 after having prepared Intermedia materials, but before the workstations on which Intermedia was to run were ready for use was that he changed the way he organized his course. As a result he felt that students grasped pluralistic reasoning styles far better than in previous years. Students were also far more satisfied with the course than in previous years (McQuillan 1987).¹⁷

Our research also shows that if acquisition of a new pluralistic cognitive style did occur through Intermedia use, the students themselves may not be aware of it. That some of the students in the biology class were able to grasp Peter Heywood's conception of the pluralistic nature of cell biology is evident by their exam answers, papers and questions in class. However, the extent to which they incorporated this way of thinking into other aspects of their scholarly life remains unknown. Most students seem to adjust to the style of their teachers as the semester goes on because they are interested in doing well in the class. Our observations may reflect this and not actual nonlinear learning.

Intermedia, it might be argued, teaches lineal thinking in the sense that students are following links preset for them by the instructor, and one might argue that this is more lineal than leaving the student to his/her own devices to range freely in a library. However, given only a textbook, the average student will not read footnotes, much less be able to mark out pages with their fingers in the books, notebooks and journal articles that relate nor will he or she go to the library to search for a reference cited. But with these resources "at their fingertips" on Intermedia the average student is more inclined to look at the related material. This in itself is valuable to the student in terms of his/her scholarly development. Further, Intermedia used in this way approaches a model for the way that the instructors want their students to think—nonlineally, in a pluralistic manner. The instructor hopes with the constant practice of seeing material in this way the student will retain this way of thinking even after the course has ended and apply it to other materials.

One early indicator of the impact of Intermedia on students is to gauge how students view a hypertext-based (Intermedia) component in terms of other resources available to them within the course; and how they compare courses containing a hypertext component to courses without such a component. Our student survey data presented in **Tables 1 and 2** (See Appendix) show that Biology students say they learned more and have a better overall opinion of their course than do English students. Biomed 106, a course with an already excellent reputation in the University, seems to be a more improved course in students' eyes.

As discussions in previous sections indicate, there are many uses to which Intermedia could have been put, but it is possible to group students' goals in using (or not using) Intermedia into two categories: performance goals and understanding goals. **Table 3** should make these categories clear. This table also shows that students found Intermedia most useful in helping them broaden and deepen their understanding of course materials, but relatively less useful in achieving performance goals such as getting better grades or avoiding doing other work.

We found that the people who were most involved in the authoring process learned to think about their subject areas more pluralistically by constructing materials for Intermedia. We learned from working with the instructors and the teaching assistants who put the material on the system that there is a learning curve to thinking in this pluralistic way. Both of the professors involved in

the project, as well as the teaching assistants, developed their own thinking as they worked more extensively with the hypertext model.

The rethinking process was long, and the longer the two groups worked with the system, the more ideas they generated about how it should and could be used for their courses. The professors and the teaching assistants were able to grasp "hypertext thinking", and perhaps they benefited more from its nonlinearity than the students did, simply because they were more immersed in using the system and the whole authoring process.

The students didn't utilize the system as much as the professors and teaching assistants, or use it in the same way. They were not forced to make their own links; they followed those laid out by the instructors. Nor did they make additions of their own to the corpus. Their use of the Intermedia system was passive. So it is unclear whether true pluralistic nonlinear thinking occurred for them or whether, as in most of their classes, they merely replicated what they thought the instructor wanted to hear. These instructors wanted to hear their students tying ideas together—ideas which were tied together for them on Intermedia by links.

Intermedia forced the instructors and the teaching assistants who prepared the material to clarify their thinking about the ways in which the various materials with which they were working fit together as an interrelated whole. Just as a first draft of a paper is often not well formulated, the first Intermedia corpora were not as well formulated as they eventually came to be. One English 32 teaching assistant described the process this way: with Intermedia "there were some new ways of conceiving things, forcing myself to think about relationships, literary relationships that seemed a little fuzzy before, and certainly a lot of the stuff that I did was new background research. . . I probably did it in a different way (for Intermedia) than I would have if I had done pure research on my own."

One of the teaching assistants for Biomed 106 commented on the fact that after spending so much time thinking about biology in terms of linking material, that he was unable to resist thinking about his other courses in this way. He said that it became apparent to him that he had succumbed to a new way of thinking upon lending his class notes from his geology class to a friend. Upon returning the notes, the friend asked him what all of the margin markings which said, "link to—" were about. It is evident from this, that working with Intermedia intensively had nonlinear learning effects on him.

We have seen that creating material and making links caused a change in thinking for some of the participants in the project. One of the students who made links for the English corpus thought that it changed the way that she thought about literary materials months later. This change in thinking: the ability to incorporate new material and to draw connections was added for her. In a sense, she learned to think pluralistically.

"The most striking instance of "remembering Intermedia-style" happened when a professor of mine was telling a story about an eccentric colleague. He said "this guy was devoting his life's work to a totally obscure 19th century writer, Max Berenbohm." I said "I'm getting a picture in my mind of a funny little pen and ink caricature. ." "exactly!" says the professor.

I realized later that the picture in my mind was one of the illustrations in the English corpus. It just popped upon the screen in my mind just the way it does on Intermedia—it was a real verbal-visual link. It didn't provide a whole lot of real information, to be sure, but it was there. On a more mundane level, I've experienced countless recognitions that I owe completely to working with the

English corpus. I think the links really do help create a sense of connectedness in my mental image of what "English Lit" is. It is a totally different kind of picture from the typical "reading list" or "great books" approach. For one thing, the sense of network rather than a list makes it easier, in a way, to insert a neglected work or authors in my mind's picture of the canon. It's more adaptable and flexible. Connectedness and difference both fit in."

In sum, for those people (especially the teaching assistants) who used the system extensively, in the way that it was intended to be used, real pluralistic learning occurred. We are not certain whether the students' exposure to Intermedia fostered the same types of learning effects. This may be due to the fact that they used the system passively rather than actively; they did not contribute directly to the corpus. Another possible reason could be that the amount of time they spent working on the system was not as great and their use not as intensive. It is possible that they simply never got to the point on the learning curve where they were able to fully exploit the system.

CONCLUSIONS

It is too early to state conclusively that students using Intermedia were able to acquire a new pluralistic cognitive style. Nevertheless the Intermedia system was clearly a powerful educational tool for destroying the lineality of traditional information in classroom teaching. In the eyes of instructors and students its use promoted greatly increased understanding of classroom materials in a pluralistic mode. The fact that persons involved in creating the materials used in teaching experienced a powerful reorganization of their own thinking regarding these materials suggests that increased use of Intermedia results in cumulative reinforcement of pluralistic, non-linear thinking.

It will be interesting to see if other hypertext systems produce the same cognitive effects in users as they are introduced in teaching and research. Clearly this is an extremely fruitful field for further study.

NOTES

- 1 Thanks to James M. Nyce for valuable aid in the preparation of this paper.
- 2 There are many formulations of the "Sapir-Whorf hypothesis." The most extreme forms posit a direct connection between language form and thought processes. A weaker form emphasizes the shape communication about thought takes when filtered through the channel of a specific language.
- 3 Bernstein emphasizes socialization as the principal determinant of cognitive style as opposed to Piaget who maintains that cognitive styles change with natural processes of human growth.
- 4 Piaget's developmental stages with younger children are often compared with Perry's work in college student development. Cf. Haisty (1983).
- 5 See also Perry and Whitlock (1958) for an earlier formulation.
- 6 This is doubtless conditioned in part by the extraordinary number of colleges founded, run by or affiliated with religious organizations. American institutions of higher education also seem to feel the necessity to embody an educational philosophy unique to each institution in their overall educational structure. In other nations where higher education is centralized and run through the national government, it is a national educational policy which is implemented.
- 7 There is no single ideological frame for this pluralistic integration, only a commitment to moving students toward this style of thinking. Certain religious colleges are quite explicit in their desire for pluralistic education of a special kind. They strive to equip students for the application of a particular religious philosophy in every scholarly and professional discipline, and for every life situation which might be encountered by an adult.
- 8 Ashby's characterization of post-conventional thinking as quoted by Entwistle and Ramsden is as follows:

The student (moves) from the uncritical acceptance of orthodoxy to creative dissent over the values and standards of society . . . (In higher education) there must be opportunities for the intellect to be stretched to its capacity, the critical faculty sharpened to the point where it can change ideas. (Ashby 1973:147-9).

Note also Stephen Ehrmann's broad characterization of the means to achieving these pedagogical goals: "the three academic conversations. (Ehrmann 1987)," consisting of conversations with objects, such as books, pieces of text, diagrams and maps; synchronous conversations with other people, such as in lectures or discussions; and asynchronous conversations with others such as in written communications.
- 9 Lee's characterization derives from a strong formulation of the Sapir-Whorf hypothesis.
- 10 It may be that some languages, such as that of the Trobrianders, facilitate discussion of non-linear thought processes, but we argue that Westerners are quite capable of this kind of thought despite the limitations of Indo-European languages.
- 11 We are not able to really determine whether such a decision matrix exists and is transversed very rapidly, or whether some other structure governs pluralistic cognitive processing. Dreyfus and Dreyfus liken the difference between non-expert decision making and expert decision making to the functioning of a non-compiled vs. a compiled computer program.
- 12 We are reminded of novelist John Barth's notion from *The End of the Road*, "cosmopsis," the ability to maintain contradictory information and beliefs within a single mental frame.
- 13 Perry, *op. cit.* provides a nine-stage schema for this development leading from "simple dualism" through "complex dualism," "relativism," and finally "commitment in relativism," where knowledge becomes integrated with one's own life in a framework of personal choice. Use of the Perry scheme is widespread in the study of student learning

and writing. See Entwistle and Ramsden (1982) and Ryan (1984) for discussion of studies deriving from Perry.

- 14 The Intermedia system is described in detail in Yankelovich 1986 and Meyrowitz 1986. More theoretical statements about Intermedia and its applications are contained in Yankelovich and van Dam 1987; and Yankelovich, Meyrowitz and van Dam 1985.
- 15 Details of preparation of materials for both courses are contained in Yankelovich, Landow and Cody 1986 and Yankelovich, Landow and Heywood 1987.
- 16 The assessment of Intermedia experience was undertaken by the Office of Program Analysis in Iris in conjunction with an outside team of advisors, Rob Kling (team coordinator), Kathleen Gregory-Huddleston, John King, and Kenneth Kraemer. Full details of the assessment plan and its associated research methodologies are contained in Beeman, Gregory-Huddleston, King, Kling and Kraemer 1986.
- 17 In Tables 1, 2 and 3 in the Appendix to this paper, the three repetitions of English 32 are referred to as English 32a, 32b and 32c. Only in English 32c did students use Intermedia.

BIBLIOGRAPHY

Ashby, E.

- 1973 The structure of higher education: a world view, *Higher Education*, 2 : 142-51.

Barth, John

- 1958 *The End of the Road*. New York: Doubleday & Company.

Beeman, William O., Kathleen Gregory-Huddleston, John King, Rob Kling and Kenneth Kraemer

- 1986 Assessment Plan for a Network of Scholar's Workstations in an University Environment: A New Medium for Research and Education. Submitted to the Annenberg/CPB Project, March.

Bernstein, Basil

- 1971-5 *Class, Codes and Control*. 3 vols., London: Routledge and Kegan Paul.

Clark, R.E.

- 1983 Reconsidering Research on Learning from Media. *Review of Educational Research*. 53(4)
- 1985a Confounding in Educational Computing Research. *Journal of Educational Computing Research* 1(2)
- 1985b The Importance of Treatment Explication: A Reply to J. Kulik, C-L Kulik and P.A. Cohen.

Dreyfus, Hubert L. and Stuart E. Dreyfus

- 1986a Putting computers in their place." *Social Research* 53(1): 57-76.
- 1986b *Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*. New York: Free Press.

Ehrmann, Stephen C.

- 1987 Hypertext as a Medium for The Three Academic Conversations. Paper submitted for Hypertext Workshop, University of North Carolina, November 13-15, 1987.

Entwistle, Noel J. and Paul Ramsden

- 1982 *Understanding Student Learning*. London and New York: Croom Helm and Nichols.

Haisty, Donna B.

- 1983 *The Developmental Theories of Jean Piaget and William Perry: An Application to the Teaching of Writing*. Ph.D. dissertation, Department of English, Texas Christian University, Ft. Worth Texas.

Heath, Roy

1964. *The Reasonable Adventurer*. Pittsburgh: University of Pittsburgh Press.

Kohlberg, Lawrence and Rochele Mayer

- 1972 Development as the Aim of Education. *Harvard Educational Review* 42: 449-96

Kulik, J.A., C-L Kulik and P.A. Cohen

- 1980 Effectiveness of Computer-based College Teaching: A Meta-analysis of Findings. *Review of Educational Research* 50(4): 525-544.

Kulik, J.A., C.C. Kulik and R.L. Bangert-Drowns

- 1985 The Importance of Outcome Studies: A Reply to Clark. *Journal of Educational Computing Research* 1(4): 381-387.

Lee, Dorothy

- 1959 Codifications of reality: lineal and non-lineal. In *Freedom and Culture*. Englewood Cliffs: Prentice Hall. pp. 105-120.

Meyrowitz, Norman

- 1986 *Intermedia: The Architecture and Construction of an Object-Oriented Hypertext/Hypermedia System and Applications Framework*. OOPSLA '86 Proceedings. Portland, Oregon, September 29-October 2, 1986.

McQuillan, Patrick J.

- 1987 *Computers and Pedagogy: The Invisible Presence*. Working Paper, Office of Program Analysis, Institute for Research in Information and Scholarship, Brown University.

Perry, William G.

- 1968 *Forms of Intellectual and Ethical Development in the College Years: A Scheme*. New York: Holt, Rinehart and Winston.
- 1976 *Students as Makers of Meaning*. *Higher Education* 5: 125-132.

Perry, William G. and Charles P. Whitlock

- 1958 *Of study and the Man*. *Harvard Alumni Bulletin*. (February): 1-13.

Piaget, Jean

- 1950 *The Psychology of Intelligence*. Trans. Malcolm Piercy and D.E. Berlyne. London: Routledge and Kegan Paul.
- 1959a *The Language and Thought of the Child*. 3rd ed. Trans. Marjorie and Ruth Gabain. London: Routledge and Kegan Paul.
- 1959b *Judgment and Reasoning in the Child*. Trans. Marjorie Warden. Patterson, N.J.: Littlefield, Adams.

Ryan, Michael P.

- 1984 *Monitoring Text Comprehension: Individual Differences in Epistemological Standards*. *Journal of Educational Psychology* 76 (2): 245-258.

Sapir, Edward

- 1924 *The Grammarian and His Language*. *American Mercury* 1: 149-155
- 1949 *Selected Writings of Edward Sapir*. David G. Mandelbaum, ed., Berkeley: University of California Press.

Whorf, Benjamin

- 1956 *A Linguistic Consideration of Thinking in Primitive Communities*. In John B. Carroll, ed. *Language, Thought and Reality: Selected Writings of Benjamin Lee Whorf*, pp. 65-86. Cambridge: MIT Press.

Yankelovich, Nicole

- 1986 *INTERMEDIA: A System for Linking Multimedia Documents*. IRIS Technical Report 86-2, Institute for Research in Information and Scholarship, Brown University, Providence, RI, 1986.

Yankelovich, Nicole and Andries van Dam

- 1987 *Spinning Scholarly Webs*. The Annenberg/CPB Project Report to Higher Education. The Annenberg/CPB Project, Washington, D.C. 1987.

Yankelovich, Nicole, George P. Landow and David Cody

- 1986 *Creating Hypermedia Materials for English Literature Students*. Institute for Research in Information and Scholarship and Department of English, Brown University, Providence, RI, October 1986.

Yankelovich, Nicole, George P. Landow and Peter Heywood

1987 Designing Hypermedia Ideabases—the Intermedia Experience. Paper submitted to the Annenberg/CPB Project, August 30, 1987.

Yankelovich, Nicole, Meyrowitz, Norman and Andries van Dam

1985 Reading and Writing the Electronic Book. *Computer* 18(10) [October]: 15–30.

APPENDIX

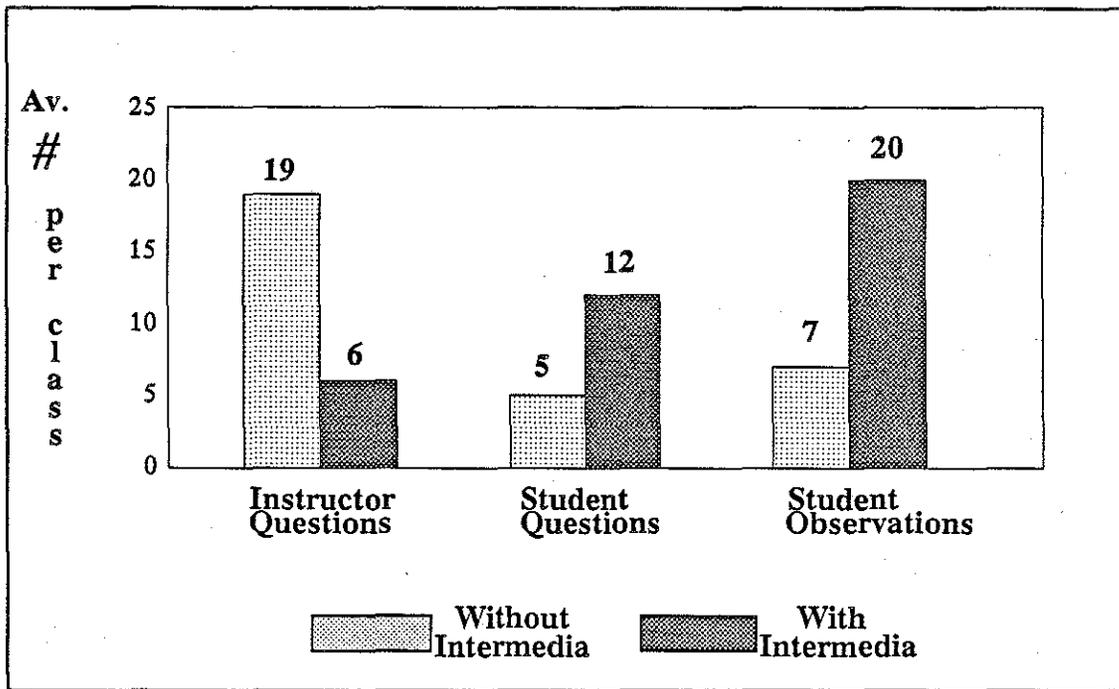


Chart 1

Effect of Intermedia Use on English 32 Class Discussion

TABLE 1
Amount Learned in Course
Compared to Other Courses Taken at Brown

	Much More	More	About Average	Less	Much Less
ENGLISH 32a	.0%	19.2%	53.8%	11.5%	15.4%
ENGLISH 32b	11.5%	42.3%	34.6%	7.7%	3.8%
ENGLISH 32c	10.0%	40.0%	36.7%	13.3%	.0%
BIOMED 106a	22.2%	55.6%	22.2%	.0%	.0%
BIOMED 106b	27.3%	72.7%	.0%	.0%	.0%

TABLE 2
Overall Evaluation of Course
Compared to Other Courses Taken at Brown

	Much Better	Better	About Average	Worse	Much Worse
ENGLISH 32a	7.7%	23.1%	34.6%	23.1%	11.5%
ENGLISH 32b	15.4%	38.5%	30.8%	11.5%	3.8%
ENGLISH 32c	10.0%	36.7%	26.7%	23.3%	3.3%
BIOMED 106a	27.8%	50.0%	16.7%	5.6%	.0%
BIOMED 106b	45.5%	45.5%	9.1%	.0%	.0%

TABLE 3: Opinions about Intermedia
 Biomed 106b=regular face type; N=11
 English 32c=bold face type; N=30

RANK			NO, NOT REALLY	YES, SOMEWHAT	YES, DEFINITELY	N/A
4	1	reinforced the lectures	18% 7%	27% 60%	55% 30%	3%
2	2	deepened understanding of course material	18% 7%	46% 47%	36% 43%	3%
3	3	helped your understanding	27% 7%	46% 60%	27% 33%	
5	4	helped prepare for exams	36% 17%	9% 50%	55% 33%	
1	5	taught things not found in other sources	36% 3%	46% 33%	18% 60%	3%
6	6	increased class participation	55% 53%	18% 37%	9% 10%	18%
7	7	improved your grades	64% 53%	9% 30%	18% 10%	9% 7%
8	8	replaced other course readings	73% 82%	27% 18%	0% 0%	

Experiences and Writing

Hypertext Habitats: Experiences of Writers in NoteCards

Randall H. Trigg and Peggy M. Irish

Intelligent Systems Laboratory
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

ABSTRACT

This paper reports on an investigation into the use of the NoteCards hypertext system for writing. We describe a wide variety of personal styles adopted by 20 researchers at Xerox as they "inhabit" NoteCards. This variety is displayed in each of their writing activities: notetaking, organizing and reorganizing their work, maintaining references and bibliographies, and preparing documents. In addition, we discuss the distinctive personal decisions made as to which activities are appropriate for NoteCards in the first place. Finally, we conclude with a list of recommendations for system designers arising from this work.

1. INTRODUCTION

There is an apparent contradiction inherent in the notion of using hypertext for writing papers.¹ To be sure, there is the possibility that in the future readers will access "soft" documents online. The "published" form of a document could then be a network of interlinked chunks of text, graphics, etc. [Trig83, Nels81]. However, for today and the foreseeable future, the audiences for our papers will not access them via a computer, but rather by reading hardcopy renderings. This means that we need the final product of our writing efforts to be a traditional, predominantly linear, paper document. Nonetheless, we have found hypertext to be quite well-suited to supporting the writing process. This is especially true when writing is viewed broadly as consisting of a range of activities beyond simple text composition. These include among others: notetaking, organizing and structuring, outlining, and maintaining references and bibliographies.

To investigate the topic of writing in hypertext, we conducted a series of interviews of researchers at Xerox using NoteCards, a popular hypertext system available on Xerox workstations. (See Hala87 for a detailed description of the system and its design rationale.) In addition to providing a powerful interactive hypertext framework, NoteCards allows its users tremendous freedom to create and adopt personal styles. Many of the users we interviewed spend a significant portion of their online time doing research. Because of the extended scope of their work and the relatively unconstrained nature of NoteCards, they evolve for themselves online "habitats". In the same way that a person's office often has a "lived in" feel, these users' online environments reflect their personal styles as well as the substance of their work.

The writers discussed here are 20 researchers who have used NoteCards for one or more writing tasks, usually on several papers. They come from a wide variety of backgrounds and include anthropologists, physicists, linguists and computer scientists. Some are accomplished Lisp programmers while others have had no programming experience and use the computer predominantly for writing and electronic mail. The writing projects vary in size from short papers and presentations to Ph.D. theses. With nine of these writers, we tape-recorded in-depth interviews which included extensive "guided tours" of each of their hypertext environments. We also had access to the writers' notefiles (i.e. files containing their hypertext networks). The figures appearing in this paper are taken from those notefiles.²

An important feature of this study is that each of the writers discussed here is a voluntary user of NoteCards. That is, their use of NoteCards is neither one of their job requirements nor part of a writing "task" imposed by us. Rather, these researchers freely choose to do their personal writing projects in NoteCards and can leave it at any time. In that sense, this study is naturalistic rather than experimental. That is, it reflects what people are doing independent of the study. Furthermore, our writers select for themselves the particular writing activities that are appropriate for NoteCards and those that are not. In fact, we have included two cases of writers who worked for some time in NoteCards and found it sufficiently unsatisfactory that they were forced to discontinue use altogether. The experience of such users who "leave the fold" is particularly valuable as it points out dramatically those parts of the system in need of improvement. More generally, it affords us the opportunity to investigate those writing activities, styles and situations that are perhaps *not* as appropriate for hypertext support.

The primary goal of this paper is to describe the variety of writing activities our users elect to perform in NoteCards and the wide variety of styles they employ. Our hope is that this information will be useful to our fellow hypertext system designers who intend to support the writing process. In Section 2, we provide an overview of the fundamental concepts in NoteCards, concentrating on those of particular relevance for writers. In Section 3, we discuss the user-specific limits of NoteCards, that is, which activities our users found natural to do in NoteCards, and which were best done elsewhere. The next four sections discuss particular writing activities grouped into the loose categories of: notetaking, gathering and maintaining references, structuring and organizing, and creating a document. Note that the order of these sections is not meant to imply that the corresponding activities constitute successive stages of writing. On the contrary, these groups of activities are generally interleaved, each likely to occur at any time during a writing project. Finally, we conclude with a discussion of the implications of this work for hypertext system designers, and recommendations for future research.

2. OVERVIEW OF NOTECARDS

NoteCards is an extensible computer environment designed to help people manipulate and structure information [Hala87]. The basic NoteCards construct is a semantic network made up of a set of *notecards* interconnected by typed *links*. This network may be organized, displayed and managed by tools provided by the system. NoteCards also provides a set of methods and protocols for creating programs to manage the network.

A *notecard* (or sometimes just *card*) is an electronic analogy of a 3x5 index card. Every notecard includes a title and an arbitrary amount of an editable "substance," such as a piece of text, a structured drawing, or a graph representation. Some examples of cards are shown in Figure 1. Different kinds of notecards are defined in an inheritance hierarchy of card types (eg., text cards, sketch cards, graph cards, etc.) "Bringing a card up" from the database, i.e. displaying the card, activates an editor appropriate to its card type; text cards activate a text editor, sketch cards a sketch editor, and so on. Multiple cards can be displayed simultaneously, each in an arbitrarily sized window. NoteCards contains a set of standard card types, along with a facility for adding new types of cards ranging from small extensions of existing types, to those based on entirely different substances (e.g., video cards).

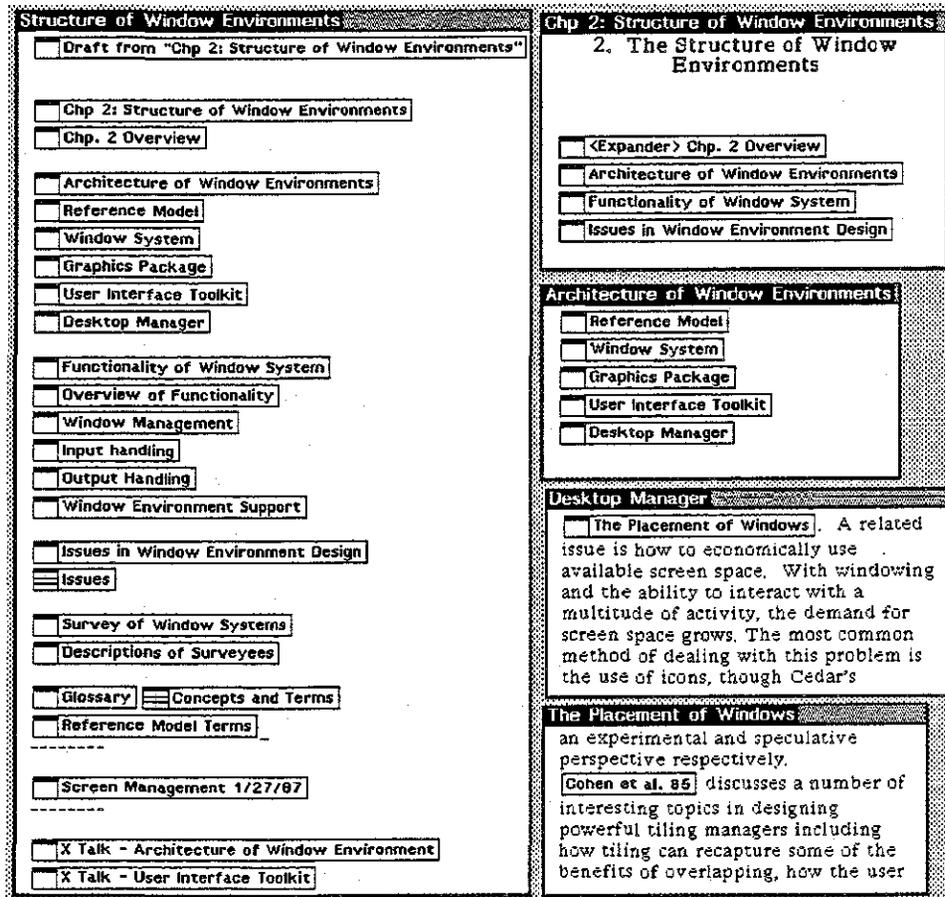


Figure 1. Cards, fileboxes and links.

Cards can be connected to other cards by arbitrarily typed, directed *links* to form a network. Links are used both to organize a network and to navigate through it. A link type is a user-specified label signifying the nature of the relationship between two cards. It is usually up to the user to decide how to take advantage of link types to organize the network. An individual link appears as a link icon (typically a boxed title) in the body of its source card, as shown in Figure 1. Selecting the link icon with the mouse causes the link to be traversed and a window containing the link's destination card to be opened.

The *filebox* is a specialized card type provided to help users manage large networks. Fileboxes provide a hierarchical filing structure that can be used to cluster and organize collections of notecards. NoteCards requires that every card (including fileboxes) be filed in one or more fileboxes, and that the filebox structure form a true hierarchy (i.e., a directed acyclic graph). Fileboxes may contain text, cards and other fileboxes, as shown in Figure 1.

A *browser* card displays a structural diagram of a network of notecards. The browser contains a graph, the nodes of which are link icons representing cards in the network and the edges of which are lines representing links between cards. (Nodes can also be simple textual labels.) The dashing styles of the lines appearing in the browser represent link types. Browsers can be edited in two ways. First, nodes and edges can be added, deleted or rearranged in the browser diagram without affecting the corresponding cards and links. For example, the vertical spaces between nodes in the browser shown in Figure 2 were added by the user to reflect logical groupings. Second, new cards and links can be created or existing ones deleted using operations available through the browser card interface. Both sorts of editing operations appear in the menu attached to the lower right corner of the browser in Figure 2. Another feature of the browser card is the *browser overview window*, which displays the contents of the entire browser scaled to fit into a window, and appears to the left in Figure 2. The position of the wire frame in the overview indicates the portion of the browser appearing in the main window; "dragging" it with the mouse causes the main window to scroll.

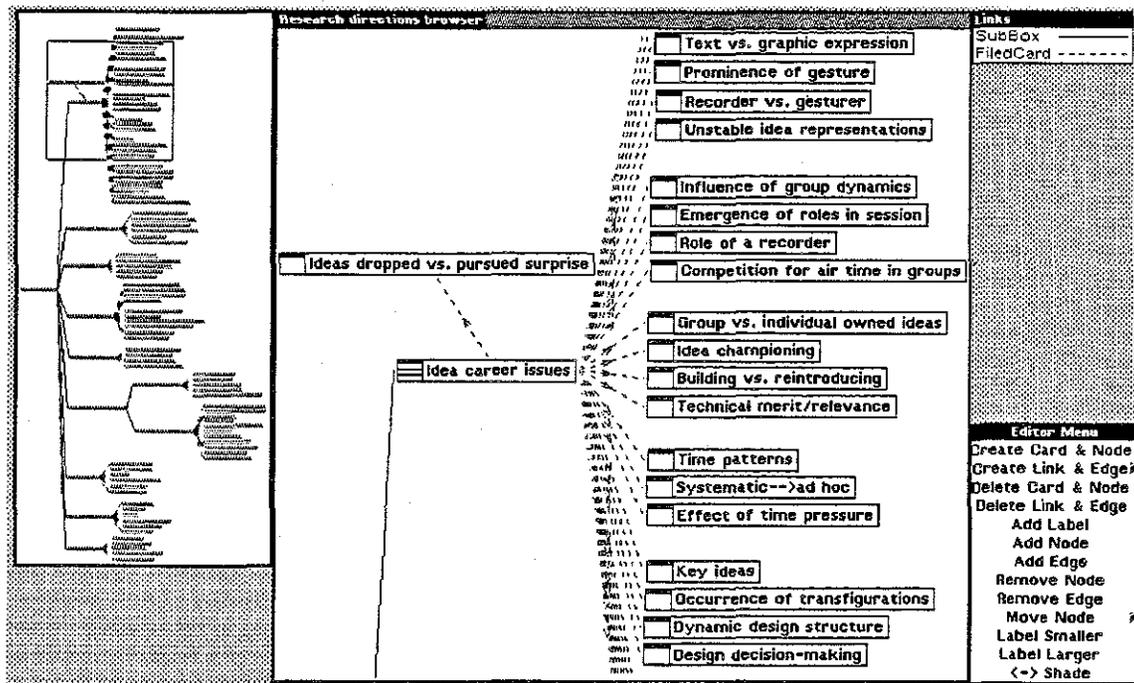


Figure 2. Browser card.

A *document* card contains a linearization of a portion of the network specified by the user. When creating a document card, the user selects a starting card, the set of link types to follow, and certain formatting options. NoteCards then traverses the resulting network and places in the document card the contents of each card it encounters. The end effect is of collapsing a network of individual linked chunks of text and graphics into a linear document.

All information pertaining to a network of cards and links is stored in a *notefile*, a structured file managed transparently for the user by NoteCards. A single user can simultaneously access any number of notefiles, and can create and traverse special links (known as *cross-file links*) connecting cards residing in different notefiles. The boundaries of a notefile can be relaxed further using *file cards*. A file card is a card whose substance is simply the name of a file. When a link is followed to a file card, NoteCards finds the file with that name and opens it in a text editor window. File cards were designed to allow work to exist in a file external to the notefile, and yet still be made available from within the notefile.

Finally, NoteCards contains a programmer's interface of more than 150 lisp functions that allow the user to create, access, and modify all NoteCards objects. Functions are provided for modifying the characteristics of individual cards and links, and for building new structures. This interface also gives users the ability to create new card types, modify the user interface to NoteCards, and specify global actions over an entire notefile. It is designed to support both small modifications and large system developments.

3. WHEN TO USE NOTECARDS

An important goal of this study was to investigate the decisions of writers as to the appropriateness of NoteCards for their particular work. In general, this decision is not made once and for all. Most revisit the issue each time they begin to write a paper. Furthermore, for different writers the NoteCards medium is found to be appropriate for different subactivities of the writing process. For example, some writers do early notetaking and data gathering in NoteCards, electing to move out of the system for final document preparation. Others only enter NoteCards when they perceive the need to structure or restructure parts of their paper.

When to enter NoteCards?

The first decision writers make is whether to try NoteCards at all. The outcome presumably depends on such factors as advertising, recommendations from friends, general curiosity, etc. Some of our writers played with NoteCards on toy problems before using it for serious work, while others jumped in with a real project at the outset. In addition, several of our writers had already used NoteCards for non-writing tasks and were thus familiar with its general capabilities.

Beyond the initial decision to try NoteCards, our writers decide for each particular paper or research project whether to do that work at least partially within NoteCards. For the most part, these decisions are based at least in part on personal experience with the system (rather than simply hearsay). We found three factors to be relevant to the decision of whether to write all or part of a new paper in NoteCards. The first concerns the complexity of the work to be done. Papers for which the organization or structure is already known seem less appropriate for composition in hypertext. These are papers that can be generated using a single instance of the text editor in a window. Most of the work involves tinkering with prose rather than large scale structuring.

The second factor involves the degree to which the paper borrows from previous work already available in NoteCards. In the extreme case, the paper is seen as part of or closely related to a larger project, and merely requires assembling pieces of prior work. At the other

extreme, the paper is on a new topic and none of the author's prior work in NoteCards can be seen to be relevant. Note, however, that if prior non-NoteCards writing is relevant, then the idea of creating a new notefile containing both the old work and the new is worth considering.

Finally, the overhead for using NoteCards must be considered. Assuming the NoteCards system is already loaded in the user's environment (this happens automatically for most users), the costs for starting work on a new paper consist merely in creating and opening a new notefile. Once work is underway, however, NoteCards imposes some overhead as compared to say, a single text editor. Mont86b discusses some of the requirements made on the user including segmenting material into cards, card titling and filing, etc.

In most cases, some combination of these three factors is at work. Usually, the writer weighs the overhead of using NoteCards against its benefits, primary among these being support for structuring and restructuring the paper. For example, consider the following case. One user had been working outside of NoteCards, using the text editor to create multiple files containing descriptions of experiments to be included in a paper on music perception. The window on the left side of Figure 3 displays a system filebrowser over these files. He subsequently brought some of these files into NoteCards in order to manage versions and changes, and to help in rewriting the introduction to include a literature review. He realized that the introduction would contain many references and felt that NoteCards would support organizing and linking these references into the paper. He also preferred to view and edit his sections as cards in a filebox rather than as text editor files on a server. The two cards on the right side of Figure 3 show a portion of the resulting NoteCards organization.

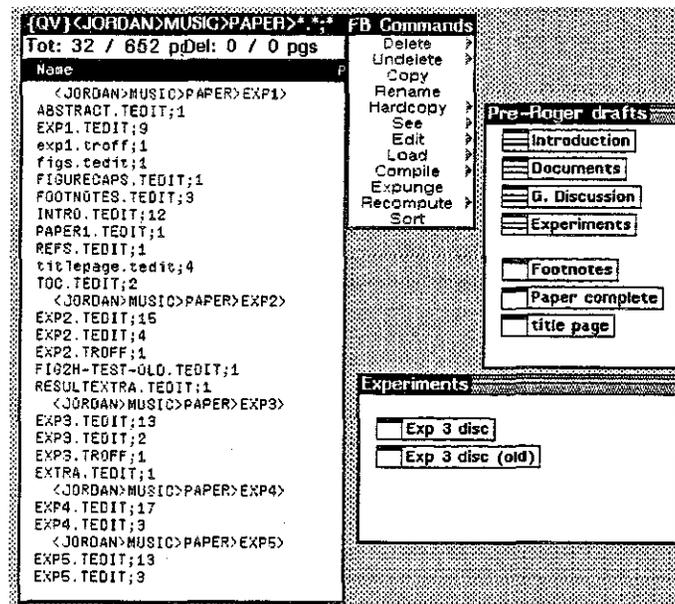


Figure 3. Paper structuring inside and outside NoteCards.

Often it is the case that papers with short time horizons are less likely to be written using NoteCards, perhaps because there is less time to take advantage of the hypertext environment for reorganizing and restructuring one's work. In this case, however, restructuring was precisely the purpose of the move into NoteCards. We were told by this writer that the fact that he had less than two weeks to complete the paper argued in favor of

using NoteCards; he didn't see how he could reorganize the paper in such a short time without it.

For projects involving long papers and extended research, the tradeoffs are sometimes different. On the one hand, such projects usually involve restructuring on both large and small scales. At the same time the length of the paper makes it likely that the structure will be more complex. Each of these argues for the appropriateness of embarking on the project in NoteCards. On the other hand, large projects tend to magnify the risks of using NoteCards (or any computer environment); the more one is invested, the harder it is to "bail out."

"Falling out" of NoteCards

At some point in the writing process, many writers find it necessary to move out of NoteCards. This happens most often during the final document preparation phase. Once the organization of the paper has been determined and a draft exists, smoothing the text is sometimes done outside of NoteCards in a single text file. It seems that the major reason for this is the increased accessibility of the document; changes can be made, comments integrated and hardcopies generated without having to open the notefile.

For example, towards the end of writing a Master's thesis in NoteCards, one writer made external text files out of the chapters in his notefile, and worked on these files for several weeks until the chapters were in final form. He even changed one chapter significantly, splitting it into two new chapters. Note, however, that this work was later brought back into NoteCards. Indeed, the process of using NoteCards is more complex than a simple 3-stage model of enter, work, and fall out. For example, earlier in the course of that same user's work, he wrote a thesis proposal in NoteCards. Near the end of that period, he moved it into an external text file for smoothing. Later, as he started work on the thesis proper, he brought the proposal back into NoteCards breaking it up to allow piecewise access.

A special card type called file card (see Section 2) was added to NoteCards precisely to help in such situations. One writer did, in fact, make extensive use of file cards. As in the case described above, he moved his document from NoteCards into an external text file for final smoothing. Upon completion of this work, he created a file card in the notefile pointing to the completed document. This allowed access to the final copy both from inside and outside NoteCards.

Nonetheless, such facilities are insufficient to solve the accessibility problem in general. One of the researchers who left NoteCards altogether did so for just this reason. This individual was one of our most serious users having written several papers in NoteCards, as well as using it for various non-writing projects. His major complaint is that cards in a notefile (other than file cards) are not generally accessible except through NoteCards. This is due to the fact that all cards and links in a notefile are stored together in a single file, a situation that is probably exceptional among hypertext systems. In Unix-based hypertext systems for example, each card (or node) is stored in a separate file [Trig86a, Deli86]. This presumably allows access to individual cards without loading in the network.³

4. NOTETAKING

There are a set of activities that are common elements of doing research, though not strictly part of the process of composing text for a final document. They usually take place before writing begins and include: taking notes from reference books and articles, gathering data from various sources, and generating notes "from one's head." Given that these notes are at best loosely structured, hypertext can be an ideal environment for creating and organizing them. In our study, we found notetaking to be a fairly common activity among our writers, particularly those engaged in longer term projects for which the form and even the focus of a final paper were not initially known. In this section, we look at the decision to use NoteCards to store particular notes, as well as styles of organization for those notes that do make it into NoteCards.

There are two primary reasons our writers give for deciding not to put certain notes online. Both involve trading off the work involved against perceived benefits of having the notes available at the time the paper is written. One problem is that notes sometimes have a graphical component which can be difficult to transfer online. In the case of one writer, the notes contain complex equations. Although there are facilities for laying out equations in the text editor (and thus in a text card), using these takes far more time than writing them out by hand. This writer finds entering equations into the computer to be too much work unless he is confident that they will appear in the paper. Similar arguments apply to quick diagrams and flowcharts.

A second problem for some writers arises when they aren't sure where a particular note fits into the current structure of the notefile. One writer stated that he likes everything in his notefile to fit into a structure, preferring not to have cards in the notefile that are "unorganized". As evidence for this, he generally goes to some lengths to clean up the contents of his ToBeFiled and Orphans fileboxes.⁴ Notes that don't easily fit into an appropriate place in the notefile are written in an offline notebook instead and only later inserted into the notefile if their relevance becomes apparent.

Other writers, however, develop schemes and conventions in NoteCards especially to handle notes whose precise resting place in the notefile is not obvious. One writer uses a "WhiteBoard" filebox similar to a physical whiteboard. This filebox contains old cards, cards without a home, and loose text. Periodically he sorts through this box, filing cards and moving text into new or existing cards. In the interview, he compared these actions to copying notes and figures from a whiteboard to paper, and then erasing the whole board to start over. This same user has a second mechanism he calls "bins" for storing some of his notes. A bin is an unordered, unstructured filebox containing cards and boxes related to some general topic. Like many other users, he prefers not to let unfiled cards build up in the ToBeFiled and Orphans fileboxes. This writer occasionally looks through the bins in his notefile to see what might be of use in his writing. Another writer has a "Miscellaneous Thoughts filebox," shown in Figure 4, where he puts cards not yet categorized. As soon as he sees how a card fits into his paper, he files it in one of the fileboxes making up the paper "outline".

The situation of having notes with no place to put them has a flip side; namely, that of having fileboxes in a structure and no notes to fill them. This is relevant for writers who occasionally adopt a top-down approach to notetaking, creating topical fileboxes to contain

notes that don't yet exist. These fileboxes become a sort of agenda of work to be done. For example, one of our writers is in the process of taking notes from a set of audio tapes. Each tape has a corresponding filebox in the notefile, though several are empty. Similarly, this user keeps empty fileboxes meant to contain notes from books and articles which he hasn't yet read. For example, he created an empty "Sociolinguistics" filebox to contain notes on materials relating to that subject. This filebox appears on the right side of Figure 4, along with the filebox in which it is filed.

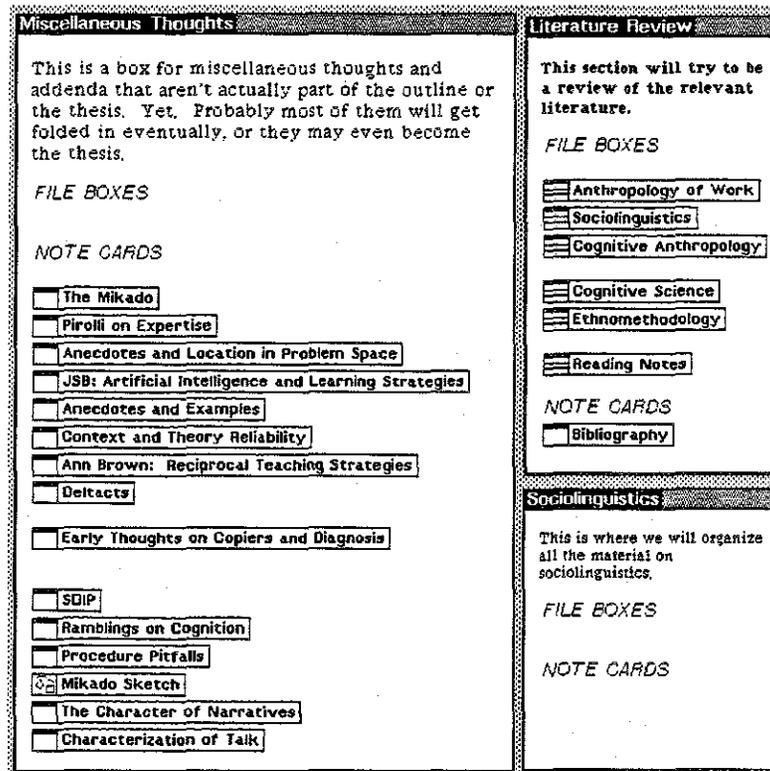


Figure 4. "Miscellaneous Thoughts" and top-down filebox creation.

One special kind of notetaking involves the articulation of thoughts preparatory to writing a paper and has often been called *brainstorming* [Osbo79]. Rather than face an empty pad of paper (or text card window), the brainstormer jots down thoughts on small pieces of paper (or electronic notecards) in order to get started. These notes act as seed material for the paper. Among practitioners of this method in NoteCards, we found two distinct styles. The first involves the use of a text editor window in which to write phrases or short sentences that capture relevant ideas. These bits of text are moved around in the card in order to reflect some appropriate ordering. Changes to font size and face are also used as a way of organizing the thoughts.

The other scheme uses a NoteCards browser (see Section 2) to easily create cards having short phrases as titles. These titles appear in link icons in the browser and can be arranged spatially on the canvas to capture grouping, etc.⁵ Links can also be created between the new cards from within the browser, though we found few instances of that behavior. Rather, the common practice involves creating new cards and moving their link icons around for grouping

or to vaguely represent an outline. At some point, the user brings up one of these cards and enlarges on the idea represented by the card's title with text (or graphics). Figure 5 contains the "Thoughts Browser" used by one writer to organize ideas for a paper. The cards that were generated while brainstorming in the browser, as well as the browser itself, are all filed in the filebox shown on the left side of the figure.⁶

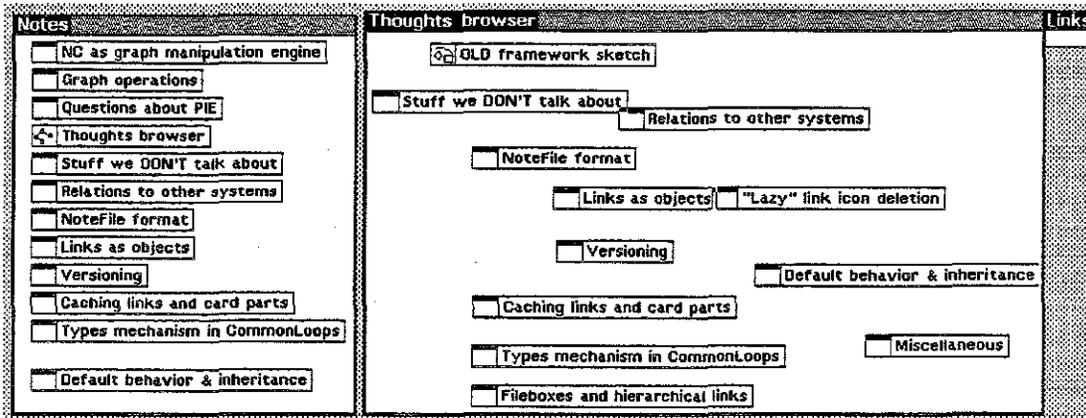


Figure 5. Brainstorming browser.

5. REFERENCES AND BIBLIOGRAPHIES

Another set of activities important for both research and writing pertains to managing references. These activities can take place during the entire writing process and include: jotting down the name of a reference, recording a full bibliographic entry, maintaining references, inserting a reference into a document, and generating a bibliography. Many of these tasks require attention to organization, and hypertext provides the necessary organizing capabilities.

We found a wide range of complexity in the reference-handling styles of the writers we studied. Not surprisingly, we found that the extent to which different users work at coping with references in NoteCards depends on the size of their reference "databases," ranging from less than 10 references to more than 150. In this section, we take a closer look at the methods used by some of our writers for handling references in NoteCards.

A number of writers developed sophisticated structures to aid in the task of maintaining and organizing large numbers of references. One example is the system employed by one user while writing his Master's thesis in NoteCards. For each of his references, he creates a text card titled with the author's name and the year of publication, and places the bibliographic entry into the card's contents. These reference cards are linked into his thesis with Reference links, one of which appears in the text of the card in the lower right corner of Figure 1. One of his reference cards appears in the bottom window in Figure 6. He files all of these reference cards in one large filebox, called "Bibliography Bin," shown in the window on the right of Figure 6. This filebox is loosely organized by ordering the cards and by adding line breaks to indicate subject breaks. As an aid in generating a bibliography for each of his thesis chapters, this writer links all of the references for a chapter into one bibliography card, as can be seen in the "Window Environment Bibliography" card in Figure 6. This card is later used to construct the bibliography for the chapter by means of a Draft card (see section 7) containing the

contents of the cards linked to by this card. The filebox in the upper left corner of Figure 6 contains all of the bibliography cards for the chapters. The three cards in the top line of this filebox are used to store reminder notes falling into three categories: the references on order from the library, the references he's considering obtaining, and those he is in the process of acquiring on his own.

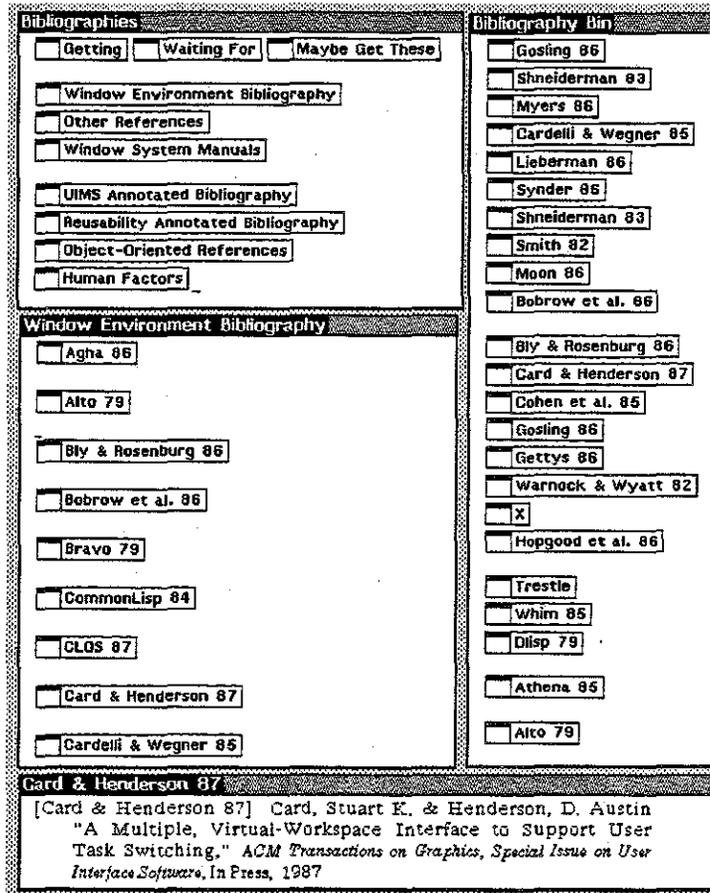


Figure 6. Reference cards and bibliographies.

Another writer chooses to take a slightly less structured approach to the task of maintaining references and generating a bibliography. In the course of notetaking, he creates a reference card, titles it with its author and title (or abbreviated title), and places notes taken from the reference into the card. He then updates a running bibliography by typing the bibliographic entry for this reference into a Bibliography card containing entries for all of his references. To simplify the creation of the entry, he stores several copies of a sample entry in this card, so that at least one is "within reach" at all times. For ease in locating a particular reference in the Bibliography card, he groups the entries under subject headings. Although this writer constructs his bibliography entirely by hand, it isn't an unmanageable task because he adds to it only occasionally.

Some writers have made extensive use of the NoteCards programmer's interface to customize NoteCards to their particular information management needs. This becomes apparent when we examine the programs they have written for creating and maintaining

references as well as bibliographies. One writer, in particular, has created several programs to help him with this task. His primary motivation for implementing these programs was the large number of references he maintains for one paper in particular (more than 150). As an aid to creating reference cards, he defined his own Reference card type, which upon creation contains the fields of a bibliographic entry ready to be filled in. At that time, it also automatically prompts him for a filebox and a title, in the form of author and year. Once a Reference card is created, a link to it may be inserted into the text using another of this user's programs. This program allows him to choose an item from a menu which prompts for a reference card. It then inserts, into the text currently being edited, a link to that reference card. To facilitate the coercion of references from the form of link icons to numbers in the text of a paper, and to generate a bibliography, he created another program using the programmer's interface. This program runs through a document card containing a paper and collects all of the links to reference cards. It orders this list of references, either by their occurrence in the document or alphabetically, and assigns numbers to each of them. It then makes another pass through the document card, replacing each link to a reference card with its assigned number. Finally, it generates the actual bibliography from this collected list of references.

Another writer working on a Ph.D. thesis also created a program using the programmer's interface to partially automate reference tasks related to his literature search. His system for maintaining references includes: creating a filebox for each article, inserting a bibliographic reference (in proper format) as well as a summary of the article into the filebox, and titling the filebox with the title of the article. If he finds specific items of interest in the article, he places them in separate notes and files them in the filebox. These notes contain Source links back to the article filebox, so that the reference can be found when only the note is on the screen. His program semi-automates this process by: creating a blank notecard; filing it in a specified article filebox and inserting a "back" Source link; and filing it in the Notes filebox (which contains all notes, unsorted). He still must manually title the notecard and fill in its contents, but the organizational part of the task is accomplished by his program.

Some of the writers have a tendency to refer to only a few sources in their papers, and their papers are typically shorter than those of the writers whose reference styles are described above. These people do not use complicated reference organization schemes because they feel such schemes incur too much overhead for their application. One such writer chooses to simplify the reference-tracking procedure by titling reference cards with the titles of the sources, and by not bothering to store the full bibliographic entries for the sources in his notefile. Instead, he looks up the bibliographic information when it comes time to generate the bibliography at the completion of the paper. He also chooses to keep all of his reference cards in one Source filebox; since he doesn't have many, he doesn't bother to order them. Several users don't bother with reference schemes at all. They simply insert references into their text and manually generate a bibliography when the document is complete.

6. STRUCTURING AND RESTRUCTURING

One of the well known strengths of hypertext systems is that they allow multiple organizations of the same information to coexist and even evolve in parallel. Indeed, we found

widespread "redundancy" (to use the term of one of our writers) in those notefiles that were part of longer term projects. That is, the same cards are filed in multiple fileboxes or linked to by several other cards. For example, Figure 1 shows two parallel organizations of a set of cards on the structure of window environments. The filebox on the left contains all of the cards relevant to this topic, while the cards on the right are part of the multilevel structure of the paper, constructed by the writer with links. Note that three of the four cards on the right are filed in "Structure of Window Environments."

What has perhaps been less widely acknowledged is the need for multiple structures when converting from one organization to another. We found that none of our writers perform instantaneous reorganizations of their notefiles. Rather, changeovers are gradual often leaving part of the information in the old structure (or in both old and new), while more recent information is filed and linked according to the new structure. One of our writers displays this sort of gradual conversion in his thesis notefile. He plans to write his thesis section by section working from a structured "outline" filebox. He grew dissatisfied with his first outline, but rather than modify it, he created a new version by copying those parts of the old outline likely to remain relatively constant. (Corresponding sections of the two outlines are shown in Figure 7; the original appears on the left, the current version on the right.) Some of the new outline is less developed and less inter-linked than the original, but it better reflects his current thinking. In short, the present state of his thinking on the thesis structure can be captured by neither of the outline cards alone, but rather by the parallel alternative views and by the gradual progression from one to the other.

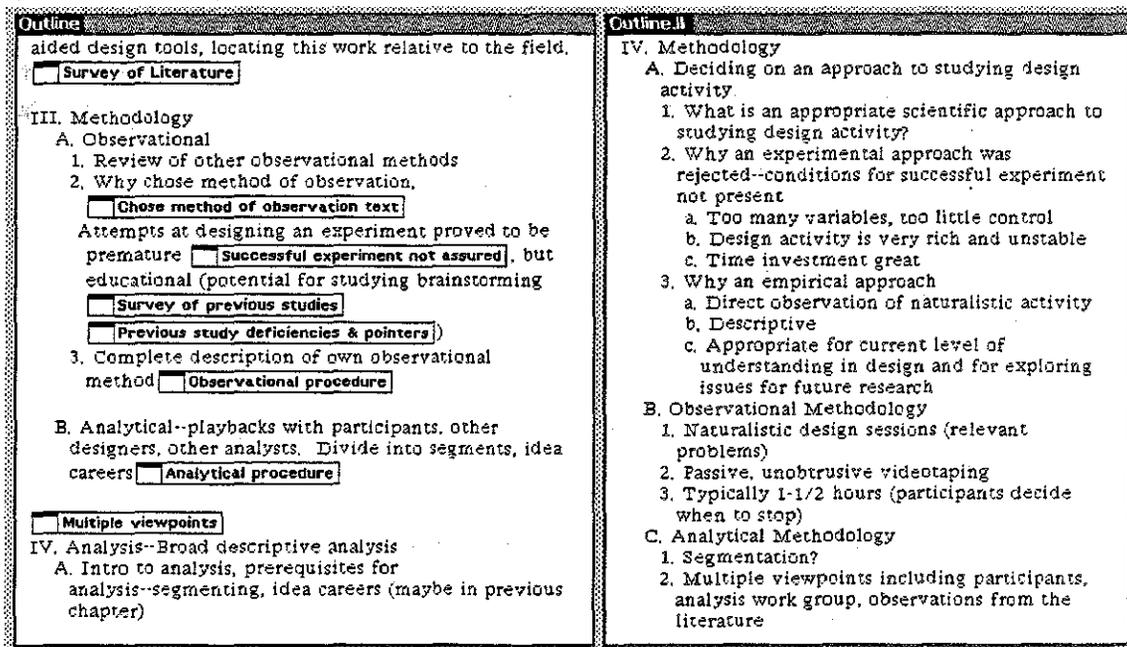


Figure 7: Evolution of an outline.

When dealing with complex structure, it is essential for the user to be able to see the overall organization. The NoteCards browser (see Section 2) was specifically designed to support such overviews. As it turns out, the browser is used by several of our writers to help

convert between organizations as well. There are three common means of keeping the contents of a browser card up to date. First, one can *recompute* the browser. This causes the current contents to be cleared and a new version to be created based on a traversal of the network. This traversal starts from the same root cards and follows the same link types used when the browser was first created. The second approach, *reconnecting* the browser, preserves both the nodes and their positions. Only the edges are redrawn to correspond to the current links in the notefile. Finally, users can manipulate the nodes and edges in the browser by hand, adding new nodes and deleting existing ones. For example, the writer responsible for the browser in Figure 2 uses it to locate and bring up cards quickly, and to organize his thoughts. He adds vertical spacing in the layout of the browser to group related nodes together. Consequently, he can't use NoteCards to recompute the browser because his modifications to the layout would be lost. Instead, he adds new nodes by hand and lets NoteCards reconnect the browser (i.e. redraw the edges).

A second example is shown in Figure 8. In this case, two collaborating writers make use of browsers to help them view the organization of their paper over time. They create a browser over the hierarchy of fileboxes and cards representing the outline of the paper. The browser allows them to reorganize the structure. For example, when creating a card, they might recognize that it should be filed in a new filebox. They sometimes know immediately where the new filebox belongs in the hierarchy, since they keep much of the paper's structure in their heads. But when this isn't possible, they bring up the browser to obtain a global view of the hierarchy.

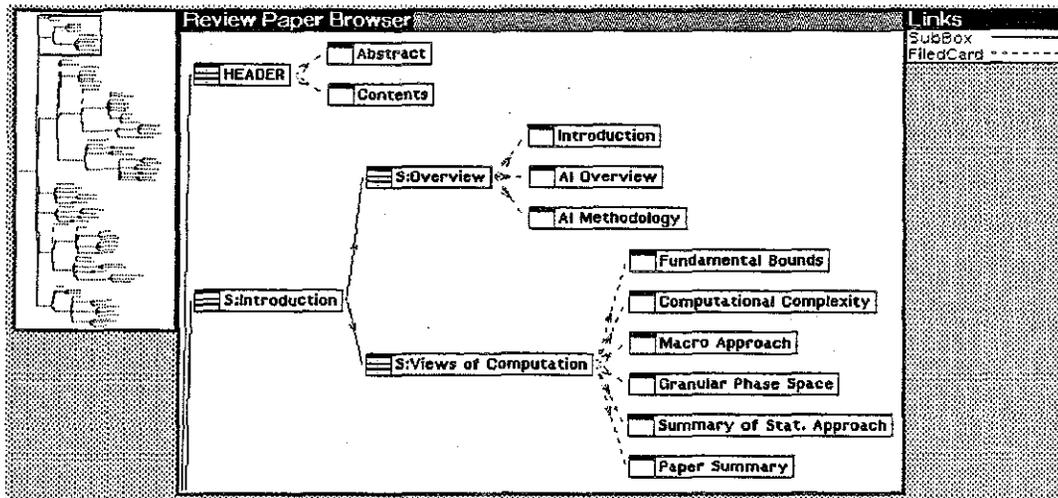


Figure 8: Browser over filebox and filed card structure.

Structure can be added to notefiles without using cards and links. For example, almost everyone adds annotations of some form to larger fileboxes, including blank lines between groups of link icons, indentation, textual commentary and headings. The filebox on the left side of Figure 9 displays such structuring annotations. Simple unlabeled grouping is done with blank lines while larger categories are identified with headings. Note also the indenting appearing near the top of the filebox to represent inclusion. This writer pointed out the ease with which such annotations can be done (and undone) without incurring the overhead of

extra layers of fileboxes to represent subgroupings. For contrast, this figure also contains one of the "bins" used by this writer for filing cards in an "unstructured" format.

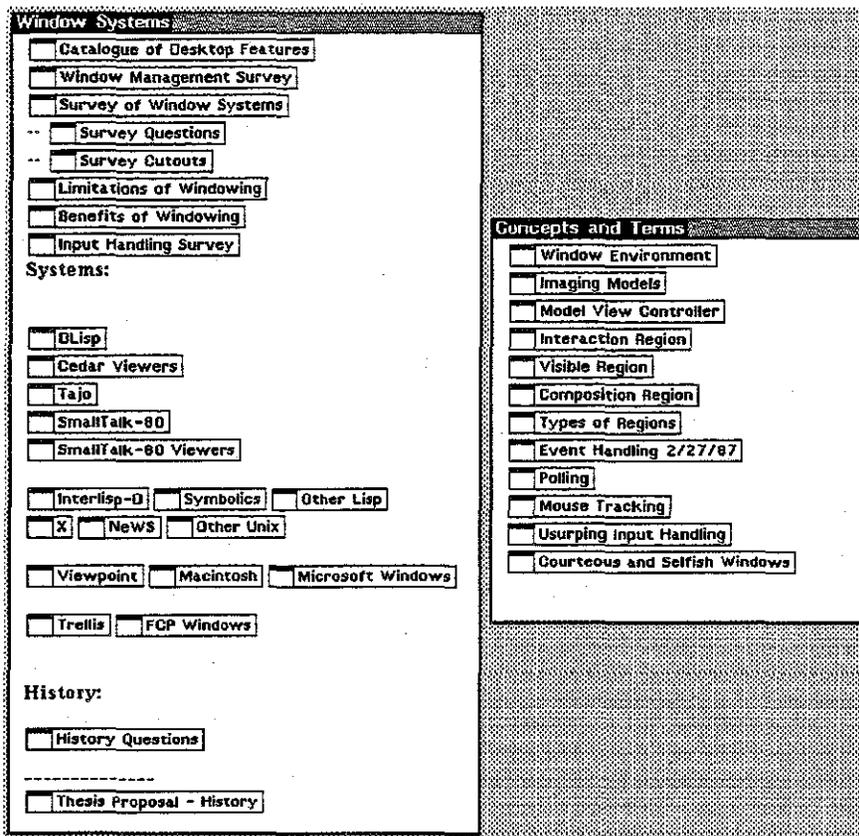


Figure 9. A "structured" filebox and a "bin."

7. DOCUMENT PREPARATION

We use the phrase "document preparation" to refer to those activities involved in the production of a linear document. These can include organizational activities like planning a linear path through some portion of the notefile as well as composing and editing text for the document. For our purposes, document preparation does not include writing activities like notetaking and reorganizing which can happen prior to and independent of considerations of a final document. As is the case for other activities described in this paper, document preparation can take place throughout the extended writing process, often commencing well before the final text is composed or assembled. Usually paper structuring or layout is done in an outline, which is massaged and fine-tuned for some time before any text is written. For some, the outline takes the form of a filebox hierarchy whose filebox titles correspond to the section titles of the paper. One writer uses a flat hierarchy consisting of cards for the sections and issues of a paper all filed in a single filebox. She arranges the cards in the order she wishes them to appear in the paper and then creates the first draft of the document by copying their contents, in order, into a new card.

If a filebox structure becomes large and complex, it is advantageous to be able to view the structure as a whole. Some writers use browsers for this purpose as shown in Figure 8.

One user demonstrated an unusual use of browsers when writing a paper. Following a period of notetaking, he created a browser over his structure of notes and taped it to a wall of his office. After looking at it for a few days (recall that the browser displays only card titles, not contents), he tore it down, threw it away, and began writing. The actual composition of the paper was done in a single linear document without further reference to the notes. In fact, it is typical of his style to rarely refer to his notes after originally composing them.

Some writers, however, create no explicit paper layout structure using cards and links, either browser or filebox-based. Rather, their outline is simply a single text card capturing the overall structure of the paper to be written. (Compare, for example, the outlines appearing in Figure 7.) Sometimes, the outline includes links to existing cards containing notes relevant to the topic of that part of the outline. Still other writers do no explicit rendering of paper layout whatsoever until the time of final document composition.

In addition to the issue of the presence or absence of an outline or other linearizing structure, our writers' notefiles vary in the degree to which the text in existing cards furnishes appropriate starting material for the paper. At one extreme are users who compose text for the document in a fresh notecard (or sometimes a fresh text file external to NoteCards). These writers often bring up relevant cards on the screen so as to have them visible while writing. They sometimes copy text from these notes, but more often they either paraphrase or simply refer to them. In a sense, this composition of the paper in a single text card signals a move out of the hypertext medium. Future work on the document including smoothing (e.g. adding transition sentences or paragraphs) and integration of reader comments happens to the linear document rather than to the source cards. Some writers take the further step of putting the document in a file card (see Section 2). This allows the document to be accessed independently of NoteCards.

For other users, the connection between the linear form and the hypertext source is maintained over time. Often, this is done using document cards, which allow users to automatically generate a linear document (in a card) covering some portion of the network. Changes to the document are made in the source cards from which the document was compiled. This allows portions of the paper to be visible in different windows and simultaneously accessible. However, this scheme can cause problems if two documents that share cards are being written in the same notefile. Here it may be difficult or impossible to make the same card function smoothly in both papers. To preclude this, some writers add an intermediate layer of cards containing sections of the paper. The document card "engine" then runs over the network of section and subsection cards to create the linear document. A writer may choose to add links from sections to relevant source notes. These can optionally be ignored by the system during computation of the document card.

Another writer, while working on his Master's thesis in NoteCards, found the functionality of document cards inadequate. Among other things, he required greater control over the traversal parameters used during their creation. This writer, being a programmer as well, used the NoteCards programmer's interface [Trig87] to create a new card type he calls *Draft* card. (The right hand side of Figure 1 in Section 2 shows part of the structure of cards over which his draft card computes in order to create a linearized document for Chapter 2 of his thesis.)

Unfortunately, a few of our users have found document cards to be insufficient for their needs. In particular, two writers collaborating on a paper needed to be able to modify the contents of a document card and have the source cards be updated automatically. For this and other reasons not relevant here, these two writers were forced to leave NoteCards altogether.⁷ In effect, their need is for the functionality of an outline processor [Enge84, Hers85] and illustrates the need for a layering of outline processor technology on top of hypertext. In such a scheme, outlines, like browsers, become one way to view a NoteCards subnetwork. These outline viewers would allow incremental inline expansion and compression of levels of the hierarchy. Changes made to the outline would automatically be passed back to the cards from which the outline was computed.

8. CONCLUSIONS AND RECOMMENDATIONS

This paper and the work it represents has proceeded under the assumption that there is no single model of the human activity of writing. Without question, there are patterns and regularities across writers and writing styles, several of which were identified here. It may indeed be the case that certain writing activities or styles can be shown to be counterproductive or that certain others might justifiably be promoted in computer-supported writing environments. (Smith et al [Smit86] have created a hypertext system for writers based more on this latter premise.) In our work, however, the approach is different. We treat the problem of providing computer support for writers as an exercise in user-centered system design. We encourage our users to "inhabit" the computer system, and then look closely at the individually distinctive environments they create in order to learn both about how the system can be improved and about the writing process itself.

This approach has important implications, some of which have already been demonstrated by the NoteCards system, while others are only now being explored. First and most important, this design philosophy requires the existence of a powerful, stable, and yet adaptable system, enabling the emergence of personal styles exhibited in the computer environment. In one sense, users of NoteCards tailor the system each time they adapt it to the particular task at hand. More explicit tailoring by both programmers and non-programmers must be supported as well [Trig87]. Furthermore, there is a need for infrastructure to encourage and support sharing the results of such tailoring among users. Toward that end, we are embarking on a project to create an online, hypertext "strategy manual" largely created by users by which new tools, conventions, strategies, and styles of use can be communicated within the NoteCards user community. Finally, there is a danger that approaches like ours can lead to sprawling unmanageable systems having little overall coherence. To preclude such an outcome, investigations like this one should be part of a larger, systematic, iterative design process.

In addition, there are several lessons learned here that may be valuable for other designers of hypertext systems intended to support writers:

Support for multiple organizations and interaction among them. We found a variety of ways in which our writers make use of multiple overlapping structures. Sometimes, these parallel organizations are meant to persist for the duration of the project. In other cases, the writers are in the process of converting from one organization to another. It is, perhaps, this

need for information sharing across multiple structures that argues best for hypertext as the implementation medium.

Multiple views of the network. Throughout our experience with hypertext, we continually confront the problem of obtaining global views of the network. Our writers found graphical overviews of structure to be useful for this purpose, as well as facilities for linearizing the network. Outline-based viewers would also be useful in NoteCards as well as a chronological browser based on the creation times of the cards it displays.

Sliding in and out of hypertext. Our writers often faced decisions as to which activities should be performed inside and outside the system as well as when to enter and leave NoteCards altogether. They also occasionally found the need to access nodes in a hypertext network from outside the hypertext system. Until totally integrated hypertext computer environments become available, our systems must continue to support such boundary crossing.

ACKNOWLEDGMENTS

We are immeasurably grateful to "our writers" who put up with extended tape-recorded interviews and follow-up questions as well as "strangers" snooping through their notefiles. And who at the same time raised our sometimes flagging spirits by doing the most ingenious things with NoteCards. Thanks also go to Lucy Suchman, Ramana Rao, Susan Newman and John Tang for discussions and comments on earlier drafts.

REFERENCES

- [Conk87] Conklin, J. , "Hypertext: An Introduction and Survey," *Computer*, September, 1987.
- [Deli86] Delisle, N., Schwartz, M., "Neptune: a hypertext system for CAD applications", *Proceedings of ACM SIGMOD '86*, pp. 132-142, Washington, D.C., 1986.
- [Enge84] Engelbart, D. C., "Authorship Provisions in Augment," *IEEE 1984 COMPCOM Proceedings*, pp. 465-472, 1984.
- [Garr86] Garrett, N. L., Smith, K. E., Meyrowitz, N., "Intermedia: Issues, Strategies, and Tactics in the Design of a Hypermedia Document System," *Proc. of Conference on Computer Supported Cooperative Work*, Austin, December, 1986
- [Hala87] Halasz, F. G., Moran, T. P., Trigg, R. H., "Notecards in a Nutshell," *Proceedings of the ACM CHI + GI Conference*, pp. 45-52, Toronto, 1987.
- [Hers85] Hershey, W., "Idea Processors," *BYTE* p. 337, June, 1985.
- [Mont86a] Monty, M. L., Moran, T. P. "A Longitudinal Study of Authoring Using NoteCards," poster presented by Monty at CHI '86 conference on Human Factors in Computing Systems, (Boston, April 13-17, 1986). Extended abstract published in *ACM SIGCHI Bulletin* 18(2), October, 1986.

- [Mont86b] Monty, M. L., "Temporal Context and Memory for Notes Stored in the Computer," poster presented at CHI '86 conference on Human Factors in Computing Systems, (Boston, April 13-17, 1986). Extended abstract published in *ACM SIGCHI Bulletin* 18(2), October, 1986.
- [Nels81] Nelson, T. H., *Literary Machines*, 1981. Available from the author (Box 128, Swarthmore, PA 19081).
- [Osbo79] Osborn, A. F., *Applied Imagination*, 3rd revised edition, New York: Charles Scribner's Sons, 1979. (First edition published in 1953.)
- [Smit86] 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.
- [Stef86] Stefik, M., Foster, G., Bobrow, D. G., Kahn, K. M., Lanning, S., Suchman, L. A., "Beyond the Chalkboard: Using Computers to Support Collaboration and Problem Solving in Meetings," *CACM* 30(1), 1986.
- [Trig83] Trigg, R., "A Network-Based Approach to Text Handling for the Online Scientific Community," Ph.D. Thesis, Dept. of Computer Science, Univ. of Maryland, 1983.
- [Trig86a] Trigg, R. & Weiser M., "TEXTNET: A Network-Based Approach to Text Handling," *ACM Transactions on Office Information Systems* 4(1), 1986.
- [Trig86b] Trigg, R. H., Suchman, L., Halasz, F. G., "Supporting Collaboration in NoteCards," *Proc. of Conference on Computer Supported Cooperative Work*, Austin, Texas, December 3-5, 1986.
- [Trig87] Trigg, R. H., Moran, T. P., Halasz, F. G., "Adaptability and Tailorability in NoteCards," *Proceedings of INTERACT '87*, Stuttgart, West Germany, 1987.
- [Yank85] Yankelovich, N., Meyrowitz, N., van Dam, A., "Reading and Writing the Electronic Book," *Computer* 18(10), 1985.

NOTES

1. Throughout the paper, we use the traditional term "hypertext", coined by Ted Nelson [Nels81]. Terms like "hypermedia" [Garr86] better capture the notion that the chunks appearing in a network can be in the form of graphics, audio, video, etc. as well as text. These, however, have yet to come into wide use. See Conk87 and Yank85 for surveys of work in this area.
2. See Mont86a for a longitudinal study of a single author using an earlier version of NoteCards.
3. In most cases this sort of access is read-only since, in general, changes to the substance of a hypertext node affect the location of its links. Note also that in the computing environment in which NoteCards runs, the one-file-per-card scheme is very inefficient due to the high overhead of file creation and access.

4. The ToBeFiled and Orphans fileboxes are maintained by the system and contain respectively, cards that haven't been filed in a filebox and cards whose last incoming link has been deleted.

5. The Colab system [Stef86] also supports such brainstorming using online graph-structured representations.

6. There are other mechanisms in NoteCards that can be used for brainstorming. Primary among these is the Sketch card and its derivatives which support line drawings and free text on a two-dimensional canvas.

7. Though there was insufficient space to discuss the particular issues raised by our collaborative writers, we firmly believe that providing support for this style of writing and for collaborative research in general is crucial. For a look at a few of the relevant issues, see Trig86.

COMPREHENDING NON-LINEAR TEXT: The Role of Discourse Cues and Reading Strategies

David Charney

Department of English
The Pennsylvania State University
University Park, PA 16802

ABSTRACT

By studying the structure of written discourse and the processes by which readers acquire information from texts, we have learned a great deal about how to design texts that facilitate learning. However, recent advances in computer technology have enabled the development of new forms of text that violate standard assumptions of what texts are like. These new forms may pose serious problems for learning because they lack discourse features that readers rely on for assimilating new information. In particular, readers traditionally rely on the writer to determine the sequence of topics and to employ conventional cues that signal relationships among topics, such as relative importance or chronology. However, on-line hypertext systems present texts non-linearly, requiring readers to decide what information to read and in what order. This paper assesses the potential impact of non-linear texts on theories of discourse and on current cognitive theories of text processing. It also describes research in progress on readers' sequencing strategies in hypertext. Research on the effect of hypertext on reading will have important practical implications for designing hypertext systems that satisfy readers' needs.

PROBLEMS POSED FOR READERS BY NON-LINEAR TEXTS

Most people think of a text as a collection of ideas that a writer has carefully organized into a coherent sequence or pattern. Discourse theorists have

identified a host of stable patterns that writers employ on every level of text, from small units such as sentences and paragraphs, up to grand schemas that outline the structure of an entire text, such as a fairy tale, a resume or a policy argument [Hall76], [Vand79], [Fahn83]. Indeed, as readers, we depend on structural patterns to help us recognize the type of text we are reading and integrate its parts. Empirical studies of reading comprehension confirm that readers understand and learn from texts more easily when the information is set out in well-defined structures and when the text provides clear signals of shifts from one part to the next [Vand83], [Kier80], [Fras70]. But apart from any natural disposition we may have to look for structure in a text, our view of text as an ordered succession of concepts is strongly reinforced by the constraints of the standard print medium: texts come to us on printed pages that we generally read in order, from the top down and from left to right.

Today, the constraints of the medium are being lifted by developments in computer technology. Instead of storing texts on-line as large, monolithic blocks, software designers are developing systems that link individual pieces of text together into complex networks. For example, the reference information for users of the text editor EMACS is stored on-line as a hierarchical network of facts that users can access in any order [Conk87]. In addition to serving a reference function, networks are also being developed to serve as sophisticated databases, as writing environments, and as instructional materials. Recent examples of such networks include Carnegie-Mellon University's ZOG system [Robe79], Xerox PARC's NoteCards [Hala87], Brown University's INTERMEDIA [Yank85], and Tektronix's Neptune [Deli86] (for a general overview, see [Conk87]). These systems give readers much greater control over the information they see and the sequence in which they see it. Instead of reading or skimming through a text from beginning to end, readers use "menus" to "order up" the pieces of text they want to read onto the computer screen. The most interesting systems link together information from a variety of media and sources so the reader can access related documents, graphics, and audio-visual displays at various points in the text. The texts become non-linear hypertexts: each reader may choose to view a different selection of pieces in a different sequence; any given reader may choose different sequences on different occasions.

Along with greater control, of course, comes a greater burden for the readers, who must now locate the information they need and relate it to other facts in the network, often without the aid of traditional structural cues. Most hypertext designers recognize the problems such networks may present, especially for readers who are unfamiliar with the concepts in the textbase. They report informal evidence that users may be overwhelmed by the choice among menu items and by the difficulties of maneuvering through the networked text structure. As a result, readers can lose track of where they are in the network (and where they have been), and often read a great deal of material that is not relevant to their purpose [Yank85], [Whit85], [Trig83]. While a fair amount of research is underway on the design and implementation of hypertext systems, very little research has investigated how readers handle such unorthodox texts and how seriously these problems interfere with normal reading processes.

ISSUES FOR RESEARCH

Given the growing interest in hypertext systems, it is important to assess the impact of non-linear texts on theories of discourse and on current cognitive theories of text processing. As in other areas of human-computer interaction, research can inform cognitive theory as well as practical system design. Three research issues appear especially important: (1) What strategies do readers adopt for sequencing pieces of hypertext (i.e., constructing a path through the network)? (2) How do such strategies influence learning (i.e., the mental representations readers build of the text)? (3) What strategies can hypertext designers employ to facilitate reading processes?

Readers' Sequencing Strategies In Hypertext

Since little systematic evidence is available about the effects of non-linear text on reading, it is important to assess the seriousness of the absence of discourse cues, especially sequencing conventions. Although we know that presenting information in a poor order can impair learning from a text [Barn84], little research has been conducted on how readers themselves choose to sequence the pieces of a text and whether reader-chosen orders are generally poor.

The most reasonable prediction is that readers new to the subject domain will have trouble sequencing the pieces of a text in groupings that reflect meaningful relationships. This prediction has not yet been tested directly, but it is supported by two kinds of evidence. First, there is empirical evidence that math students classify word problems by superficial similarities in the wording or the hypothetical problem situation rather than by deep relationships. Similar strategies have been noted for novice computer programmers [Adel81]. Second, there is anecdotal evidence that inexperienced readers of scientific articles and textbooks fail to look at figures or tables when they are referred to in the text. These results suggest that when readers are given the responsibility of selecting what text to read, they may sequence the information poorly or omit important information altogether.

Other important sequencing questions include:

Information type: do readers consistently read certain types of information (e.g., examples), while consistently skipping others (e.g., definitions, historical background)?

Prior knowlege: how does prior knowledge influence a reader's sequencing strategy? Are domain experts better than novices at deciding what to read? Determining when they have read enough?

Mental Representations of Hypertext

While it is important to know how readers construct a path through a hypertext system, the heart of the question is not what order they choose, but the effect of the chosen order on what they learn. In particular, how does the chosen sequence affect the readers' ability to grasp relationships between different pieces of text, such as relative importance, spatial or chronological ordering, cause and effect, and so on?

A large body of research suggests that readers construct hierarchical representations of the text they read [Vand83]. Readers are much more likely to remember general information at the top of the hierarchy than specific, low-level details. Crucially, the mental mechanism that Kintsch and van Dijk

[Kint78] propose to explain this "levels effect" turns on the order in which readers encounter propositions in the text and the degree to which important concepts (i.e., arguments in the propositions) are repeated in successive sentences. Readers incorporate each new proposition in the text into their internal, hierarchical textbase by creating networks with chains of repeated arguments. To the extent that the sentences in the text reuse the same arguments, the text is more coherent and the easier it is to create a mental textbase. When no explicit link to previous propositions is available in a local region of text, readers must either recall earlier propositions from memory or invent a link through inference processes. The memory advantage for high level propositions arises because these propositions are most likely to provide links throughout the text. The model assumes that links are created between propositions in working memory. Memory for a given proposition improves as the number of times it cycles through working memory increases.

The advent of hypertext raises a number of interesting issues for this approach to text processing. First, Kintsch and van Dijk's processing model [Kint78] assumes that a relatively stable textbase can be derived from a given text. This assumption seems to depend on a linear reading of a fixed amount of information. Almost by definition, however, hypertext avoids imposing a fixed order on information. Conklin [Conk87] distinguishes two types of links between the nodes in a hypertext network: organizational links (which tend to create hierarchy) and referential links (which Conklin observes are truly characteristic of hypertext). Obviously, hypertexts that fully exploit the potential of referential links raise interesting problems for the construction of a textbase. For example, will readers taking different paths through the network emerge with equivalent textbases? Second, some hypertext systems provide graphic displays of the information network. These displays may be quite important for helping readers construct a text base: by providing reminders of previous concepts and by providing an explicit guideline for an internal representation of the network. However, such a guideline may only be useful to the extent that the hypertext network itself is strictly hierarchical.

Hypertext Design: Reinventing Discourse Cues

Assuming that learners have trouble coping with the responsibility of sequencing their reading, an important research goal will be to design and evaluate methods of adapting discourse cues for non-linear texts. For example, can the listing of "menu items" serve some of the same functions as sequence cues in standard texts? Will readers follow cues such as cross-references from one piece of text to another when the referenced piece of text does not "follow" the current one?

Document designers have relied on the text processing literature to support their recommendations to writers. For example, they tell writers to organize text hierarchically (e.g., stating general information before specific information), to make presuppositions and transitions between sections explicit, to use a small set of often repeated vocabulary items and to introduce new terms only in clause predicates [Felk81], [Kier85]. While these recommendations have been found to facilitate reading, they may be quite hard to follow for writers of hypertext. How can hypertext writers interpret document design recommendations that incorporate notions of precedence or first mention?

At this point in the development of hypertext systems, it is not clear where to draw the line between system design decisions and document design decisions. For example, constraints on the types of links in the network and the "size" and content of nodes are critical for planning the segmentation and labelling of text. However, these constraints may be imposed for a particular hypertext system or they may be left up to a given writer working with a flexible hypertext system. System designers and document designers must therefore work together to define strategies for facilitating reading processes.

SEQUENCING STRATEGIES: RESEARCH IN PROGRESS

This section describes a study that addresses some of the issues raised in the previous section, specifically the issue of topic sequencing. While the study is still in its initial stages, the design is presented to illustrate one approach to collecting data for designing optimal hypertext databases and interfaces.

Participants in this experiment will study on-line information about an unfamiliar software program (viz., an electronic spreadsheet program). The text will be presented in a hypertext network (via Tektronix's Neptune system). The experiment will include a training phase and a test phase. In the training phase, subjects will study the procedures for using the spreadsheet program and their sequence of choices from the information network will be recorded. In the test phase, subjects will use the spreadsheet program to solve a series of problems that apply the studied procedures. The test is intended to reveal the effects of the sequencing choices on what subjects learn. The basic methodology for the study has been used successfully in previous work on skill learning from texts [Rede86], [Char86].

In the training phase, subjects will read about Microsoft's Multiplan program. This spreadsheet program involves a number of independent procedures that users can combine in various ways to solve problems. For example, some procedures must be executed in a fixed sequence, some can be applied at the discretion of the user, and some undo the effect of other procedures. This diversity of procedures requires subjects to acquire a deep understanding of what the procedures are for, as well as how to execute them.

Subjects

Volunteers from the PSU community (undergraduates and/or staff) will be paid a fee to participate in the study. A questionnaire developed in previous research [Char86] will be used to assess previous computer experience. This measure will be used primarily in the statistical analyses, rather than as a screening device. However, no one who has prior experience with electronic spreadsheets will be allowed to participate.

Design and Procedure

As described above, the research study will involve a training phase and a test phase. In the training phase, subjects will read the information contained in the hypertext network. Subjects will be randomly assigned to study the network in one of the following four conditions:

Pre-Planned Sequencing: subjects will be presented with a listing of menu choices and asked to rank them in the order in which they would like to read them. Subjects will subsequently read the text in the chosen order.

Opportunistic Sequencing: subjects will be presented with a listing of menu choices and allowed to select and read topics in any order. This condition should reveal the extent to which readers can use information in one piece of text as a cue to choosing the next piece.

Random Sequence: subjects will be told to read the topics in a prescribed random order (or perhaps in alphabetical order).

Guided Sequence: subjects will be told to read the topics in a prescribed logical order, based on relationships between the topics. This condition and the Random Sequence condition are intended to provide base-lines for evaluating the sequences subjects devise themselves.

This design compares reader-chosen orders to two baselines. Previous experiments on topic ordering have either enforced a random sequence or various guided sequences, but have not systematically compared the effects of these orders to those that learners choose for themselves [Barn84], [Maye76]. This design tests the prediction that the orders subjects choose will be better than random orders, but worse than a logical guided sequence.

The sequence of topics that subjects choose in the Pre-Planned and Opportunistic Sequencing conditions will be recorded and analyzed. It is likely that subjects with little computer experience will use superficial criteria to order the topics. For example, they may group together topics that use semantically related terms in the menu listings, regardless of whether or not the topics have any actual bearing on each other. In any case, the subjects' orderings will be evaluated by several metrics. For instance, the relatedness of the topics in the network will be judged by independent raters. These ratings will be used to quantify how often subjects read unrelated topics in succession.

In the testing phase, subjects will attempt to solve problems using the spreadsheet that call on the procedures they studied in the training phase. The problems, which will be similar to those employed successfully in previous research [Char86], will reveal whether subjects learned the procedures and grasped the relationships between them. For instance, subjects should be able to carry out tasks that involve fixed sequences of procedures and they should be able to choose the more appropriate of similar procedures. Subjects' performance will be evaluated in terms of the number of problems they can solve, how long it takes to solve them and the nature of the errors they make along the way. The most important analyses will compare the performance of subjects in different training conditions. It is expected that subjects who read topics in more logical sequences (whether chosen by themselves or the experimenter) will learn more and therefore perform better in the testing phase.

IMPLICATIONS FOR FUTURE RESEARCH

Non-linear text presents exciting opportunities for researchers in discourse analysis, document design and cognitive science. From the practical perspective, the need for good design is growing as rapidly as the increasing use of computers for transmitting verbal information. More importantly, however, the electronic medium is shaking our assumptions of what texts are and can be. We can now re-evaluate current theories of discourse and text processing. Some of the important long-term questions include: what it means in cognitive terms for readers to get "lost" in networked text; how the purposes readers bring to the text influence their selection strategies; and whether different strategies are needed for different types of text.

Since hypertext systems are largely still under development or are just now entering the marketplace, research on the effect of hypertext on reading will have important practical implications for designing hypertext systems that satisfy readers' needs.

REFERENCES

- [Adel81] Adelson, B. "Problem solving and the development of abstract categories in programming languages." Memory and Cognition, 9, 422-433, 1981.
- [Barn84] Barnard, P., MacLean, A., Hammond, N. "User representations of ordered sequences of command operations." Paper delivered at INTERACT '84: the First IFIP Conference on Human-Computer Interaction, 1984.
- [Char86] Charney, D., Reder, L. "Designing interactive tutorials for computer users." Human-Computer Interaction, 2, 297-317, 1986.
- [Conk87] Conklin, J. "A survey of hypertext" (MCC Technical Report STP-356-86, Rev. 1). MCC Software Technology Program, Austin, TX, February, 1987.
- [Deli86] Delisle, N., Schwartz, M. "Neptune: a hypertext system for CAD applications." SIGMOD Record, 15 (2), 132-143, June, 1986.
- [Fahn83] Fahnestock, J., Secor, M. "Teaching argument: a theory of types." College Composition and Communication, 34, 20-30, 1983.
- [Felk81] Felker, D., Pickering, F., Charrow, V., Holland, V., Redish, J. Guidelines for Document Designers. Washington, DC: American Institutes for Research, 1981.
- [Fras70] Frase, L. "The influence of sentence order and amount of higher level text processing upon reproductive and productive memory." American Educational Research Journal, 307-319, 1970.
- [Hala87] Halasz, F., Moran, R., Trigg, R. "NoteCards in a Nutshell," submitted to CHI+GI 1987, Toronto, Canada, April 5-9, 1987.

- [Hall76] Halliday, M., & Hasan, R. Cohesion in English. London: Longman, 1976.
- [Kier80] Kieras, D. "Abstracting main ideas from technical prose" (Technical Report 5). University of Arizona, Tucson, 1980.
- [Kier85] Kieras, D., Dechert, C. "Rules for comprehensible technical prose: a survey of the psycholinguistic literature" (Technical Report 21). University of Michigan, Ann Arbor, 1985.
- [Kint78] Kintsch, W., van Dijk, T. "Toward a model of text comprehension and production." Psychological Review, 85, 363-394, 1978.
- [Mayer76] Mayer, R. "Some conditions of meaningful learning for computer programming: advance organizers and subject control of frame order." Journal of Educational Psychology, 68, 143-150, 1976.
- [Rede86] Reder, L., Charney, D., Morgan, K. "The role of elaborations in learning a skill from an instructional text." Memory & Cognition, 14, 64-78, 1986.
- [Robe79] Robertson, G., McCracken, D., Newell, A. "The ZOG approach to man-machine communication" (Computer Science Technical Report CMU-CS-79-148). Carnegie-Mellon University, Pittsburgh, PA, Oct., 1979.
- [Trig83] Trigg, R. "A network-based approach to text handling for the online scientific community" (Computer Science Technical Report TR-1346). University of Maryland, November, 1983.
- [Vand79] van Dijk, T. "Relevance assignment in discourse comprehension." Discourse Processes, 2, 113-126, 1979.
- [Vand83] van Dijk, T., Kintsch, W. Strategies of discourse comprehension. New York: Academic Press, 1983.

- [Whit85] Whiteside, J., Jones, S., Levy, P., Wixon, D. "User performance with command, menu and iconic interfaces." In L. Borman & B. Curtis (Eds.), Human Factors in Computing Systems: CHI '85 Conference Proceedings. San Francisco: SIGCHI, April, 1985.
- [Yank85] Yankelovich, N., Meyrowitz, N., and van Dam, A. "Reading and writing the electronic book." Computer, 18, 15-30, October, 1985.

The Notes Program: A Hypertext Application for Writing from Source Texts

Christine Neuwirth, David Kaufer, Rick Chimera & Terilyn Gillespie

English Department
Carnegie Mellon University
Pittsburgh, PA 15213

ABSTRACT

Notes is a hypertext application developed to investigate the effects of computers on the writing process, in particular, on the processes of acquiring and structuring knowledge when writing from source texts. Notes is designed to help writers record their own ideas (e.g., reactions, inferences, plausibility assessments), recover the context for those ideas easily and view ideas from multiple perspectives. In this paper we outline the theoretical basis for the design of the Notes program. Then we briefly describe the program itself and its relation to relevant research. Finally we describe our experience with users.

INTRODUCTION

Writing an essay requires shaping a complex network of ideas, not all of which are present at the beginning of the writing process, into a coherent linear structure of sentences and paragraphs. For this linear structure to be successful, the writer must have constructed systematic conceptual groupings among ideas [Meye75]. When a writer knows a domain well, relatively simple reordering of available knowledge may be all that's necessary. However, when a domain is new to the writer, or the writer is reconceptualizing a well-known domain, the writer may need to engage in extensive reorganization and elaboration of his or her own understanding.

We have designed and implemented a computer program, called Notes, to investigate the effects of computers on the writing process, in particular, to experiment with tools to support, not replace, the decisions writers make while acquiring and structuring knowledge. Notes is one component of a larger project to develop decision support systems for reading and writing [Neuw87].

The Notes program has an analog in earlier technology: 3x5 note cards. The following section, which outlines some key components in the writing process, lays the theoretical groundwork for exploring the benefits of note cards for writers, the limitations of conventional note cards, and the expected benefits of computer-based note cards.

The Writing Process

Theories of writing processes typically identify the following activities in writing: acquiring knowledge, viewing it from different perspectives to gain new insights, structuring knowledge according to those perspectives, selecting and possibly creating knowledge to meet goals for discourse and re-arranging it so that a reader with different perspectives will find it equally coherent [Youn71]. This section explores each of these activities in some detail and comments on the use of note cards as a technology that can aid a writer in carrying out these activities.

Acquiring Knowledge

Many of the ideas that we ultimately make use of in a text come to us while acquiring knowledge, that is, while exploring a problem and finding out more about it. Typically, ideas in a new domain do not come to us in an orderly fashion. Rather, they present a puzzle of seemingly unrelated concepts and unexplained connections. It is difficult to remember specific facts that we learn. We often restructure ideas to fit patterns that are already familiar or drop ideas that are difficult to assimilate to familiar patterns [Bart32].

As we read or find out new information, we are not simply recording it. We are constructing connections, drawing inferences, imagining scenarios and examples, commenting on plausibility, noting connections to other texts and knowledge as well as connections to our immediate goal and the problem we are investigating. These elaborations play an important role in acquiring new knowledge. Researchers postulate that elaborations play two vital functions: They form connections between what people already know and the new knowledge and they build multiple retrieval paths for the ideas [Rede79]. While it is important when reading to construct elaborations and inferences, it is equally important when writing to remember that they are elaborations and inferences, and not to confuse them with the original information.

Viewing Knowledge from Different Perspectives

The second activity usually included in writing, especially by a theory that includes invention, involves viewing knowledge from different perspectives. Some inventional theories involve explicitly teaching writers a set of perspectives. For example, Aristotle's *topoi*, Young, Becker and Pike's tagmemic grid (particle, wave, field), Burke's pentad (act, scene, agency, purpose, etc.) or Nelson's system of synectics. Each of these techniques provides a system for exploring concepts, an activity essential to discovering new elaborations or relationships. Most such theories stress the importance of systematically varying perspectives, a way to overcome Burke's observation that "A way of seeing is also a way of not seeing." Indeed, studies which have examined creativity in writing have noted a direct relationship between the amount of examination of concepts from different perspectives and quality of writing and creativity [Youn73; Moor85].

Structuring Knowledge

Different perspectives also provide frameworks for structuring knowledge. Few studies have examined the process of writing while the writer is acquiring domain knowledge. Those few studies that do exist support the notion that structuring knowledge can be a significant task in writing in new domains. Newell's [Newe84] study, which examined the role of writing in learning, found that writing about a new domain required writers to move from relatively isolated and detached concepts to an integrated structure. Langer's [Lange84] study of the relationship between topic-specific knowledge and quality in expository writing suggests that the degree of organization of knowledge is directly related to a writer's success. Writers whose knowledge was highly organized, i.e., their knowledge base included superordinate concepts, precise meanings, analogies to other concepts, and explicit links among concepts, were most successful.

Selecting and Arranging

At some point in the process of writing, the writer must decide what knowledge, both acquired and original, is going to be suitable for communicating to a reader. Moreover, the writer must decide what linear order for the ideas--what juxtapositions and connections as well as oppositions--will result in best meeting the writer's goals. Exploration must, at least temporarily, come to an end.

The Benefits of Note Cards for Writers

There are many ways to "write" ideas down, to record the connections between them, to juxtapose ideas, perhaps discovering new connections: pencil and paper, 3x5 note cards, tape recorders, text-editors, etc. Some of these are better than others for aiding the processes of invention and arrangement just outlined. This section explores the benefits of note cards for carrying out some of these activities.

Writers use note cards for three primary reasons: First, note cards provide an external store for a large body of knowledge that as yet has no coherent linear structure. Second, note cards provide a convenient way for writers to record their own reactions, elaborations, and interpretations of texts while still maintaining a record of sources that the writer may want to return to or to acknowledge. Third, note cards provide a way of representing knowledge that makes some inventional activities easier.

The first benefit to writers using note cards is that they provide a convenient way to record ideas in a text. As noted above, recording concepts and propositions is particularly important when there might be a tendency to fit new knowledge to familiar but inappropriate patterns.

The second benefit to writers using note cards is that note cards give them a convenient way of recording their own reactions, elaborations, and interpretations of texts that they are reading while still maintaining a record of the source. By recording the source together with the

elaboration, note cards make both available for review and reevaluation. The importance of review and reevaluation in learning a new domain has been cited by writing researchers as a reason that writing has a major role to play in learning [Emig71].

Various studies that are relevant to taking notes have explored the strategic significance of elaborations during reading. A study of elaborations during reading in which the elaborations are written down rather than unwritten (mental or verbalized) found that written responses led to better posttest responses than unwritten [Mich61].

The third and most distinctive benefit for note cards is their power as a representational medium. A given network of ideas can be represented by a number of different structures, some of which are better than others for enabling a person to work. For example, numbers are usually better represented with Arabic than with Roman numerals. Likewise, the various structures that are encouraged by the use of note cards are better than an initial, relatively fixed, linear structure when a person needs to seek out relationships among ideas. Note cards facilitate alternative representations for the linear structuring of concepts, allowing writers to experiment with tentative arrangements until the writer discovers or can impose a workable framework.

Limitations of Conventional Note Cards

The previous section argued that note cards provide a better representational system for writers working in new domains than linear structuring: note cards facilitate a writer's exploration for alternative structures of ideas. Despite this advantage, however, conventional note cards have disadvantages. Not infrequently, writers forget the context for the original note, and must return to the source material in order to make sense of the content of the card. A similar problem occurs with paraphrasing in notes: the writer introduces inaccuracies. Writers, especially inexperienced ones, tend to spend all their time writing down quotes from the source texts rather than recording paraphrases, elaborations, inferences, interpretations, etc.

The foremost problem with note cards arises when the writer is struggling to impose a workable framework on the material: although notes offer a more tractable medium for this activity than 8x11 paper, creating alternative frameworks nevertheless destroys the previous order. Writers have two alternatives to circumvent this problem. First, they can make duplicates of note cards, a time-consuming venture. Second, they can number note cards and then record the structuring by means of the numbers. Reconstructing the ordering is then possible, but like the duplication solution, also time-consuming.

Expected Benefits of Computer-based Note Cards

When a writer is working with texts that are stored in the computer, the Notes program keeps a link between each note and the specific region in the source text from which it came. We reasoned that such a facility would free the writers (1) to paraphrase because they would

always be able to easily recover the quotation, and (2) to record their own elaborations, reactions, inferences, etc., because they could easily recover the context for them.

Recovery of context is only easily accomplished when the texts are stored in the computer. Although this is possible in a writing course in which the number of readings is small, it will be a number of years before we see vast numbers of texts stored on computers. Thus, the primary benefit of computer-based notes in the near future will be its potential for helping writers create alternative organizational frameworks more easily. Unlike paper, the computer does not collapse the storage and display of information. Because of this feature, the computer can be easily programmed to allow writers to create and view alternative organizations of their notes. Creating new alternatives does not destroy previous organizations and the computer can easily keep track of the book-keeping involved.

Design Goals

We built the Notes program to explore the ideas just outlined. In the Notes program, we use an underlying database in order to maintain links from notes to sources and from sources to notes and to allow the user to view notes from multiple perspectives.

The following list represents other design goals for the Notes program, together with their rationale.

--Ease of learning and use. Writers typically come to a program like Notes wanting to get on with a task. The program must allow them to get started with useful work immediately and must be easy for them to learn as they go along. Student writers must be able to learn the system while engaging in useful writing activities; otherwise the system will be unattractive to their teachers who will see it as taking time from the teaching of writing. In a hypertext application, ease of learning and use appears to be intimately connected to the user's ability to negotiate links among text objects without getting lost.

--Quick access to notes. The time it takes to access a note must be comparable or better than the time it takes to do so from a traditional note card file. The program must exploit the searching and retrieval power of the computer with an easy to use search interface.

--Flexibility for online and offline work. It will be some years before significant numbers of texts are online. The program must work well with off-line sources as well as online ones. Likewise, the notes must have a hard copy representation.

THE NOTES PROGRAM IN DETAIL

The Notes program consists of two basic objects, source texts and notes, and a single derived object, lists of notes. Source texts are those texts the user is reading and wants to take notes on. The source texts can be online or off, but the following discussion illustrates a user taking notes on an online text. Notes are those texts the user composes in order to record elaborations of the source texts, i.e., the user's record of his or her "writing" of the text. Notes are online.

In addition to the basic objects, the Notes program consists of a single derived object: lists of notes. In the current version of the notes program, the lists are automatically compiled by the Notes program. Lists have a linear order, alphabetically by the author of the source text and within sources, by the user-created name of each note.¹ The user can also create alternative lists, typically based on ordering principles that the Notes program cannot compute automatically. The alternative lists allows the user to impose a hierarchical structure on the notes as well.

Figure 1 illustrates how the screen might² appear to a user who is in the midst of reading on the topic of creativity. The user has taken notes on two source texts: one by Hayes and one by Perkins. The system maintains a list of all the notes a user has taken in the region labeled All Notes List. At this point and at any point, the user can select from a range of activities: view the notes, create classes and classify the notes, form alternative organizations for the notes, etc. The user controls the order of these activities. Let's suppose that the user wants to take more notes on one of the source texts, Hayes, "What is a creative act?" To do so, the user opens a set of menus and uses a mouse to select Open from a Source Text menu card.

Taking Notes

To take a note, the user selects the region in the source text where he or she wants to take a note, moves the mouse cursor anywhere in the selected region, opens a menu, and chooses Take Note from the pop-up menu (see Figure 2).

Composing a Note

When a user chooses Take Note, a note region appears below the source text. The source text itself is recentered, if necessary, so that the selected region for the note remains visible on the screen. An icon appears in the source text. The icon looks like a footnote in a square and indicates that there is a link between the source text and the note (see Figure 3).

To compose a note, the user moves the mouse cursor inside the note region, clicks the left mouse button and begins composing ("Why is it important...?"). The Notes program uses the Andrew system base editor, so the user has the full functionality of an integrated text-

editor/document-formatter to compose. In addition, the user can copy material from the source text or from other windows on the screen and paste it into the note.

Although the note region approximates a 3 X 5 card, the text of the note can be as long as the user desires. If the text that the user composes exceeds the space allocated to a note region, the entire text will not be visible. However, the user can scroll the text to view different parts of it or enlarge the Notes program window so that more text is visible.

In addition to composing the text of the note, the user must also compose a name for each note ("Why Criteria?"). A name is a mnemonic for the contents of the card, and is used by the Notes program to display a list of notes that have been composed.

To take another note, the user selects a region of text and chooses Take Note again. The previous note is replaced by a blank note and except for the name of the note, which is put into the Notes listings, the previous note is "put away" from view.

Viewing Notes

After the user has taken a number of notes, perhaps in a different session, he or she may wish to review the notes. To view notes, the user positions the mouse cursor in the All Notes List, points at a note of interest and clicks the left mouse button. The note appears in the View Notes region (see Figure 4).

The user can display up to four notes at a time. In addition, the user can ask the program to expand the viewing region so that more notes can be viewed. When the viewer has been viewing a series of notes and calls up a new note, the new note will appear in place of the note that has been dormant the longest.

Alternative Lists

In addition to viewing notes from the All Notes List or from the source text, the user can also create alternatively organized lists of notes, called alternative lists. Alternative lists support viewing notes from alternative perspectives. Users can create as many alternative arrangements as they need. They can cut and paste across different lists. In addition, they can display different alternatives on the screen and compare them.

Classifying Notes

Classes play an important role in the Notes program: classes allow users to group notes together. For example, while taking notes or after, a user may group notes according to classes that he or she creates. The classes might be related to the content or structure of the source texts, or to the nature of the elaborations that the user has composed. Figure 4 shows three classes: Original, Value, and Ability, located in the region labeled Classes at the top of the screen.

To create a class, the user displays the classes by means of a menu and chooses Add a New Class from the Edit Classes menu. There are also options to Delete a class or to Rename a class. Because deletion affects notes which might be in the specified class, the user is first informed of the number of notes which are in the class and asked to confirm or cancel the deletion. If the user responds Confirm, the class is deleted; the notes in the class are not deleted, but only removed from the class.

To add a note to an already existing class, the user makes the note the current note and clicks on its class name. The class name highlights to indicate the current note is a member of the class. Notes can be added to as many classes as the user desires.

To delete a note from a class, the user makes the note the current note, and clicks on the class name. The class box is de-highlighted to indicate that the note has been deleted from the class.

Searching

The user can search for notes on the basis of content, classes, the author of the source text, the title of the source text, the date and time the note was created, and the date and time that the note was last modified. These facilities allow users to locate notes automatically. For example, if the user had taken notes on two source texts on creativity, one by Hayes and the other by Perkins, and classified several of the notes in a user-created class of definition, the user could search for all the notes by Hayes or Perkins that are in the class definition. The search results in a listing of those notes appearing on the screen. The user can view the contents of particular cards in the search result in the same way as any list of notes.

Implementation

The Notes program runs on advanced function workstations--IBM RTs, SUN2s & 3s, and VAXstations. It runs under Andrew, a window-management and base environment for UNIX 4.2 BSD [Morr86].

RELATED RESEARCH

Text Editors

Text-editors, one of the primary user interfaces, are closely tied to computer input/output hardware: Each generation of input/output hardware (keypunches, TTYs, CRTs, and bit-mapped displays) has brought a corresponding generation of editors (batch editors, line editors, screen-oriented editors, and integrated text-editor/document formatters).

Only recently, however, has the hardware been powerful and cost-effective enough so that attention could be turned from designing software that would run efficiently on the hardware to designing software tailored especially to the editing needs of users. New systems, such as integrated text-editors/document formatters, structure-based and network-based editors, have been developed in a resulting surge of research interest [Meyr82]. The Notes program has elements in common with each of these developments. The Notes program can be viewed as an integrated text-editor/document formatter that allows the user to take full advantage of the text-editing paradigm while providing constructs that *prima facie* will facilitate parts of the writing process. This section explores the recent developments in structure-based editors and networked-based editors as they relate to Notes.

Structure-based Editors

Structure based editors are editors whose user interface and functionality exploit the structural properties of the data that are being edited. Most were developed for editing programming languages; some can edit any general data structure, including graphic structures [Fras81]; a few have been developed specifically for editing English text [Walk81; Alle81]. The widespread distribution of personal computers has brought a number of structure-based editors for English text to the general public's attention (e.g., ThinkTank).

Structure-based editors for English text usually provide two capabilities. First, they provide a diagram of the structure of the document--a hierarchical table of contents--to help readers and writers visualize the structure. Second, they provide a set of commands that exploit the structure; for example, a command to move the text cursor to the beginning of the next subsection; a command to exchange two sections; a command to show only sections and subsections, suppressing paragraph detail, etc.

The Notes program incorporates a structure-based editor for English text: Each Alternative List provides users with a structure-based editor in which they can impose structure on their notes by arranging them in a hierarchy, possibly creating new notes in the process. Unlike existing structure-based editors, which allow users to create only one hierarchical order, however, Notes allows users to create multiple hierarchies. Although users of standard hierarchical editors can approximate this capability by copying the original file and creating an alternative order in the copy, changes in the contents of the notes in the original file will not be reflected in the copy; whereas in the Notes program, changes in the contents of the notes are reflected in each alternative list, regardless of whether the alternative list was created from an already existing one.

Network-based Editors

Network-based editors are editors whose user interface and functionality allow users to build networks of data by creating links among arbitrary pieces of structure. Most were developed to experiment with non-linear organizations for data that the computer medium makes possible. Users can typically use a network-based editor to exploit the structural properties of the data, but network-based editors leave it up to the user to impose the structure; the system does not provide it or enforce it as in a structure based editor.

The concept of a network-based editor is often traced to a paper by Bush [Bush45] who proposed creating a system that would allow users to build associative links through a set of documents. Early, partial implementations of linked data include NLS/AUGMENT [Enge68; Enge73] and Hypertext [Carm69]. Later developments have taken several directions. Xanadu, an outgrowth of Hypertext, is working toward a distributed, hypertext database that could support any number of user interfaces [Nels81]. Intermedia, also an outgrowth of Hypertext, is working toward linking pieces of data objects besides text, including graphics and images [Yank85]. Textnet [Trig86] and NoteCards [Trig87] are experimenting with the effects of linked networks of data on human-computer interaction, both for individuals and for groups.

The Notes program also supports links, but of a much more restricted variety than these systems: The Notes program supports links between the source text and a note and between a bibliographic reference and a note. One way to view the Notes program is as an optimized interface for creating the links most useful for taking notes. A unit task analysis [Card83] illustrates the optimization. In a general network-based editor, taking a note and linking it to the original source text and to a bibliographical reference would require approximately 8-11 unit tasks: select a region, create a from-link, specify the type of the link as a note link, type some text of the note, select the text, create a to-link, select the text of the note again, create a from-link, specify the type of the link as a reference link, select the text of the bibliography, select a to-link. In the Notes program, taking a note requires 3 unit tasks: select a region, choose take note, type some text. The unit tasks savings comes about because Notes automatically selects the data to link to (i.e., a note) and creates two links (i.e., a note link and a reference link) in a single operation.

At least one of the general network-based editors, NoteCards, could probably be specialized to support the task-optimized linking of the Notes program. We plan to re-implement the underlying database for the Notes program so that it is based on a generalized networked database and subroutine library. A generalization of the database will allow us to make the Notes program compatible with other specialized tools for writing that we are currently implementing.

FORMATIVE EVALUATION WITH USERS

Throughout its development, we have been conducting formative evaluations of the program, where by "formative evaluation" we mean a study that attempts to evaluate a program in order to improve it. We have operated Notes in five sections of experimental writing courses for two semesters.

Participants

The participants in the evaluation have been experienced and inexperienced computer users with no prior experience with the Notes program. Some had no prior experience with Andrew, the computer system on which Notes is implemented.

Methods

Each participant comes to two sessions. The first session is a training session. In the training session, we provide a one-on-one tutorial introduction to those parts of the Andrew system that participants need in order to work with the Notes program. The training time on Andrew averages about 30 minutes. Then we give participants a hard copy tutorial introduction to the Notes program and ask them to work through the tutorial at their own pace. The average time to work through the tutorial is about 45 minutes for experienced computer users, 90 minutes for inexperienced.

In session two, we ask the participants to read two short articles on an issue (controlling human behavior), and to write an essay that (1) synthesizes the issues from the other two essays as a springboard for developing a position on the issue and (2) lays out their position on the issue. To make the task demanding, we impose a time constraint: 45 minutes to read the essays and 45 minutes to write a draft. We ask participants to use the Notes program to take notes and write their essays. In both training and work sessions, we ask participants to think-aloud as they work and we record what they say [Eric84].

So far, all participants take the full 45 minutes to read the essays and, all participants but one have taken the full 45 minutes to write the draft of the essay.

Our observations of the errors participants make and the thinking-aloud protocol data give us a wealth of information about specific problems with the program, problems that were, for the most part, relatively easy to fix. But the most valuable information about the overall design of

the program comes from interviews with participants after they have completed the reading and writing task. The interview questions, based in part on a set developed by Hidi and Klaiman [Hidi83], focus the users' attention on the process of taking notes and probe for the Notes program's effects on their usual note-taking processes (see Appendix I for a list of the interview questions).

Results of the Interview Questions

For the most part, we have incorporated solutions to participants' problems with the Notes program into the version described in this report. However, we have two outstanding problems that we urgently need to address. The first concerns the representation of notes; the second concerns support for the process of taking notes.

It is now well-established that the right representation can significantly influence the ease of problem-solving [Simo81]. Our interviews with users suggests that we would do better to provide them with not one representation for notes but a variety of representations, with each representation providing a better match to a particular sub-task. For example, some users would like to cluster their notes in a graphical network of notes before deciding on any linear order for them whatsoever. Others would like their notes to represent a path through an issue.

The second problem concerns the support for the process of taking notes. At the present time, it is easy for writers to move from notes to prose. But, not too surprisingly, writers requested the ability to move from prose to notes. Writing actual prose represents a bottom-up planning procedure. As in other complex tasks, writers engage in a combination of top-down and bottom-up planning.

We will be working to provide these additional capabilities and testing whether they provide the useful decision-support for the complex task of writing an original paper from sources.

REFERENCES

- [Alle81] Allen, E., Nix, R., & Perlis, A. PEN: A hierarchical document editor. *Proceedings of the ACM SIGPLAN/SIGOA Conference on Text Manipulation*, (Portland, Oregon, June 8-10, 1981) ACM, New York, 1981, 44-81.
- [Bart32] Bartlett, F.C. *Remembering*. Cambridge: Cambridge University Press, 1932.
- [Card83] Card, S.K., Moran, T.P., & Newell, A. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, N.J., 1983.
- [Carm69] Carmody, S., Gross, W., Nelson, T.H., Rice, D., & van Dam, A. "A hypertext editing system for the /360", in *Pertinent Concepts in Computer Graphics*, M. Faiman & J. Nievergelt, Eds. University of Illinois Press, Urbana, Ill., 1969, 291-330.

- [Enge68] Engelbart, D.C. & English, W.K. "A research center for augmenting human intellect", in *Proceedings of the Fall Joint Computer Conference*, vol. 33. AFIPS Press, Reston, Va. (1968), 395-410.
- [Enge73] Engelbart, D.C., Watson, R.W. & Norton, J.C. "The augmented knowledge workshop", in *Proceedings of the National Computer Conference*, vol. 42. AFIPS Press, Reston, Va. (1973), 9-21.
- [Eric84] Ericsson, K.A. & Simon, H.A. *Protocol analysis: Verbal reports as data*. MIT Press, 1984.
- [Fras81] Fraser, C.W. Syntax-directed editing of general data structures. *Proceedings of the ACM SIGPLAN/SIGOA Conference on Text Manipulation*, Portland, Oregon, June, 1981, 17-21.
- [Emig77] Emig, J. "Writing as a mode of learning," *College Composition and Communication*, 28, 1977, 122-128.
- [Hidi83] Hidi, S. & Klaiman, R. "Notetaking by experts and novices: An attempt to identify teachable strategies," *Curriculum Inquiry*, 13, 4 (1983), 377-395.
- [Lang84] Langer, J.A. "The effects of available information on responses to school writing tasks," *Research in the Teaching of English*, 18, 1 (Feb. 1984), 27-44.
- [Meyr75] Meyer, B.J.F. *The organization of prose and its effects on memory*. Amsterdam, The Netherlands: North-Holland, 1975.
- [Meyr82] Meyrowitz, N. & van Dam, A. "Interactive editing systems," *ACM Computing Surveys*, 14, 3 (Sept. 1982), 321-415.
- [Mich61] Michael, D.N., & Maccoby, N. "Factors influencing the effects of student participation on verbal learning from films," in A.A. Lumsdaine (Ed.), *Student response in programmed instruction*. Washington, DC: National Academy of Sciences, 1961.
- [Moor85] Moore, M. T. "The relationship between the originality of essays and variables in the problem-discovery process: A study of creative and non-creative middle school students," *Research in the Teaching in English*, 19, 1 (Feb. 1985), 84-95.
- [Morr86] Morris, J.H. et al., "Andrew: A distributed personal computing environment," *Communications of the ACM*, 29, 3 (March, 1986), 184-201.
- [Nels81] Nelson, T. *Literary Machines*. T. Nelson, P.O. Box 118, Swarthmore, PA 19081, 1981.

- [Neuw87] Neuwirth, C.M., & Kaufer, D. Decision support systems for reading and writing. *Technical Report CMU-CECE-TR-3*, Pittsburgh: Carnegie-Mellon University, Center for Educational Computing in English, English Department, 1987.
- [Newe84] Newell, G.E. "Learning from writing in two content areas: A case study of protocol analysis.," *Research in the Teaching of English*, 18, 3 (Oct. 1984), 265-287.
- [Rede79] Reder, L.M. "The role of elaborations in memory for prose," *Cognitive Psychology*, 1979, 11, 221-234.
- [Trig86] Trigg, R. & Weiser, M. "TEXTNET: A network-based approach to text handling," *ACM Transactions on Office Information Systems*, 4, 1 (Jan., 1986), 1-23.
- [Trig87] Trigg, R., Suchman, L., & Halasz, F. "Supporting collaboration in NoteCards," In *Proceedings of the Conference on Computer-Supported Cooperative Work* (Austin, Tex., DEC.). ACM, New York. 1987.
- [Walk81] Walker, J. "The document editor: A support environment for preparing technical documents," *Proceedings of the ACM SIGPLAN/SIGOA Conference on Text Manipulation*, Portland, Oregon, June, 1981, 44-50.
- [Yank85] Yankelovich, N., Meyrowitz, N. & van Dam, A. "Reading and writing the electronic book," *Computer*, 18, 10 (Oct., 1985), 15-30.
- [Youn71] Young, R. E., Becker, A., Pike, K. *Rhetoric: Discovery & Change*. Harcourt, Brace & Yovanovich, San Francisco, 1971.
- [Youn73] Young, R. E., & Koen, F.M. The tagmemic discovery procedure: An evaluation of its uses in the teaching of rhetoric. *Final Research Report to the National Endowment for the Humanities*, Grant number EO-5238-71-116, University of Michigan, 1973.

ACKNOWLEDGEMENTS

We thank the many people involved with the development of the Notes program, especially the teachers and students who have used the program and given us valuable feedback. The work described in this report represents contributions by the following people:

System design: Chris Neuwirth, Rick Chimera, David Kaufer

System programming: Rick Chimera, Gary Keim, Dale Miller, Keith Evans, Aaron Oppenheimer

User testing: Chris Neuwirth, Terilyn Gillespie

Documentation: Terilyn Gillespie, Tom Gomoll

System design consultant: Thom Peters

The design, development and testing of the Notes program has been supported, in part, by the Fund for the Improvement of Post-Secondary Education (FIPSE) as part of the Warrant Project, Preston Covey, Cheryl Geisler, David Kaufer and Chris Neuwirth, Principal Investigators. We would also like to thank Carnegie Mellon's Center for the Development of Educational Computing for programming support, Carnegie Mellon's Information Technology Center for system support and IBM for an equipment grant. Finally, thanks to Davida Charney, Erwin Steinberg and Richard Young for their comments on earlier drafts.

NOTES

¹A planned extension will allow users to see note cards by other program-generated orders, such as grouped by classes that the user has put the notes in.

²Because the system that Notes is implemented on allows multiple windows, the entire screen may not be devoted to the Notes program. If a user has more than one window on the screen, Notes occupies the portion of the screen that the user has allocated for it, but the Notes window itself would still appear as described. In addition, the Notes window itself can take on different appearances. For example, the user can hide various regions of the Notes program's window from view and expose regions to view. For example, if the user is primarily engaged in taking notes, he or she may not want the listing and viewing regions exposed to view and there is an option to Hide/Expose the All Notes List and Viewing regions.

³From Hayes, J. R., [1982] *The Complete Problem Solver*. The Franklin Institute Press, Philadelphia, pp. 197-8.

APPENDIX I: INTERVIEW QUESTIONS

1. Do you usually take notes?
2. When you do take notes, do you take them as you just did or was this particular session unusual? If unusual, how?
3. When you do take notes do you have a particular style, format or procedure that you use? Does this depend on the reason you are taking notes? Did the Notes program interfere with or enhance your style, format or procedure?
4. Do you usually have some specific ideas in mind before you begin to take notes or does the text suggest ideas to you? Did this process feel any different when working with the Notes program?
5. Do you usually reword or select particular ideas to take notes on, or do you often copy parts of the text verbatim? Does this depend on your purpose? The material? Did your rewording or copying practices change or stay the same as a result using the Notes program?
6. How do you select the ideas you take notes on? Did taking notes with the Notes program affect your selection of ideas?
7. What features did you like about the Notes program?
8. What features did you dislike?
9. What would you change? What would you change it to?
10. If you had to name just one additional feature that the Notes program should have, what would it be?
11. Would you use the Notes program again if you had the opportunity? Why or why not?

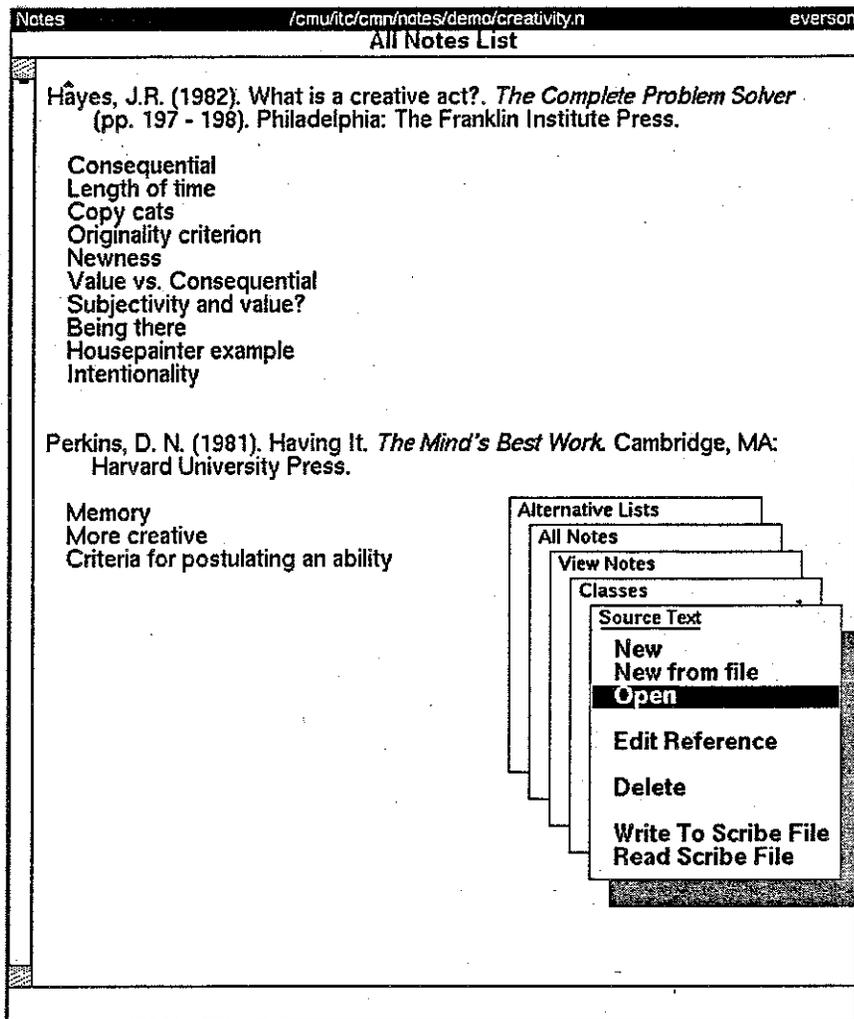


Figure 1. The Notes Program Window

Notes	/cmu/itc/cmn/notes/demo/creativity.n	everson
Source Text : /cmu/itc/cmn/notes/demo/hayes.d		All Notes List
<p>Creative acts come in a great variety of forms. A creative act may be quite ordinary and inconsequential--for example, it might be something as simple as making up a bedtime story to tell our children--or it may be world shaking--as was Galileo's invention of the science of physics. A creative act may involve years of concentrated work--consider the decades Darwin devoted to developing the evidence for the theory of evolution--or it may be brief--condensed into a sudden flash of insight--the sort of insight that drove Archimedes naked from his bath shouting, "Eureka!"</p>	<p>Hayes, J.R. (1982). What is a creative act?. <i>The Complete Problem Solver</i> (pp. 197 - 198). Philadelphia: The Franklin Institute Press.</p>	
<p>What is there about these very different acts that leads us to call them all 'creative'? Typically, we apply fairly stringent criteria in judging creativity. In most cases, we require an act to pass three tests before we call it creative. First, we must believe that the act is original. Second, we must believe that it is valuable. And third, it must suggest to us that the person who performed the act has special mental abilities. For example, when we see what the person has done, we ask ourselves, "How did she ever think of that?" or, "How did he have the patience to work all that out?"</p>	<p>Consequential Length of time Copy cats Originality criterion Newness Value vs. Consequent ial Subjectivity and value? Being there Housepainter example Intentionality</p>	
<p>Let's examine these conditions in or</p>	<p>Take Note</p>	
<p>Originality</p>	<p>Copy</p>	
<p>We certainly wouldn't judge a painter creative who simply copied the pictures of other painters. To be judged creative, painters must use their own resources to shape the painting. They must paint their <i>own</i> pictures.</p>	<p>Perkins, D. N. (1981). <i>Having It. The Mind's Best Work</i>. Cambridge, MA: Harvard University Press.</p>	
<p>We don't mean though that everything in a creative work must be original. Painters, writers, and inventors routinely use ideas borrowed from</p>		

Figure 2. Taking a Note

Notes	/cmu/itc/cmn/notes/demo/creativity.n	everson
Source Text : /cmu/itc/cmn/notes/demo/hayes.d		All Notes List
<p>Creative acts come in a great variety of forms. A creative act may be quite ordinary and inconsequential--for example, it might be something as simple as making up a bedtime story to tell our children--or it may be world shaking--as was Galileo's invention of the science of physics. A creative act may involve years of concentrated work--consider the decades Darwin devoted to developing the evidence for the theory of evolution--or it may be brief--condensed into a sudden flash of insight--the sort of insight that drove Archimedes naked from his bath shouting, "Eureka!"</p>	<p>Hayes, J.R. (1982). What is a creative act?. <i>The Complete Problem Solver</i> (pp. 197 - 198). Philadelphia: The Franklin Institute Press.</p>	
<p>What is there about these very different acts that leads us to call them all 'creative'? Typically, we apply fairly stringent criteria in judging creativity. In most cases, we require an act to pass three tests before we call it creative. First, we must believe that the act is original. Second, we must believe that it is valuable. And third, it must suggest to us that the person who performed the act has special mental abilities. For example, when we see what the person has done, we ask ourselves, "How did she ever think of that?" or, "How did he have the patience to work all that out?"</p>	<p>Consequential Length of time Copy cats Originality criterion Newness Value vs. Consequent ial Subjectivity and value? Being there Housepainter example Intentionality Why criteria?</p>	
<p>Why criteria?</p>		
<p>Why is it important to have criteria?</p>	<p>Perkins, D. N. (1981). <i>Having It. The Mind's Best Work</i>. Cambridge, MA: Harvard University Press.</p>	

Figure 3. Composing a Note

Notes		/cmu/itc/cmn/notes/demo/creativity.n		everson
Classes		View Notes		
		Consequential	Copy cats	
<p>Original Value Abilities a new class</p>		<p>Well, maybe. But it seems that the creative acts that are important are consequential. Otherwise, why all the fuss about creativity?</p>	<p>Copy cats not creative. But someone can be "inspired" or "influenced" by another's work and still be creative. Where do we draw the line?</p> <p>Maybe borderline case aren't important.</p>	
<p>All Notes List</p>				
<p>Hayes, J.R. (1982). What is a creative act? <i>The Complete Problem Solver</i> (pp. 197 - 198). Philadelphia: The Franklin Institute Press.</p> <p><i>Consequential</i> <i>Length of time</i> <i>Copy cats</i> <i>Originality criterion</i> Newness Value vs. Consequential Subjectivity and value? Being there</p>		<p>Length of time</p> <p>OK. But presumably Archimedes was working on the problem for some time before the "Eureka!"</p> <p>Chance favors the well-prepared mind.</p>	<p>Originality criterion</p> <p>Isn't originality just another word for creativity? What is the concept going to explain about creativity? publicity subject</p>	

Figure 4. Viewing and Classifying Notes

Notes		everson
Alternative Lists		All Notes List
Abilities	draft1	
Housepainter example	Subjectivity and value?	Hayes, J.R. (1982). What is a creative act?. <i>The Complete Problem Solver</i> (pp. 197 - 198). Philadelphia: The Franklin Institute Press. Consequential Length of time Copy cats Originality Newness Value vs. Consequential Subjectivity and value? Being there Both Hayes and Perkins discuss criteria for creativity. Whereas Hayes' seems to be driving at ways to distinguish creative acts from ones that are not creative, Perkins' criteria seem designed to distinguish a specifically creative ability from onther, perhaps more general mental abilities. (1) The ability in itself should make a
Being there Analogy	Value vs. Consequential	
Intentionality Hayes idea	Length of time	
Abilities mix vs. stuff	Copy cats	
Really a mix?	Originality criterion	
Talent or knack	Newness	
Memory	Value vs. Consequential	
More creative	Subjectivity and value?	
Creative i s better?	Being there	
Criteria for postulating an ability		
Perkins		Perkins, D. N. (1981). <i>Having It. The Mind's Best Work</i> . Cambridge, MA: Harvard University Press.

Figure 5. Alternative Lists

Translating Text into Hypertext

Hypertext and the New Oxford English Dictionary

Darrell R. Raymond and Frank Wm. Tompa†

Centre for the New Oxford English Dictionary
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

ABSTRACT

An alternative to manual composition of hypertext databases is conversion from existing texts. Such conversion often requires careful analysis of the text document in order to determine how best to represent its structure. We illustrate some of the issues of conversion with an analysis of the Oxford English Dictionary.

INTRODUCTION

CD ROM and other low-cost high-capacity storage devices are making widespread computerized access to existing reference works a reality. Examples of documents currently available on CD ROM include Bowker's *Books In Print*, Microsoft's *Bookshelf* (which contains ten works including *Roget's Thesaurus*, *Bartlett's Familiar Quotations*, the *World Almanac*, and the *U.S. Zip Code Directory*), the *McGraw-Hill Concise Encyclopedia of Science and Technology*, and the *Thesaurus Linguae Graecae*. The basic assets of computerization — speed of access and quantity of storage — make almost any electronic form of these documents useful. However, computerization also presents new opportunities in advanced representations for such documents, both for storage and for display. Hypertext is one currently underexplored area of representation [Mich87].

Producing a good hypertextual representation of an existing document can be difficult. The document may have been created decades or centuries ago, and so its authors or editors cannot be consulted about their intentions. As a result, structure must be inferred from a careful study of the text. Further, the characteristics of pen and paper or the printing press often exert a powerful influence on a document, and this influence must be isolated from the purely formal or logical constraints of its structure. Once the characteristics of the original document have been determined, the hypertext designer attempts to represent these accurately within the restrictions of the computer medium. A final problem in conversion is that we lack a rigorous characterization of hypertext. Some writers emphasize speed of access or non-linearity of the medium [Conk86], others emphasize selectivity [Jone87]; text conversion focuses attention on the discreteness of the hypertext fragments and the explicitness of hypertext links. Fuzzy definitions of hypertext make it difficult to evaluate the acceptability of a given conversion.

† On leave at Bellcore, Morristown, N.J. during 1987-1988

This paper reports our experiences in evaluating the potential for hypertextual representations of the *Oxford English Dictionary*. With the completion of the *Supplement* to the *OED*, Oxford University Press has turned to consideration of an electronic form for the dictionary, referred to as the *New OED* [Wein87]. The text of the *OED* has been completely keyboarded, and we are currently designing suitable computer-based tools for storage, editing, and searching. Our work on the *OED* has provided some insight into the problems of converting existing documents to hypertext.

THE OXFORD ENGLISH DICTIONARY

The *Oxford English Dictionary* [Murr28] is the largest and most scholarly dictionary of written English. Its production spanned the period from 1884 to 1928, with nearly thirty years of preliminary effort in planning and collecting material. In its standard form the *OED* consists of twelve books containing 42.51 million words in 252,259 entries, and 1.86 million quotations. An important subsequent work is the four-volume *Supplement* that contains entries for words which were not covered in the *OED*, and also contains additional material and emendations to existing entries. The *Supplement* was produced from 1958 to 1986.

Abbréviate (äbr'vi,et), *v.*, also 5-7 *abbréviate*.
 [I. **ABBREVIATE** *ppl. a.*; or on the analogy of *vbs.* so formed; see **-ATE**. A direct representative of L. *abbreviäre*; as **ABBIDOE**, and the obs. **ARREVT**, represent it indirectly, through OFr. *abregier* and mid. Fr. *abrévier*. Like the latter, *abbréviate*, was often spelt *a-bréviate* in 5-7.] To make shorter, shorten, cut short in any way.
 1530 PALSGR., *Abrevyate*: I make a thyngeshorte, *Ye abrevge*.
 1625 BACON *Essays* xxiv. 99 (1862) But it is one Thing to Abbreviate by Contracting, Another by Cutting off.
 † 1. *trans.* To make a discourse shorter by omitting details and preserving the substance; to abridge, condense. *Obs.*
 c. 1450 *Chester Pl.* I. 2 (Sh. Soc.) This matter he abbreviated into playes twenty-four. 1592 GREENE *Comy catching* iii. 16 The queane abreviated her discourse. 1637 RALPH *Mahomet* 34 Abbreviated out of two Arabique writers translated into Spanish. 1675 MANLEY *Interpreter* pref. I have omitted several Matters . . . contracted and abbreviated Others.
 † b. To make an abstract or brief of, to epitomize. *Obs.*
 c. 1450 *TREvisa Higden's Polychr.* I. 22 (Rolls Ser.) Trogus Pompeius, in hys 21^d iiii. booke, allemoste of alle the storyes of the worlde, whom Iustinus his disciple did abbreviate.
 1603 FLORIO *Montaigne* (1634) 627 To reade, to note, and to abbreviate Polibius. 1648-9 *The Kingdome Weekly Intelligencer* Jan. 16 to 23 The high court of Justice did this day sit again concerning the triall of the King. The charge was brought in and abbreviated.
 † c. *Math.* To reduce (a fraction) to lower terms. *Obs.*
 1796 *Mathem. Dict.* I. 2 To abbreviate fractions in arithmetic and algebra, is to lessen proportionally their terms, or the numerator and denominator.
 † 2. *intr.* To speak or write briefly, to be brief. *Obs.*

Figure 1. Abbreviate.

The basic unit of the *OED* is the entry; each entry details the historical development of a given word. A typical *OED* entry is that for *Abbreviate*, part of which is seen in Figure 1. An entry consists of the main form, (the word being defined), pronunciation, part-of-speech, a list of variant forms, etymology, and a set of senses. Each sense contains a definition and typically some illustrative quotations chosen from different sources and time periods. Senses often have nested subsenses in the case of commonly used words, prefixes, or suffixes.

While some of the structural elements in an entry are indicated by special symbols (e.g., etymologies are surrounded by “[” and “]”), most are indicated only implicitly, by position and font or a combination of both. It was necessary to keyboard the text of the *OED* manually so that

tags could be added to the data to designate the implicit structure. These tags were designed for ease of data entry rather than completeness of structure representation. As a result, extensive work was conducted to build parsers which could complete the tagging and verify the data. [Kazm86] The tagged and parsed text for Abbreviate is shown in Figure 2, up to and including the first two quotations. These tags are currently being exploited in the development of a database index.

```
<entry> <hwgp> <hwlem>abbreviate </hwlem> <pron id=0000041884>a&breve.br
<i>i&mac.</i> &sd.vi&syllab.<i>e</i> <su>i</su> t </pron>, <pos>v.</pos> </hwgp>
<vfl> Also <vd>5&en.7</vd> <vf>abreviate</vf>. </vfl> <etym> f.<xra
id=0000041880><xlem>abbreviate</xlem> <pos>ppl. a.</pos> </xra>; or on the
analogy of vbs. so formed; see <xra id=0000041881> <xlem>-ate</xlem> </xra>.
&es.A direct representative of L. <cf>abbrevia&mac.re</cf>; as <xra
id=0000041882><xlem>abridge</xlem> </xra>, and the obs. <xra
id=0000041883><xlem>abrevy</xlem> </xra>, represent it indirectly, through OFr.
<cf>abregier</cf> and mid.Fr. <cf>abre&acu.vier</cf>. &es.Like the latter,
<cf>abbreviate </cf>, was often spelt <cf>a-breviate</cf> in 5&en.7. </etym>
<sen4> <sen6> To make shorter, shorten, cut short in any way. <qpara> <quot>
<qdat>1530</qdat> <auth>Palsgr.</auth>, <qtxt>I abrevyate: I make a thyngge
shorte, <i>Je abrege</i>. </qtxt> </quot> <quot> <qdat>1625</qdat>
<auth>Bacon</auth> <wk>Essays</wk> xxiv. 99 (1862) <qtxt>But it is one Thing to
Abbreviate by Contracting, Another by Cutting off.</qtxt> </quot> </qpara>
</sen6> </sen4>
```

Figure 2. Tagged data for Abbreviate.

CONVERTING TEXT TO HYPERTEXT

Much attention has been focused on attractive aspects of hypertext such as multiple windows and multiple access paths to text fragments. Contemplating the conversion of the *OED* to hypertext has led us to consider other issues, in particular the problems involved in defining nodes and links. A document can be broken into many networks of arbitrarily chosen pages or nodes, but not all of these are suitable representations of the whole content of the original text. The choice of nodes, links, and network of interconnection each have a significant impact on the nature of the whole hypertext.

From the point of view of document conversion, what distinguishes hypertext is that its text fragments and links are discrete and explicit. Hypertext nodes are text fragments which have the special characteristic of being both semantically and syntactically discrete. Ideally, each node represents a single, independent concept which is susceptible to classification in a large number of ways. However, not all concepts are representable in this fashion. Some previous hypertext work seems to suggest that any text would be better off as hypertext. We suggest that if the document's themes are too closely interwoven then syntactic fragmentation must lose semantic information.

Similarly, hypertextual links are also discrete and explicit, as can be observed from the requirements that they need labels and that users actively select them. We suspect that the richness of some documents is directly related to the implicit nature of their links, since the source and sink of an implicit link varies with interpretation and hence is more flexible than an explicit link. Finally, decomposition of the document into pages and links necessarily involves some consideration of the network and its associated substructures. If this organization is not sufficiently

representative of the theme, argument, or structure of the original document, then information will again be lost.

These issues suggest that for some purposes and documents no hypertext representation will be adequate. Hypertext makes an implicit structure explicit, so the key question in conversion must be *will explicit structure be as expressive as implicit structure?* When the answer is yes, the document will gain from conversion; otherwise, conversion will degrade the representation of the document.

In the case of the *OED*, the bulk of the text is quotations which have been extracted from other texts. For these it is clear that the fragments exhibit the necessary discreteness, and so a hypertext representation should provide an appropriate delivery vehicle. The subsections of the entries adhere to a formal structure since we can describe them with a context-free grammar. However, our discussions with Oxford editors [Raym86] have led to the observation that an entry is not merely a collection of independent parts. Entries are stylistic, creative wholes in which the relative sizes and arrangement of the parts often conveys important (though implicit) information about the development of a word. We have yet to determine if this implicit structure can be explicitly represented in a useful form.

WHY HYPERTEXT FOR THE NEW OED?

The main reason for considering a hypertext representation of the New *OED* is to support browsing. The *OED* can be treated as a text database to which formal queries are posed, e.g. *What interjections were in common use in the period 1670-1720?* [Gonn87a][Raym87a] However, experience with an extremely rapid searching program called PAT [Gonn87b] has shown that browsing the dictionary is an invaluable adjunct to formal querying, and is often more fruitful, serendipitous, and enjoyable. Browsing is a two-stage process: users specify a pattern to be located and then navigate in the vicinity of the pattern. Our users typically employ PAT to search for entries containing interesting quotes, phrases, or words, and then navigate the original paper dictionary for the context of the quotes. This switch of medium is necessary because PAT is not knowledgeable about the structure of the dictionary, and so does not support navigational browsing. An obvious approach is to supply a hypertext-like browsing facility as a front end to PAT.

A second reason for hypertextual representation is to improve the existing representation, in particular to remove the constraints imposed for the original medium. The *OED* was originally contracted for a specific number of pages, which the editors soon determined would not be sufficient. Nevertheless, they employed every means at their disposal to make the *OED* fit within this page restriction, short of reducing its quality or comprehensiveness. The need to conserve "real estate" led to very dense typesetting and the extensive use of abbreviations and symbols; very few spatial techniques could be employed to give the structure of an entry visual saliency. While the result is admirable, we need not observe this constraint any longer. A hypertextual representation should provide better visual saliency and more rapid navigation around large entries. A key mechanism to be employed is dynamic reformatting of entries according to user specification.

A third reason for hypertextual representation is integration of the *OED* with the user's task. Our users typically query and browse the *OED* as part of a more extended task. At a minimum, users want to save their results and queries for use in a new session, but they also expect access to annotation facilities, the ability to cut and paste fragments of *OED* text into other documents, routines for sorting and filtering extracted quotations, and tools for statistical analysis of selected variables. The most important observation here is that hypertext can facilitate a consistent and simple interface to a wide range of tools.

STRUCTURAL CHARACTERISTICS OF THE OED

A few minutes of browsing the *OED* reveals its most important characteristic — the variance in size and structure of its entries. Entries in the *OED* range from “Gig: see JIG” to the entry for *set v.*, which is almost half a megabyte. Some entries have hundreds of quotations, but many have none. An entry may contain a well-balanced arrangement of senses and quotations, or it may consist largely of possible compounds or formations e.g., the entry for *un-*. Some entries can be perceived very quickly, while comprehension of even the structural skeleton of long entries is extremely difficult. Clearly, attempting to find a representation for such a wide range will not be a simple task.

Merely knowing the size of the largest and smallest entries is insufficient to develop a representation. Figure 3 shows the distribution of the sizes of the entries in the *OED*, where the size of each entry has been rounded off to the nearest ten characters.

The most striking observation is that some 49,000 entries (one-fifth of the total) contain fewer than 50 characters. The majority of these are entries for obsolete variants of words and cross-references to other entries. 95% of all entries are smaller than 4000 characters, however 21% of all entries are larger than 1000 characters. It would appear that large entries are not the most common, and hence the problem of representation may be less severe than it first appeared. However, the probability of access is not equal for every entry. The size of an entry is dependent on the number of suitable quotations found for the word; thus it should not be surprising that a large number of the smaller entries are obsolete or infrequently used words, while commonly used words have larger and more complicated entries. If the commonly used words are also more likely to be browsed, the need for a good representation increases. An argument could be made that the larger entries are more likely to be browsed whether they are common or not, since their structure and content are more difficult to perceive and remember. It should also be noted that the size of entries will generally increase when the material in the *Supplement* is integrated with that in the *OED*.

The display of larger entries may be facilitated by structural views or abbreviations such as those described Furnas [Furn86]. Structural information can be extracted from the tags, and hence they can be employed in construction of a structural view. Figure 4 shows the relationship between number of tags and size of entry for *J*. Clearly, the larger the entry, the more tags it contains.

However, many tags are only typographical, or otherwise designate flat structure which does not facilitate abbreviated display. A more illuminating measure would compare the number of sense tags to entry size. Each sense tag denotes a level in the logical hierarchy, so this

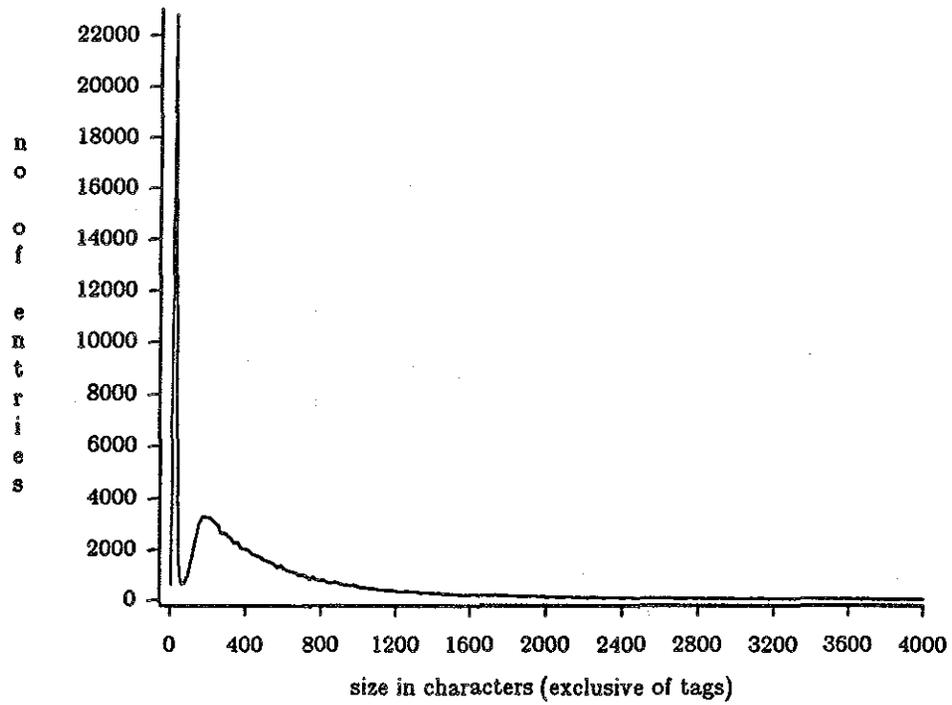


Figure 3. Distribution of entry size.

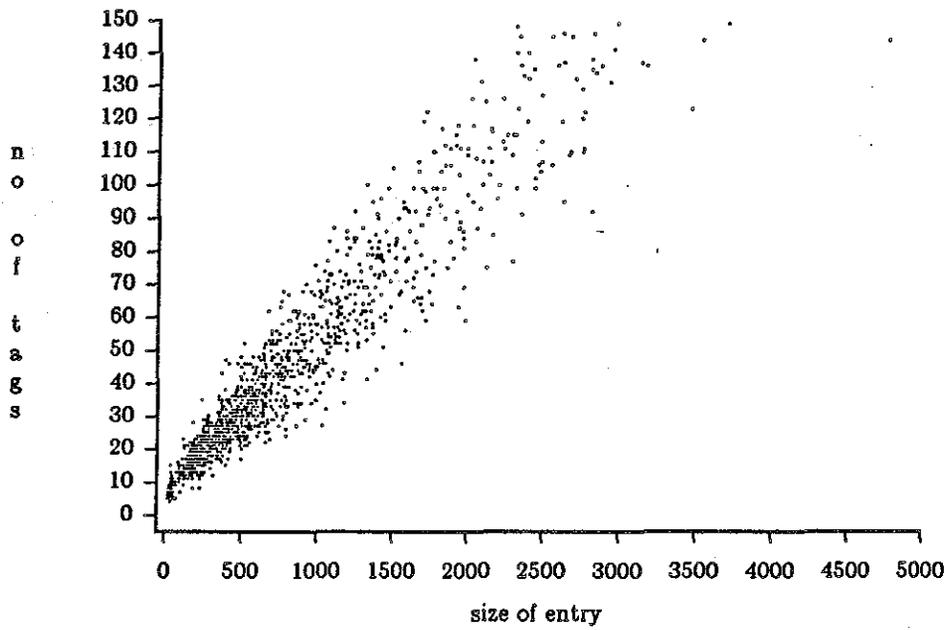


Figure 4. Distribution of tags vs. size of entry.

measure roughly indicates the correlation between size and quantity of structural information. This data is shown in Figure 5. Clearly the number of sense tags is not well correlated with entry size. We have observed that some entries have a large number of quotations or long etymological notes but a very shallow sense structure; a tag-derived skeletal view of such an entry is not indicative of its size or comprehensiveness. On the other hand, some entries have a deeper hierarchy with relatively little leaf content; in this case a tag-derived skeletal view overemphasizes the content of the entry.

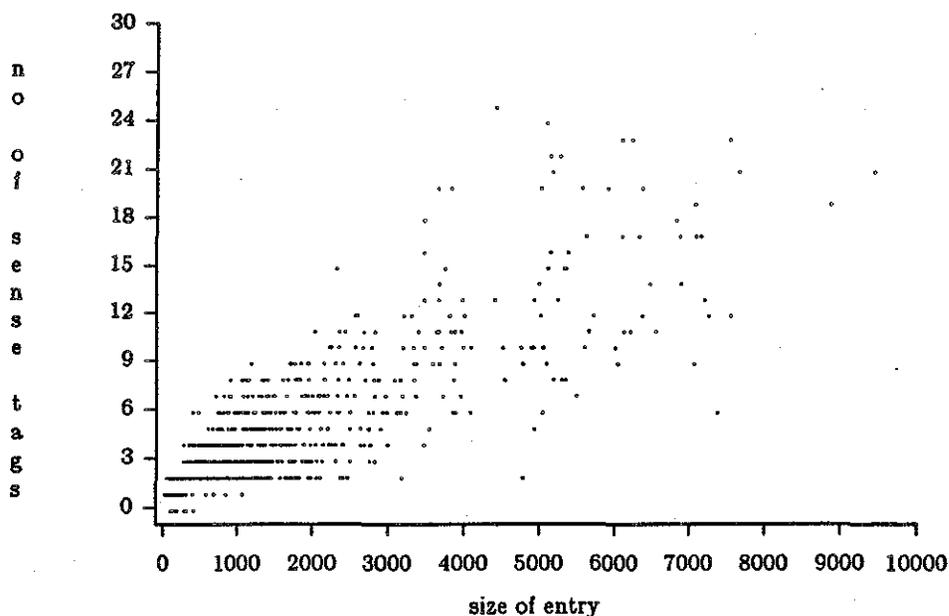


Figure 5. Distribution of sense tags vs. size of entry.

We are currently evaluating several representation prototypes, running on a Sun 3/50 under the X window system. Details of these prototypes will be presented at the workshop.

Forming an *OED* hypertext also requires identification of links. First we must ascertain which links are already present in the data or tags. The most explicit links are the cross-references, which are pointers to other entries. These can appear in etymological notes or sense text, and are visually designated by printing the main form of the cross-referenced entry in small capitals. In Figure 1 there are cross-references to *-ATE*, *ABRIDGE*, and *ABREVVY*. The *OED* contains 475,000 cross-references for an average of 1.88 per entry, making these a substantial source of hypertextual links.

However, a large number of cross-references does not necessarily imply an equitable distribution of cross-reference destinations. Figure 6 and 7 show the distribution of cross-reference destinations from *J* and *A* to the letters of the alphabet. The last bar of each graph represents the number of cross-references to suffixes (i.e., words beginning with "-"). It is apparent that most cross-references are to words in the same letter of the alphabet, with the only major exception being cross-references to suffixes.

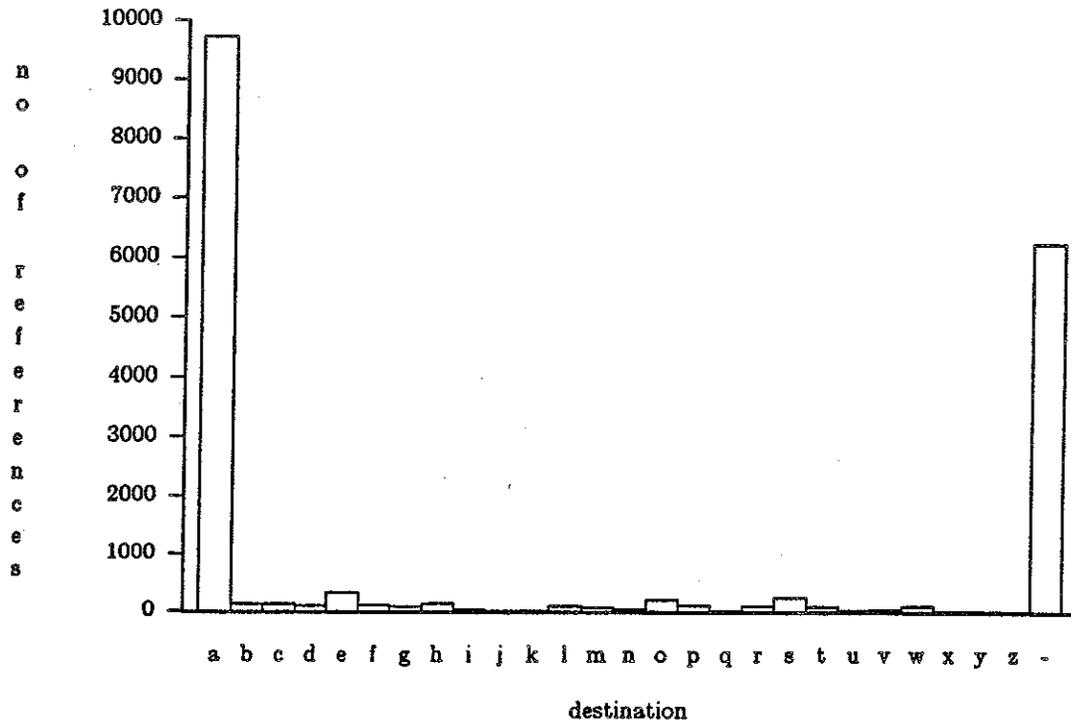


Figure 6. Distribution of cross-reference destinations for A.

This distribution is indicative of several factors. First, cross-references in the *OED* generally point to words with similar spelling. Many cross-references are prefaced with *erroneous spelling of*, *variant of*, *obs. form of*, *verbal form of* and other such phrases which typically indicate a word with very similar spelling. The major exception to this rule is suffixes, which are distributed more evenly about the alphabet. The second factor is the history of the development of the dictionary. Volumes were generally compiled in alphabetical order, and so the compilers of the later volumes naturally had more information on which to base cross-references, and would be more likely to cross-reference existing entries than as-yet uncompiled entries. We observe from Figure 7 that J cross-references earlier volumes more than later ones. A third factor is psychological; since an editor would be most familiar with the section of the *OED* currently in progress, it is more likely that he or she would insert cross-references to entries in that section.

A more general type of link is known as a *lexicographical* link. These links result from the simple observation that since the *OED* is a book which defines most English words, every word in the *OED* can be seen as the source of a implicit link that points to its definition. The *OED* and other comprehensive dictionaries are thus unique in containing to some extent their own reference material. Lexicographical links are simple and natural for the casual user, and following them about the dictionary is merely a special case of the general ability to point to any text and find its entry in the *OED*. However, determining the source and destination of these links is not simple. The word being used as the source may need to be reduced in tense or number to its morphological root. Then the *OED* must be checked to see if the root is a main form; that is, if there is an entry corresponding to the root. It is possible for the root to be the main form of

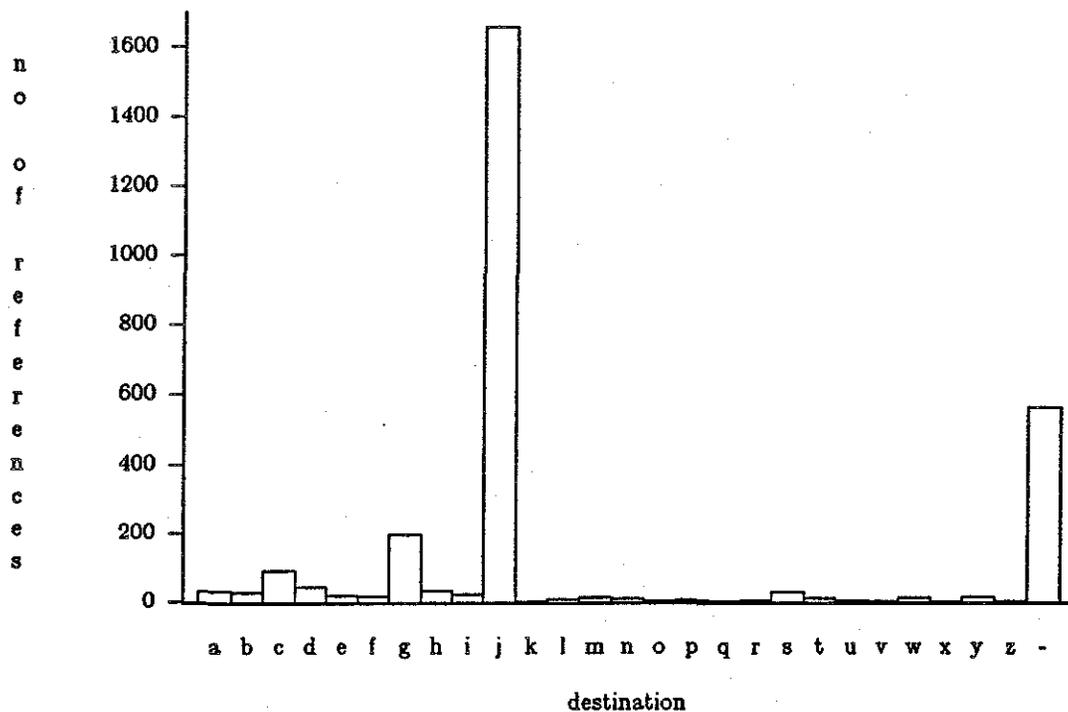


Figure 7. Distribution of cross-reference destinations for J.

several entries, in which case either the part of speech must be extracted from the surrounding text, or the set of entries is returned to the user for manual evaluation.

If the root is not a main form, there are still three possibilities: the root (or the original word) is a bold lemma; the root (or the original word) is an italic lemma; the root (or the original word) is a variant form. Bold and italic lemmas are used to indicate compounds and other derived forms which do not require separate entries but which may have their own supporting quotations. We note that the supporting quotations for bold and italic lemmas are not structurally identified in the current tagging, and must be extracted from the quotation list by observing the occurrence of the bold or italic lemma in the quote. Variant forms are spelling variations; each entry has a set of variants that is divided according to the century in which the variant is acceptable.

The existing cross-reference links in the *OED* are important, but limited in their scope. Furthermore, they can be treated as a special case of the more general lexicographical links, which present several difficulties. Supporting these links will require sophisticated access software.

RELATED WORK

In addition to developing hypertextual representations of the *OED*, we have identified two other benefits of considering the *OED* in the context of hypertext.

The first area is the use of the *OED* as a generator of hypertextual links for other documents. For example, co-citation links could be derived for every two texts that supply a quotation to the *OED* that illustrates a given sense of a word. Useful links within and between other texts can be identified from collocations in the *OED*; for example, a large overlap among words in definitions might serve to identify similarity of topic. More refined techniques to extract meaningful pairings remains a difficult research problem in computational linguistics.

More importantly, the issue of developing and editing entries for the *OED* increases the need for powerful editors for hypertext systems. A significant problem in editing is the creation and maintenance of the sense structure [Raym86]. The source material for each entry is a large set of quotations obtained over the years from volunteer and directed readers, each handwritten on a 6" by 4" slip of paper. In a method still practised today, the editors distill the senses of the word from the set of quotations by arranging and re-arranging the slips into spatial categories on a desktop, looking for the pattern of historical development. In a computerized environment, one's first guess is that an appropriate system might employ a desktop metaphor as does a system like NoteCards [Hala87]. However, existing systems seem to be designed for tens of slips at a time, whereas *OED* editors can be faced with organizing thousands of slips. A more fundamental problem has been identified in experiments we have conducted on organization of proverbs [Raym87b]. These experiments indicate that there is a quantifiable decrease in the quality of semantic categorization when a categorization task is performed in an electronic environment employing a spatial metaphor. It appears that the ability to create temporary, unnamed categories is a key factor in the development of good semantic structures, and that current systems and metaphors interfere significantly with this ability.

CONCLUSIONS

Our experience with the *OED* illustrates some of the issues involved in converting existing texts to hypertext. Conversion of complex texts requires a careful analysis of the document and an understanding of its history and nature. The analysis of the *OED* led us to the observation that a key characteristic of hypertext is the discrete nature of its components. We considered display of large entries in the *OED* and learned that a simple structural abbreviation would not be satisfactory in all cases. The investigation of explicit links in the *OED* showed significant amounts of local interconnectivity, but relatively few links between sections of the database that had been compiled at different times. We suspect that this localization of links to be a general tendency in large documents and possibly even in hypertexts, simply because of the cost and difficulty of continually integrating old material with new. Concurrent work on the *OED* and hypertexts also highlighted two further directions for hypertext research: improving data content by using reference materials to identify related subject matter, and improving hypertext editors to address the problems of organizing vast amounts of textual data.

ACKNOWLEDGEMENTS

We gratefully acknowledge the financial assistance received from the Natural Sciences and Engineering Research Council of Canada through the University-Industry Program under grant CRD-862.

REFERENCES

- [Conk86] Conklin, J., "A Survey of Hypertext", MCC Tech. Report STP-356-86, October 23, 1986
- [Furn86] Furnas, G.W., "Generalized Fisheye Views", *Proceedings of the CHI '86 Conference on Human Factors in Computing Systems*, April 13-17, 1986, pp. 16-23
- [Gonn87a] Gonnet, G.H., Tompa, F.W., "Mind Your Grammar: A New Approach to Modelling Text", *VLDB '87*, Brighton, England, September 1987, pp. 339-346
- [Gonn87b] Gonnet, G.H., "Examples of PAT", OED-87-02, UW Centre for the New Oxford English Dictionary, Waterloo, Ontario, August 1987
- [Hala87] Halasz, F.G., Moran, T.P., Trigg, R.H., "NoteCards in a Nutshell", *Proceedings of the CHI + GI Conference on Human Factors in Computer Systems and Graphics Interface*, April 5-9, 1987, pp. 45-52
- [Jone87] Jones, W.P., "How Do We Distinguish the Hyper From the Hype in Non-Linear Text?", *INTERACT '87*, Stuttgart, September 1-4, 1987, pp. 1107-1113
- [Kazm86] Kazman, R., "Structuring the Text of the Oxford English Dictionary Through Finite State Transduction", CS-86-20, Department of Computer Science, University of Waterloo, Waterloo, Ontario, June 1986
- [Mich87] Michel, S., "Guide — A Hypertext Solution", *CD ROM Review*, pp. 22-24, July/August 1987
- [Murr28] Murray, J.A.H., *The Oxford English Dictionary*, Oxford at the Clarendon Press, Oxford, England, 1928
- [Raym87a] Raymond, D.R., Blake, E.G., "Solving Queries in a Grammar-Defined OED", unpublished technical report, UW Centre for the New Oxford English Dictionary, Waterloo, Ontario, February 1987
- [Raym87b] Raymond, D.R., Cañas, A.J., Tompa, F.W., Safayeni, F.R., "Measuring the Effectiveness of Personal Database Structures", *International Journal of Man-Machine Studies* (in press)
- [Raym86] Raymond, D.R., Warburton Y., "Computerization of Lexicographical Activity on the New Oxford English Dictionary", OED-87-03, UW Centre for the New Oxford English Dictionary, Waterloo, Ontario, December 10, 1986
- [Wein87] Weiner, E., "The Electronic English Dictionary", *Oxford Magazine*, February 1987, pp. 6-9

Content Oriented Relations between Text Units - a Structural Model for Hypertexts

Rainer Hammwöhner and Ulrich Thiel

University of Constance, Dept. of Information Science
Project TWRM-TOPOGRAPHIC
Postfach 5560, D-7750 Konstanz, F.R.G.

I. ABSTRACT

A common feature of various recently developed information systems is the decomposition of linear document structures which are enforced by conventional print media. Instead, a network organization of information units of different forms (textual, graphical, pictorial and even auditive presentation modes may be combined) is provided. Documents organized this way are called "hypertexts". However, two questions arise immediately when an effort is made to build information systems on the basis of this conception:

- What are the "units" constituting a hypertext?*
- What sort of links between the units will be provided?*

Most approaches to hypertext systems impose the task of deciding these questions on the authors of hypertexts, thus the systems are hypertext management devices (eg CHRISTODOULAKIS ET AL. 86, WOELK ET AL. 86). The approach taken in this paper leaves a more active role to the software by applying knowledge based techniques.

The starting point is the automatic content analysis of machine-readable full-text documents which may be downloaded from a full-text data base. The analysis process results in a partitioning of the document into thematically coherent text passages, which are one kind of node of the hypertextual version of this document. Other nodes contain graphics, tables and summarizations. The content analysis is accomplished by a semantic parser, which has access to an explicit model of the discourse domain. The TOPIC-System (HAHN/REIMER 86) comprises prototypical implementations of these components. Due to the semantic modeling relations between the nodes may be formally defined in order to provide content oriented browsing facilities. The graphical retrieval system TOPOGRAPHIC (THIEL/HAMMWÖHNER 87) employs an already implemented subset of them to guide users to relevant text parts.

In this paper we outline a structure model for hypertexts based on partial representations of the meaning of text parts. Formal definitions of content oriented relations between such text units are given in terms of a logic specification language.

II. TEXT UNITS: RESULT OF A CONTENT ORIENTED FRAGMENTATION OF DOCUMENTS

In this section, we will first present a basic model of general hypertext systems that allow users to browse in a set of "text units" (they may be presented in a multi-media environment). In order to access relevant units, the user must be guided along content oriented links connecting units that have a semantic overlap.

A concept oriented framework for modeling the semantics of text units is outlined in the second part of this section, serving as a foundation for the definition of relations between the concepts modeled. These relationships may be used in two ways: First, they provide information for the semantic parsing that is needed to obtain representations of the units' contents. (We briefly outline some problems of the parsing process that can be solved by the defined relations.) In the next chapter, the construction of relations between text units will then be based on the same relations.

Content analysis of text units provides not only a powerful browsing facility, but also the opportunity to propose an augmented hypertext model. This model features "derived text units", which may be added to the original ones. These new text units may be regarded as summarizations of the contents of text fragments. They provide an overview over larger units, and surplus give rise to a special type of browsing. This navigation operator leading from the condensed abstract to detailed original information is called "informational zooming". Parts of the abstract may be weighted, thus the zooming is interest-driven.

II.1 Basic Hypertext Features: Browsing in a Set of Text Units

Most hypertext systems employ graphical user interfaces, which are part of object oriented programming environments. Windowing being a usual feature, the common technique of assigning a window (or icon as a shrunken form) to each text unit allows to choose appropriate presentation methods for each type of text unit.

The surface structure of the given original document and the media used to communicate can be employed to introduce the following classification of text units:

- 1) The units conveying the simplest semantic structures are formatted graphical entities like tables. Their surface structure corresponds directly to their contents.
- 2) Textual units consist of sentences, therefore their surface structure is linear. The semantics of a text unit can only be partially modeled due to its complexity.
- 3) Graphical units are assemblies of graphic primitives, whereas pictures and icons are sets of pixels. The analysis of pictures requires a dedicated methodology, which will not be a topic of this paper, but the (partial) modeling of pictorial semantics can be accomplished within the formal framework outlined in the following.

As speech fragments, animated graphics, software modules, and videos may also be nodes in hypertext networks, the classification above does not claim to be a complete one. However, the types of text units included in the subset can be regarded as the most important ones for our approach to construct hypertext versions from machine-readable documents.

In order to access text units that are not depicted on the screen the user is usually given the opportunity to browse in the hypertext network. There are two kinds of browsing (cf BATES 86):

- a) undirected browsing: The user investigates text units in an arbitrary ordering. This type of navigation may provide a survey of what can be accessed in general, but if the number of items available increases the user will ask for a more dedicated access to text units which are relevant for him.
- b) directed browsing: This way of navigation requires that the text units are interconnected by meaningful links. Thus, a selection of the appropriate link may be accomplished, which results either in the replacement of the text unit currently displayed on the screen, or in the presentation of one or more additional text units.

In this paper, we are concerned with the latter. In most hypertext systems, the selection of text units to be accessed is based on keywords or descriptors (eg WEYER 82, WEYER/BORNING 85) which are either assigned to the text units by the hypertext author or are detected in the units by string matching procedures. However, if the network of text units is to be constructed automatically, formally specified relations which refer to partial representations of the contents of the interconnected units are needed to provide content oriented browsing facilities. The semantic modeling may be restricted to topical descriptions, because in most situations it suffices to know what a text unit is about instead of having a detailed account of its contents. Therefore, the text analysis may be accomplished by selective parsing procedures that capture the meaning of nominal phrases. As far as full texts are concerned, a method which allows the determination of the "aboutness" of a text in a reasonable time is required that is applicable in situations where larger collections of texts have to be processed. (Similar restrictions of image analysis may aim at identification of the objects depicted in a picture, while neglecting deeper analysis like action detection and scene interpretation.)

II.2 Capturing the Contents of Text Units: Knowledge Representation and Semantic Parsing

In the following, we specify the properties of a knowledge representation formalism which is powerful enough to capture the contents of tables and to support a semantic parsing yielding topical descriptions of text passages on an indicative level. There is evidence for the appropriateness of similar approaches to image and speech analysis, but this hypothesis will not be discussed to further extent in this paper.

We start with the category of text units whose semantics is intuitively understood in terms of simple relationships: tables. Added to textual parts of a document, they often serve to summarize the main facts or to communicate sets of formatted data records. The semantics is constituted by the aggregation of columns or rows as a table, thus providing a framework for entering the individual items into the right place. Each column (row) is associated with its set of entries, as well as the table is constituted by its columns or rows. This obviously may be modeled by a "frame" (cf MINSKY 75) by identifying the columns (rows) with "slots", the data items contained in a column being its "slot entries".

Here is a concise verbalization of the frame construct from which first-order predicates are obtained. They may be regarded as abbreviations of the informally specified structural conditions for frames, slots, and entries. (Although it is possible to give a complete axiomatization (cf HAYES 79), for the purposes of this paper it is sufficient to adopt a descriptive view on frames treating them as an abstract data type (cf HAYES/HENDRIX 81). Thus, implementational issues may be left outside, while the discussion concentrates on properties of (and relations between) objects that are presumed to match the frame specification.)

A frame consists of a name and a set of slots. This requirement may be formalized using the basic predicate

$\text{is-frame}(f)$

asserting that f is a frame, the function

$\text{fn}(f)$

yielding the name of the frame f , and for each slot s the condition

$\text{is-slot}(f,s)$

stating the assignment of s to the frame f . A slot has a name given by the function

$\text{sn}(s)$

and a (potentially empty) set of entries such that for each entry the proposition

$\text{is-entry}(f,s,e)$

holds. Slot entries may either be unstructured individuals or may be frames, having a slot set of their own. The latter possibility allows a modeling of aspects (slots) of a frame by nesting the representation structures. In the following example, a frame called "Zenon-X" represents a hypothetical micro-computer which is characterized by the features "Manufacturer", "Vendor", and "CPU". A fictive corporation is assigned to the "Manufacturer"-slot: "Zeta-Machines". Thus, the meaning of the table below is captured completely.

Zenon-X	Manufacturer	Vendor	CPU
	Zeta-Machines		

The same frame may also be taken as a partial semantic representation of a sentence fragment like:

... the Zenon-X which was recently developed by Zeta-Machines...

However, for the purpose of text analysis procedures which yield such a frame representation of a given sentence, it is necessary to restrict the possible slot filling operations by means of integrity rules in order to model linguistic regularities (cf REIMER 86). A first step in this direction is the introduction of "singleton slots" which stand for properties that can only have one value at a time. The singleton slots of the frame f fulfill the condition

$is-single(f,s).$

Further integrity rules may require that an item must be a member of a specified set of allowed entries, if it is to be assigned to the slot as a value during the parsing process. (We give no further formalization of the notion of integrity rules here, because they are primarily important for processes that change the knowledge base, i.e. editing domain-specific knowledge and parsing the documents, and have been discussed in REIMER 86, and REIMER/HAHN 85 where a frame representation model (FRM) is presented which captures the semantics of concepts. As FRM is used as the knowledge representation formalism in the text analysis system TOPIC, the prototypical hypertext system TOPOGRAPHIC accessing the results of TOPIC may therefore assume the knowledge structures to be consistent.)

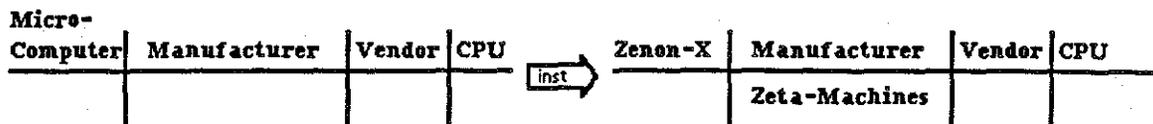
In order to capture the conceptual contents of a given text correctly the text analysis mechanism of TOPIC has to perform two main tasks: anaphora resolution and (restricted) concept learning. The solution to these problems is primarily based on providing two different kinds of frames: "A prototype frame acts as a representative of a concept class consisting of instance frames which all have the same slots but differ from the prototype in that they are further characterized by slot entries. Thus, instance frames stand for individual concepts of a domain of discourse" (HAHN/REIMER 86). The notion of prototypes can be formalized in the following definition:

Def. 1:

$\forall f: is-prototype(f) \leftrightarrow is-frame(f) \wedge \neg \exists s: [is-slot(f,s) \wedge \exists e: is-entry(f,s,e)]$

There is a canonical relation associating each instance frame to its corresponding prototype: the *inst*-relation, which relates frames with similar slot structures.

Example: inst



The formal definition below is a generalized version of the *inst*-predicate, holding not only between a prototype and its corresponding instance frames, but also between two instances if the second frame is a specialization of the first one:

Def. 2:

- (1) $\forall f_1, \forall f_2 : \text{inst}(f_1, f_2) \leftrightarrow \text{is-frame}(f_1) \quad \wedge \text{is-frame}(f_2) \wedge$
 (2) $\forall s_1 \exists s_2 : [\text{is-slot}(f_1, s_1) \Rightarrow \text{is-slot}(f_2, s_2) \wedge \text{sn}(s_1) = \text{sn}(s_2)] \wedge$
 (3) $\forall s_2 \exists s_1 : [\text{is-slot}(f_2, s_2) \Rightarrow \text{is-slot}(f_1, s_1) \wedge \text{sn}(s_1) = \text{sn}(s_2)] \wedge$
 (4) $\exists s_2 \exists e : [\text{is-entry}(f_2, s_2, e) \wedge \neg \exists s_1 : [\text{sn}(s_1) = \text{sn}(s_2) \wedge \text{is-entry}(f_1, s_1, e)]]]$

This can be employed in a simple but often sufficient heuristic of concept learning: If an unknown noun occurs during the parsing process and there is an indicator of what concept class it may belong to (eg if it is a compound noun containing a prototype identifier), then it can be integrated into the knowledge base as a frame inheriting the slots of its supposed prototype. The slots may then be filled with further information from the text.

In the process of anaphora resolution the inst-relation is used for identifying the instance frame that occurred in the previous text part, if a prototype frame is encountered (and there is linguistic evidence that it is used anaphorically). This method can be extended to other prototypes which are generalizations of the instance's prototype. In this case, the is-a-relation holds between the prototypes. (Note that the above descriptions of concept learning and anaphora resolution are idealized to emphasize the very ideas. More technical specifications give HAHN/REIMER 86).

The formal definition of the is-a-relation is recursive due to the fact that slots may be frames themselves. To cover this case we use the extended is-a-relation e-is-a, which is the transitive hull of the union of the inst- and is-a-relations.

Def. 3:

- (1) $\forall f_1, \forall f_2 : \text{is-a}(f_1, f_2) \leftrightarrow \text{is-prototype}(f_1) \quad \wedge \text{is-prototype}(f_2) \wedge$
 (2) $\exists s : [\text{is-slot}(f_2, s) \wedge \neg \text{is-slot}(f_1, s)] \wedge$
 (3) $\forall s : [\text{is-slot}(f_1, s) \Rightarrow \text{is-slot}(f_2, s)] \vee$
 (4) $\exists s' \exists f' \exists f'' : [\text{is-slot}(f_2, s') \wedge \text{fn}(f) = \text{sn}(s) \wedge \text{fn}(f') = \text{sn}(s') \wedge \text{e-is-a}(f, f'')]$

Def. 4:

- (1) $\forall f_1, \forall f_2 : \text{e-is-a}(f_1, f_2) \leftrightarrow \text{is-a}(f_1, f_2) \vee \text{inst}(f_1, f_2) \vee$
 (2) $\exists f' : [\text{is-a}(f_1, f') \wedge \text{inst}(f', f_2)]$

In the following example, the is-a-relation holds between "Micro-Computer" and "Graphics-Workstation", because of the additional slot "Graphics-Screen".

Example: is-a



The knowledge representation mechanism of TOPIC/TOPOGRAPHIC combines the modeling of concepts as frames with the modeling of certain relationships between frames, a technique originally devised in the area of 'semantic networks'. Furthermore, the relations are defined mathematically exploiting the structural properties of the frames involved. Due to the concise definitions, the concept hierarchy of the knowledge base is system-controlled, i.e. each new frame entered into the knowledge base will be classified automatically by computing all the relational links that connect it to modeled concepts (cf REIMER 86). The system incorporates a variety of other relations (eg parts), which support the semantic parsing procedure. Thus, the parser - organized as a lexically distributed grammar in the format of word experts (cf HAHN 86) - is not only enabled to assemble factual knowledge by recognizing concepts, filling slots or classifying sub-concepts but also to detect topical shifts by combining syntagmatic indicators (start of a new paragraph, occurrence of idiomatic phrases that indicate a new topical focus) with semantical criteria (eg if the current sentence has no semantic overlap with the previous ones.)

II.3 An Augmented Hypertext Model

Whereas the world knowledge base of TOPIC/TOPOGRAPHIC contains a taxonomic model of the discourse domain, the text knowledge base consists of "text graphs" which represent the knowledge obtained by the parsing process. Each analyzed text is thus stored not only in textual form (i.e. the original text units), but also associated with its topical and, to some extent, factual content, which is organized as a conceptual graph. The following information about the analyzed text can be found in the text knowledge resulting from the analysis and the subsequent condensation process:

- a) A multi-hierarchical graph whose nodes contain the topical structures of the text in decreasing generality. The contents of these nodes are similar to world knowledge structures.
- b) Fragments of world knowledge denoting the main topics of the text passages, i.e. the frames that match the most salient concepts in thematically coherent text parts. The frames are connected by relational links, thus a network representing the topical structure of the text unit is given.
- c) The frames occurring in the networks may have 'filled' slots, i.e. there may be entries assigned to them during the process of text analysis. The filling of slots contributes to the factual information from the text by adding more precise details to the general information provided by the frames and their slots.

The semantic representations of text units offer the opportunity to support the user with interaction techniques which complement the access to original text units via browse operations. The basis for this augmentation of the dialog facilities are artificial (or derived) text units:

- The natural language presentation of text unit contents does not necessarily depend on the original text fragments. Text generation procedures that cast the knowledge structures into predefined templates are currently under development. Thus, it will be possible to provide

abstracts of the text units.

- A graphical presentation of the semantic structures - as tables or networks of nodes representing concepts - provides an automatic "text mapping" (a technique of drawing conceptual graphs in order to memorize the contents of texts (cf DANSERAU/HOLLEY 82) which enhances remembering performance). This facility is featured by our prototypical information system TOPOGRAPHIC.

The presentation of artificial text units as "condensates" of the original ones entails the possibility of switching between different layers of specificity which may be assigned to the given text units (THIEL/HAMMWÖHNER 87 provide a more detailed discussion of the layered organization of text units). Especially the access to more detailed text units is supported in TOPOGRAPHIC by a general operator: informational zooming. As in optics zooming reveals more details of physical objects, the 'zoom' option in TOPOGRAPHIC can be used to access more detailed informational structures, or, in other words, to switch to a layer below. This is facilitated by navigating along the semantic relation that holds between an abstract (or conceptual network) and the text unit it has been derived from during the text analysis. (The next chapter provides an overview of the semantic relations that may additionally be used for navigation in a hypertext graph.) Thus, it is easy for the user to access an original text unit whose corresponding abstract or conceptual graph are relevant. The expansion of simple objects, usually the nodes (i.e. frames) of a network given, also fits into this model. Zooming alone, however, does not suffice for a goal oriented dialog, because there may be too much detail information on the layer below. Therefore, a sort of focussing is needed. This is accomplished by the 'select' option which allows to mark those features of a given layer, which are to be shown in detail by the zoom operator. Selecting a topical profile from a knowledge base representing the taxonomic structure of the discourse domain to which the analyzed documents pertain induces a relevance weighting on the text units. This will be employed to define pragmatic relations between text units in the next chapter.

III. HYPERTEXTUAL RELATIONS

As we have outlined in the previous chapter intertextual relations are of crucial importance for hypertext systems in order to supply the user with operators for content oriented navigation within the hypertext graph. According to the semiotic categories there are the following types of hypertextual relations:

- Syntagmatic relations are derived from the surface structure of the documents within the hypertext.

The relation "next-passage-within-the-same-text" eg hold between two text units which can be found in the same document at adjacent positions. This means that our notion of syntagmatic relations excludes surface structures of hypertext presentation structures at runtime, which are generated according to semantic and pragmatic relations. Thus the computing of syntagmatic

relations is straightforward.

- Semantic relations which represent paradigmatic aspects of text units depend on the content of the text units. Informational inconsistency between text units for instance induces a semantic relation. Special types of semantic relations connect original and derived text units, which result from parsing, text condensation or generation.
- Pragmatic relations represent dependencies between the dialog context, the intentions of the user which may be given as an interest profile, the content of the hypertext and the intentions of the authors of the text units. Examples for pragmatic relations are "next-lesson" which holds between tutorial text units or "next-relevant-text" which gives an answer to a query.

III.1 Semantic Relations

Semantic relations are based on structural similarities of the semantic representations of the text units. Some of the large number of possible semantic relations will be defined in the following. For each relation a informal verbal introduction to its meaning - to show its relevance to the task of guiding the user through a hypertext graph -, a formal definition and an example will be given.

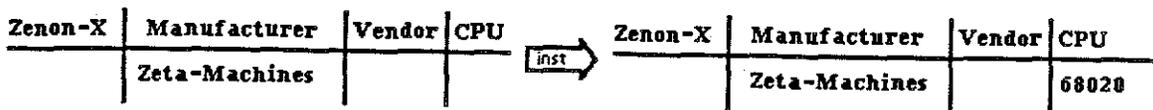
A formal definition of a relation is based on properties (slot-entries) of frames which are elements of the semantic representations of text units. Thus hypertextual relations can be inferred by the means of relations between frames.

- same name:

The hypertext consists of several text units belonging to different texts. Each unit has its own set of frames as semantic representation the name of each frame denoting a concept the text is about (synonyms being normalized) and the slots cumulating the facts concerning this concept as entries. A frame may be member of more than one frame set and there may be several frames (as members of different frame sets) with the same name, but with different slot fillers. Most of the semantic relations as defined below describe relations between different descriptions of the same topic, therefore a means of testing whether two frames have the same name is a prerequisite of the definition of such relations.

Def. 5:

$$\forall f_1, \forall f_2 : eqn(f_1, f_2) \leftrightarrow is-frame(f_1) \wedge is-frame(f_2) \wedge fn(f_1) = fn(f_2)$$



- complement:

All information contained in the first frame must also be found in the second one completed by (at least) an additional entry. The information of the first frame is confirmed and completed by the second one.

Def. 6:

$$(1) \quad \forall f_1, \forall f_2 \forall s : \text{compl}_1(f_1, f_2) \leftrightarrow \text{eqn}(f_1, f_2) \wedge \exists f : [\text{inst}(f, f_1) \wedge \text{inst}(f, f_2)] \wedge$$

$$(2) \quad \text{inst}(f, f_2)$$

- (1) The relation is defined on the set of frames, therefore both parameters must be frames. Additionally they should have the same name (eqn def. 5) to indicate that they refer to the same topic. These presumably different frames taken from two distinct text units must have the same slot structure to be comparable, therefore the inst relation (def.3) must hold between each of them and a third frame.
- (2) f_2 must contain an additional entry in any slot and thus be an instance of f_1 .

Example: compl



- x-complement:

The information given by these frames is disjunct. Both frames are needed to obtain the complete information.

Def. 7:

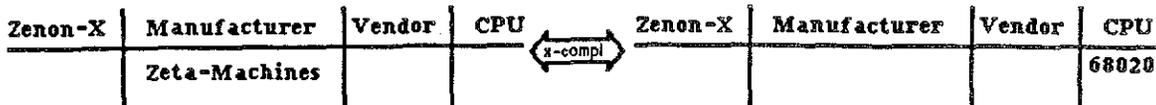
$$(1) \quad \forall f_1 \forall f_2 : \text{x-compl}_1(f_1, f_2) \leftrightarrow \text{eqn}(f_1, f_2) \wedge \exists f : [\text{inst}(f, f_1) \wedge \text{inst}(f, f_2)] \wedge$$

$$(2) \quad \forall s \forall s' : [\text{is-slot}(f_1, s) \wedge \text{is-slot}(f_2, s') \wedge$$

$$(3) \quad \forall e : [\text{is-entry}(f_1, s, e) \wedge \text{is-entry}(f_2, s', e) \Rightarrow \text{sn}(s) \neq \text{sn}(s')]]$$

- (1) see def. 6.
- (2,3) Corresponding slots of f_1 and f_2 (the slots have the same name) must contain disjunct sets of entries. Thus, if a concept is entry to both of the frames, it must be entry to slots with different names.

Example: x-compl



- add-inf:

This relation is similar to the complement relation, but the focus is on a special property of the frames, therefore the relation has a slot as a third argument.

Def. 8:

- (1) $\forall f_1, \forall f_2, \forall s : \text{add-inf}_f(f_1, f_2, s) \leftrightarrow \text{eqn}(f_1, f_2) \quad \wedge \exists f : [\text{inst}(f, f_1) \wedge \text{inst}(f, f_2)] \quad \wedge$
- (2) $\exists s' : [\text{is-slot}(f_1, s) \quad \wedge \text{is-slot}(f_2, s') \quad \wedge \text{sn}(s) = \text{sn}(s')] \quad \wedge$
- (3) $\forall e : [\text{is-entry}(f_1, s, e) \Rightarrow \text{is-entry}(f_2, s', e)] \quad \wedge$
- (4) $\exists e' : [\text{is-entry}(f_2, s', e') \wedge \neg \text{is-entry}(f_1, s, e')]]$

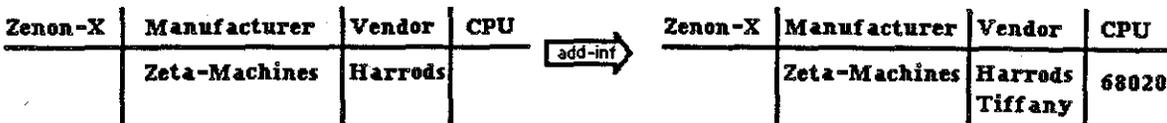
(1) see def. 6

(2) The specified slot must be slot of f_1 . f_2 must have a slot with the same name.

(3) All entries, which are assigned to the slot of the first frame, must be assigned to the corresponding slot of the second one too.

(4) The second frame must contain an additional entry to this slot.

Example: add-inf (Vendor)



- alt-inf:

This relation is derived from add-inf in the same way x-compl is derived from compl.

Def. 9:

- (1) $\forall f_1, \forall f_2, \forall s : \text{alt-inf}_f(f_1, f_2, s) \leftrightarrow \text{eqn}(f_1, f_2) \quad \wedge \exists f : [\text{inst}(f, f_1) \wedge \text{inst}(f, f_2)] \quad \wedge$
- (2) $\exists s' : [\text{is-slot}(f_1, s) \quad \wedge \text{is-slot}(f_2, s') \quad \wedge \text{sn}(s) = \text{sn}(s')] \quad \wedge$
- (3) $\exists e : \text{is-entry}(f_1, s, e) \quad \wedge \exists e' : \text{is-entry}(f_2, s', e') \quad \wedge$
- (4) $\forall e : [\text{is-entry}(f_1, s, e) \Rightarrow \neg \text{is-entry}(f_2, s', e)] \quad \wedge$
- (5) $\forall e' : [\text{is-entry}(f_2, s', e') \Rightarrow \neg \text{is-entry}(f_1, s, e')]]$

(1) see def. 6.

(2) see def. 8

(3) Each of the corresponding slots must contain at least one entry.

(4,5) The set of entries to these slots must be disjunct.

Example: alt-inf (Vendor)



- conflict:

The frames contain inconsistent information.

Def. 10:

$$\begin{array}{llll}
 (1) & \forall f_1 \forall f_2 : \text{confl}_1(f_1, f_2) & \leftrightarrow \text{eqn}(f_1, f_2) & \wedge \exists f : [\text{inst}(f, f_1) \wedge \text{inst}(f, f_2)] \wedge \\
 (2) & & \exists s \exists s' : [\text{is-slot}(f_1, s) & \wedge \text{is-slot}(f_2, s') \wedge \text{sn}(s) = \text{sn}(s') \wedge \\
 (3) & & \text{is-single}(f_1, s) & \wedge \text{is-single}(f_2, s') \wedge \\
 (4) & & \exists e_1 \exists e_2 : [\text{is-entry}(f_1, s, e_1) \wedge & \text{is-entry}(f_2, s', e_2) \wedge \\
 (5) & & [\text{is-frame}(e_1) & \wedge \text{is-frame}(e_2) \wedge \\
 (6) & & [\text{fn}(e_1) \neq \text{fn}(e_2) & \wedge \neg[\text{e-is-a}(e_1, e_2) \vee \text{e-is-a}(e_2, e_1)]] \vee \\
 (7) & & \text{fn}(e_1) = \text{fn}(e_2) & \wedge \text{confl}(e_1, e_2)] \vee \\
 (8) & & \neg \text{is-frame}(e_1) & \wedge \neg \text{is-frame}(e_2) \wedge e_1 \neq e_2 \text{]}]
 \end{array}$$

- (1) see def. 6.
- (2) In f_1 and f_2 there must be two corresponding slots which fulfill the following conditions.
- (3) The two slots must be singletons, therefore they must not contain more than one entry.
- (4) Each of the two slots contains an entry.
- (5) If the entries are frames then (7) or (8).
- (6) If these (see 6) frames have different names, the conflict relation must not hold if one frame is a specialization of the other.
- (7) If they have the same name, the conflict relation must hold between them.
- (8) If the entries are no frames they must be different strings.

Example: confl

Zenon-X	Manufacturer	Vendor	CPU		Zenon-X	Manufacturer	Vendor	CPU
	Zeta-Machines		8080	↔ confl		Zeta-Machines	Harrods Tiffany	68020

- property coincidence:

Similar properties are assigned to two distinct objects. The passages may be read in order to compare these objects with respect to other properties.

Def. 11:

$$\begin{array}{llll}
 (1) & \forall f_1 \forall f_2 \forall s : \text{same-prop}_1(f_1, f_2, s) & \leftrightarrow \text{is-frame}(f_1) & \wedge \text{is-frame}(f_2) \wedge \\
 (2) & & \neg \text{eqn}(f_1, f_2) & \wedge \exists f : [\text{inst}(f, f_1) \wedge \text{inst}(f, f_2)] \wedge \\
 (3) & & \exists s' : [\text{is-slot}(f_1, s) \wedge \text{is-slot}(f_2, s') & \wedge \text{sn}(s) = \text{sn}(s') \wedge \\
 (4) & & \exists e : [\text{is-entry}(f_1, s, e) \wedge \text{is-entry}(f_2, s', e) &]
 \end{array}$$

- (1) With the exception, that the names must not be the equal, see def. 6.
- (2) Because eqn does not hold it frames must be tested that f_1, f_2 are frames.
- (3) see def. 8
- (4) The two slots contain the same entry.

Example: same-prop (CPU)

Sun	Manufacturer	Vendor	CPU		Zenon-X	Manufacturer	Vendor	CPU
			68020	↔ same-prop ↔		Zeta-Machines	Tiffany	68020

These relations which are defined on a set of frames can be used to define relations on semantic representations of text units which are sets of frames. Thus, the semantic relations are completely independent from the surface structure of texts. Two interrelated text units may therefore be part of the same text or belong to different ones. The predicate unit-rep tests whether a set of frames represents a text unit. A complete definition of unit-rep would require a deeper understanding of the parsing process which is out of the scope of this paper. The following lemma which can be derived from the complete definition suffices our purpose.

Lemma

$$\forall K : \text{unit-rep}(K) \Rightarrow \forall f \in K : [\text{is-frame}(f) \wedge \neg \exists f' \in K : \text{eqn}(f, f')]$$

A relation between two text units holds iff the corresponding frame oriented relation holds between two frames which are members of the units (def. 12). The semantic relations which hold between frames with respect to special properties (eg def. 8) are used to define relations on text units according to def. 12a. A relation on text units, which depends only on the interrelation of two frames may be a too weak restriction. A relation of partial identity (id_n) between text units demands that the intersection of their frame sets must have at least n elements (def. 13). The intersection between this relation and a frame based semantic relation may be used to enforce stronger restrictions (def. 14, def. 14a).

Def. 12:

$$\begin{aligned} (1) \quad \forall K_1, \forall K_2 : \text{rel}_K(K_1, K_2) &\leftrightarrow \text{unit-rep}(K_1) \wedge \text{unit-rep}(K_2) \wedge \\ (2) &\quad \exists f_1 \in K_1, \exists f_2 \in K_2 : \text{rel}_K(f_1, f_2) \end{aligned}$$

Def. 12a:

$$\begin{aligned} (1) \quad \forall K_1, \forall K_2, \forall f, \forall s : \text{rel}_K(K_1, K_2, f, s) &\leftrightarrow \text{unit-rep}(K_1) \wedge \text{unit-rep}(K_2) \wedge \\ (2) &\quad f \in K_1 \wedge \exists f_1 \in K_2 : \text{rel}_K(f, f_1, s) \end{aligned}$$

Def. 13:

$$\begin{aligned} (1) \quad \forall K_1, \forall K_2 : \text{id}_n(K_1, K_2) &\leftrightarrow \text{unit-rep}(K_1) \wedge \text{unit-rep}(K_2) \wedge \\ (2) &\quad |K_1 \cap K_2| \geq n \end{aligned}$$

Def. 14:

$$\forall K_1, \forall K_2 : \text{rel}_n(K_1, K_2) \leftrightarrow \text{rel}_K(K_1, K_2) \wedge \text{id}_n(K_1, K_2)$$

Def. 14a:

$$\forall K_1, \forall K_2, \forall f, \forall s : \text{rel}_n(K_1, K_2, f, s) \leftrightarrow \text{rel}_K(K_1, K_2, f, s) \wedge \text{id}_n(K_1, K_2)$$

III.2 Pragmatic Relations

Pragmatic relations between text units reflect the situational context in which the dialog between user and hypertext system takes place. This can be described by a variety of parameters - eg models of the discourse or the intentions of users or hypertext authors. In the following we will restrict our interest to two aspects of dialogs which are important for the design of hypertext systems and can be tackled by the formal instruments we have introduced above:

1. the amount of details the text units contain (D_U)
2. the specificity of the user's wishes - formulated as a query (S_Q).

The combination of these aspects allows to distinguish several dialog situations. Weyer's "Dynamic Book" (WEYER 82) supplies the user with original text units, therefore D_U cannot be manipulated. Variations of specificity are gained on a syntagmatic level by alternatively presenting titles, subtitles or text passages. According to different S_Q 's two prototypes of dialog situations may be defined.

1. If S_Q is high - several terms are selected from the subject index - the system behaves like an encyclopedia from which the user may derive information by dialog.
2. Unspecific (or unknown) queries enforce browsing on the syntagmatic level, eg skimming the headlines of the next chapters.

Systems which are able to present information with several degrees of abstraction allow to adjust D_U to the dialog situation. Intelligent tutoring systems adapt the level of abstraction heuristically based on an explicit model of the student and his presumable information needs that may be derived from this model (SLEEMAN 83). Information retrieval systems - on which we will focus our interest in the following - allow the specification of D_U - by selecting more general or more special index terms - and S_Q - eg by ranking the search terms. To what extent text units match a query - with respect to the content and the degree of abstraction - can be defined by a relevance relation $\text{relev}(Q, TU, r)$, where Q is a query, TU a text unit and r the degree of relevance. In our retrieval model we use the explicit representation of the semantics of a text unit to define the degree of relevance by relating the text unit to the query as shown in the following definition.

Def. 15:

$$\begin{aligned}
 (1) \quad \forall Q \forall F \forall r : \text{relev}_{\text{eqn}}(Q, F, r) &\leftrightarrow \exists T : F \in T \wedge r = \sum_{f_1 \in Q} \sum_{f_2 \in F} g_{f_1} \cdot \text{chr}_{\text{eqn}}(f_1, f_2) \\
 (2) &+ \sum_{L \in T} \sum_{f_1 \in Q} \sum_{f_2 \in L} g_{f_1} \cdot \text{chr}_{\text{eqn}}(f_1, f_2)
 \end{aligned}$$

In this formula Q represents the query, F the semantic representation of a text unit and r the degree of relevance.

- (1) The relevance of a text unit is computed by summing up the weights g of those frames, which are related (by a predefined relation, here : eqn) to a frame of the query. The weight g , which can be defined by the user during the dialog (see dialog example in section IV.), is in the range from 1 to 10. chr is the characteristic function of this relation mapping all pairs (f_1, f_2) which are elements of the relation on 1 and all other pairs on 0.
- (2) The relevance of a text unit can't completely be separated from the relevance of the text the unit is taken from. Therefore the overall relevance of the text (a set of text units denoted by T) is added to the relevance of the text unit itself.

Recall and precision of the query may be adjusted by choosing an appropriate relation, which must hold between the frames of query and text unit. The relation of name similarity (eqn see def. 5) is the most elementary of the possible tests. Instead of chr_{eqn} the characteristic function of the union of eqn and e-is-a may be applied in the relevance measure yielding a controlled expansion of the recall. (This is comparable to downposting operations in thesaurus based retrieval systems.)

IV. GUIDING THE USER FROM SEARCH TERMS TO RELEVANT TEXT CONTENTS: A DIALOG EXAMPLE

After discussing the hypertext model we want to give some insight to the experimental information system TOPOGRAPHIC which supports knowledge based interaction facilities which provide content oriented access to text knowledge bases. The results from the text analysis and condensation process stored therein may be regarded as text units in the sense mentioned above.

We illustrate the essential features of the user interface - which supplies the user with a 'graphical retrieval language' meeting the needs of hypertexts and is based on the representation structures as defined above - by means of a (slightly simplified) dialog. This example shows all layers of information that can be accessed in a series of zooming operations (which means switching from a more general to a more special text unit) in order to give an overview of the system's capabilities. On each layer shown the zooming is prepared by selection operations that facilitate focussing on relevant sections of the layer below. If the items to be selected are not visible due to the limited size of the screen, browsing is used to access them. (A real life dialog may not have such a straightforward zooming structure, there might be 'loops' in it in cases the user returns to higher levels to change his focus up there and then zooms again. Thus a feedback facility for query refinement is given.)

At the beginning of the dialog the most general concepts of the world knowledge base (which can be thought of as a representation of a hypothetical text unit defining the terminology of the domain of discourse) are presented to the user so that he is informed about the domain of discourse. The user starts to explore this conceptual hierarchy by applying the browse option to the concepts 'Product', 'Software', 'System Software' and 'Operating System' (cf fig. 1). (He needn't know that they are frames, he only operates on graphical items.) To shorten the process of investigation, the user can

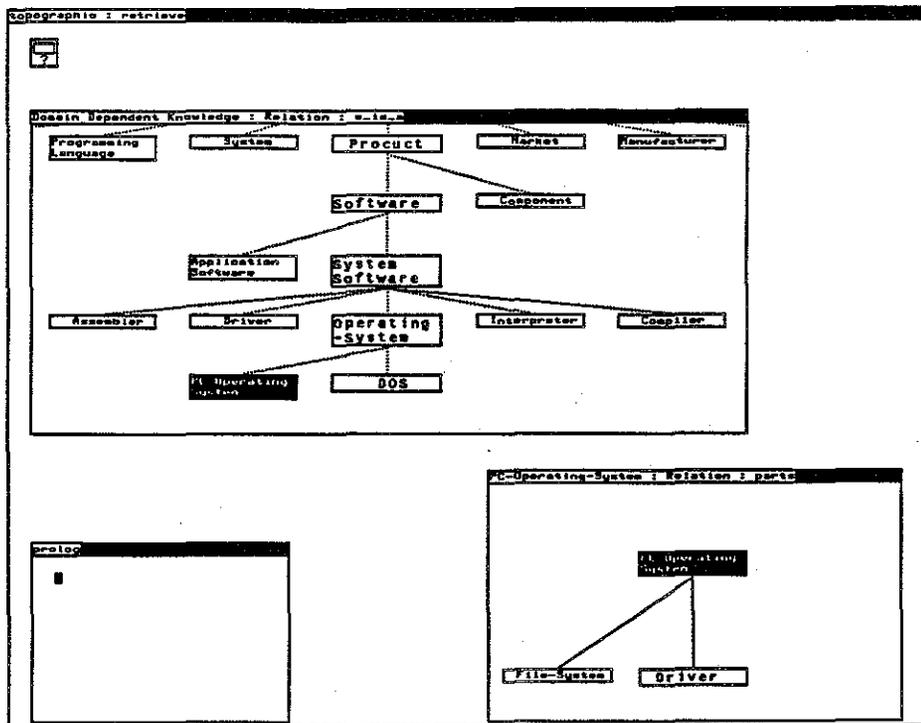


fig. 1

enter search terms tentatively, which are not offered by the system at the time being. The command "find('Operating System')." entered via the "Prolog" window is equivalent to the browse-sequence mentioned above. Additional to the 'e-is-a' relation connecting the concepts (see def. 4) other relational dependencies of one concept can be shown on demand (such as the 'parts' relation which is similar to but not identical with the relation holding between an object and its parts (cf fig. 1)).

While browsing the user constructs a query by selecting relevant terms (selected terms are presented in inverted mode). Zooming the window which presents the domain dependent knowledge on the taxonomic level yields a list of all selected terms and their activation weights indicating their relevance for the further retrieval process (cf fig. 2). (The weights may be increased or decreased if necessary.) A subsequent zooming of the 'selected terms' object produces a list of text passages which are related to the query by the relevance relation (see def. 15). The passages are ranked according to descending values of r . At the same time the graphical representation of the knowledge base shrinks to the format of a box due to the shift of the user's attention to information layers below. Bibliographical information (title etc.) about the text and a short textual extract of the beginning of each passage are given. In fig. 2 the semantic representation of the most relevant passage (topic profile of passage k18 of text t1) is shown. Applying the zoom operator to other list elements would reveal their representation, respectively. Zooming the node 'UCSD-PASCAL' in the topical network of the most relevant text

part reveals the factual information about this PC-Operating-System that was extracted from this particular text part during the analysis process (cf fig. 3), whereas zooming the whole window results in the corresponding text passage (cf fig. 2).

The screenshot shows a window titled "topographie : retrieve". It contains several panels:

- AI : k18 : topic profile:** A hierarchical diagram with nodes: "Übersicht: PC-System", "UCSB-PASCAL", "CP/M", "Produkt", "IBM", "Personal-Computer", and "IBM-Personal-Computer".
- 2 relevant passages taken from 2 texts:**

k18 : 12 (s1)	Nit Bisketten wird der IBM-Personal-Computer unter des Software-BUS 88 von NI ...	int: Spielkarte CHIText: Bar no
k21 : 12 (s2)	Bereits jetzt steht ein PASCAL-Compiler zur Verfügung und ein Anwendungs ...	int: Spielkarte CHIText: Bar no
- Selected terms:** A table with "PC-Operating-System" selected.
- Text Passage (k1 : k18):** A text box containing the following German text:

Nit Bisketten wird der IBM-Personal-Computer unter des Software-BUS 88 von NI... betrieben. Ueber diese Schnittstelle steht die ganze Sprechpalette zur Verfügung. Besonders hinaus wird, allerdings nicht von IBM selbst, CP/M 86 und UCSB-PASCAL angeboten. Auf diesem Softwarekonzept basierend ist fuer diesen Rechner der Anschluss an vielfaelligste Anwendungsprogramme und Sprachen gestattet.

fig. 2 (above)

fig. 3 (below)

This screenshot is similar to the one above but includes additional features:

- semantic browser:** A menu with options: "alt-inf (Personal-Computer)", "complement", "x-complement", and "conflict".
- UCSB-PASCAL: slots/entries:** A table showing relationships between "Personal-Computer" and "IBM-Personal-Computer".

Personal-Computer	IBM-Personal-Computer
Vendor	
Manufacturer	SeiTech
Price	

Switching between topic profiles may be accomplished not only by zooming from the list of text units but by browsing the current text profile as well. The user is then asked (a pop up menu will be displayed (cf fig. 3)) whether he wants to see the next relevant passage or another relevant one which is connected to the current text unit by a semantic relation (cf fig. 3). Only those relations will be offered which promise a successful continuation of the browsing process, i.e. lead to other text units. Thus, the combination of pragmatic and semantic or syntagmatic information about possible successors of the current text unit allows to restrict the set of new objects to be presented efficiently. In our example, the conjunction of requirements selected (all items of the menu in fig. 3 yields one salient text unit (cf fig. 4)) containing a frame named UCSD-Pascal as well. These relations may be inferred from the properties of the two frames according to def. 14 and def. 14a (definition of hypertextual relations by frame relations) and:

- a) x-compl : all entries are different (def. 6).
- b) alt-inf : the sets of entries of two corresponding slots are disjunct, i.e. the product is sold by different vendors (def. 9).
- c) conf : there are different entries to corresponding slots which are singletons, i.e. supposed a product has no more than one manufacturer there is a conflict between the two frames (def. 10).

Again, this dialog fragment is somewhat idealized, in real live dialogs a set of more or less qualifying text units might be obtained, which would then be presented in a table similar to the list of relevant text passages retrieved by the query (fig. 2).

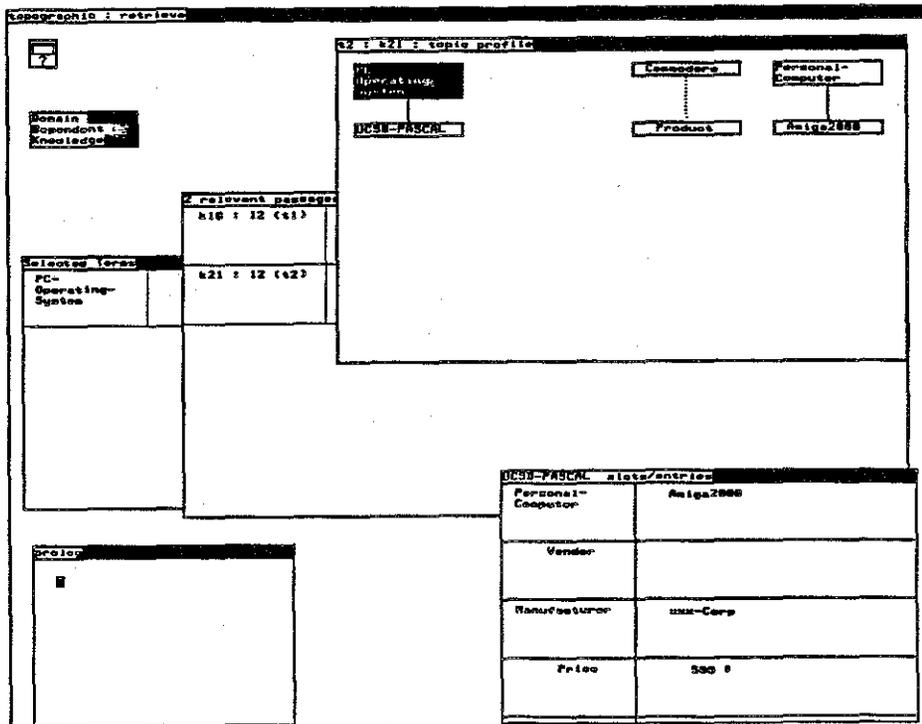


fig. 4

(a remark on the example: TOPOGRAPHIC supports the retrieval of German texts, therefore the text example is taken from a German (computer-) magazine. This text is about software products available for the IBM-PC. For convenience, all identifiers occurring in the example have been translated.)

V. IMPLEMENTATIONAL REMARKS

The development of the TOPOGRAPHIC system is supported by BMFT/GID under contract '1020018 1'. It is implemented in Prolog and C on a CADMUS 9200 with UNIX. The Prolog-System as used in TOPOGRAPHIC was developed as part of the project by augmenting the IF-Prolog interpreter with new built-in predicates. Additional to common features of Prolog it supports access to frame based knowledge bases and graphical tools for interface management, which are implemented in C for the purpose of efficient execution. The basic frame predicates used in the definitions above are a (small) subset of the predicates provided for knowledge base access. TOPOGRAPHIC's graphics-predicates include multi window and mouse interaction techniques as well.

REFERENCES

- CHRISTODOULAKIS ET AL. 86: Christodoulakis, S.; Ho, F.; Theodoridou, M.
The Multimedia Object Presentation Manager of MINOS: A Symmetric Approach.
In: SIGMOD Record, Vol. 15, No. 2, 1986, pp. 295-310.
- DANSEREAU/HOLLEY 82: Dansereau, D. F.; Holley, C. D.
Development and Evaluation of a Text Mapping Strategy.
In: Flammer/Kintsch (eds.): Discourse Processing, Amsterdam, North Holland, 1982, pp. 536-554.
- HAHN 86: Hahn, U.
On Lexically Distributed Text Parsing: A Computational Model for the Analysis of Textuality on the Level of Text Cohesion and Text Coherence.
In: Kiefer, F. (ed.): Linking in Text, Dordrecht, D. Reidel, 1986.
- HAHN/REIMER 86: Hahn, U.; Reimer, U.
TOPIC Essentials. In: Coling 86: Proc. of the 11th Conf. on Computational Linguistics, August, 25-29, 1986, Bonn, FRG, ACL 1986.
- HAYES 79: Hayes, P. J.
The Logic of Frames.
In: Metzger, D. (ed.), Frame Conceptions and Text Understanding, Berlin, New York : De Gruyter, 1979, pp. 46-61.
- HAYES/HENDRIX 81: Hayes, P. J.; Hendrix, G. G.
A Logical View of Types.
In: Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling, SIGART Newsletter Nr. 74, Jan., 1981, pp.128-130.
- MINSKY 75: Minsky, M.
A Framework for Representing Knowledge.
In: Winston, P. (ed.): The Psychology of Computer Vision, New York: McGraw Hill, pp. 211-277.
- REIMER 86: Reimer, U.
A System-Controlled Multi-Type Specialization Hierarchy.
In: Kerschberg, L. (ed.): Expert Database Systems. Proceedings of the 1st International Workshop, Menlo Park/CA: Benjamin/Cummings, 1986, pp. 173-187.
- REIMER/HAHN 85: Reimer, U.; Hahn, U.
On Formal Semantic Properties of a Frame Data Model.
In: Computers and Artificial Intelligence 4. 1985. No. 4., pp. 335-351.
- SLEEMAN 83: Sleeman, D. H.
Inferring Student Models for Intelligent Computer Aided Instruction.
In: Michalsky et al. (eds.), Machine Learning. An Artificial Intelligence Approach, pp. 483-510, Palo Alto, CA: Tioga, 1983.
- THIEL/HAMMWOHNER 87: Thiel, U.; Hammwöhner, R.
Informational Zooming: An Interaction Model for the Graphical Access to Text Knowledge Bases.
In: Yu; van Rijsbergen (eds.), Proc. of the 10th Annual Int. ACM SIGIR Conf. on Research & Development in Information Retrieval. New Orleans, Louisiana, 1987, pp. 45-56.
- WEYER 82: Weyer, S. A.
The Design of a Dynamic Book for Information Search.
In: Int. J. of Man-Machine Studies, Vol. 17, 1982, pp. 87-107.
- WEYER/BORNING 85: Weyer, S. A.; Borning, A.H.
A Prototype Electronic Encyclopedia.
In: ACM Transactions on Office Information Systems, Vol. 3, No. 1, 1985, pp. 63-88.
- WOELK ET AL. 86: Woelk, D.; Kim, W.; Luther, W.
An Object Oriented Approach to Multimedia Databases.
In: SIGMOD Record, Vol. 15, No. 2, 1986, pp. 311-325.

SuperBook: An automatic tool for information exploration - hypertext?

Joel R. Remde, Louis M. Gomez, Thomas K. Landauer

Bell Communications Research
435 South St.
Morristown NJ 07960

ABSTRACT

The goals and methods of the text browser, SuperBook, are compared with those of hypertext systems in general. SuperBook, intended to provide improved access to text existing in electronic form, employs cognitive tools arising from human computer interaction research, such as full-text indexing, adaptive aliasing, and dynamic views of hierarchical information. Superbook automatically preprocesses on-line text written for paper publication, and produces a multi-window display, including a dynamic table of contents, pages of text, and a history of search words. Although SuperBook and hypertext share common goals of improved search and navigation, SuperBook is designed for accessing existing documents while most hypertext systems are better suited for authoring new information structures. Further studies are needed to evaluate the effectiveness of each of these kinds of systems.

INTRODUCTION

The text-browser that we will describe here, SuperBook, performs some of the functions often envisioned for hypertext systems, but differs in several significant ways from implemented or proposed examples of such systems. SuperBook takes as input ordinary text in a standard text formatting language, and automatically converts it into a multi-windowed browser with rich search, navigation, annotation and display enhancements. We are not especially concerned about whether it is or is not appropriate to call SuperBook "hypertext", but we do think it is instructive to draw attention to the similarities and differences in goals, approaches and methods.

As we understand the relevant history and philosophy, hypertext is proposed as an improvement on several perceived deficiencies of traditional paper text presentation of information. Let us list some of these deficiencies.

1. It is too hard to find information in ordinary text.

2. It is too hard to acquire information in a sequence other than that determined by the author.
3. It is impossible to properly present many domains of knowledge because some important relations cannot be represented in a linear structure.
4. It is extremely difficult to integrate and update large bodies of frequently changing information from many different sources.
5. Book style organization is a highly inefficient storage and retrieval technology; the same information often appears in many different places.

Hypertext is, then, conceived as an alternative information delivery vehicle in which modules of text (or other media) are stored in a different kind of data structure, ordinarily a graph, and some graph-like display is used to describe the modules and their interrelations to the reader. It is claimed, or at least hoped, that this form of organization and presentation of information will be more effective and desirable. Both readers and authors are expected to benefit. Readers, it is hoped, will be better able to find just the right information in the right sequence to serve their particular needs and support their individually optimal trajectories of learning, problem solving or entertainment. Authors, it is hoped, will be better able to organize and relate information and ideas. In addition, multiple, asynchronous "authorship" should be facilitated. Both author and reader, who may be the same person, as well as communities of users, will presumably profit from the greater freedom, flexibility and efficiency of this mode of organization.

The SuperBook project had its origin in somewhat different and narrower, although overlapping goals. First, a very large and growing proportion of textual material is being produced in electronic form with word-processing and text-formatting equipment. The delivery of such documents directly in electronic form is very attractive. However, with currently available tools, using text in this form is *not* very attractive. Video screens are small, hard to read and awkward to transport. Paging or scrolling through raw text files intended for print is slow and disorienting. With computer power to help, we thought, it should be possible to ameliorate these difficulties and to provide additional useful facilities that would compensate for the inherent disadvantages of linear text viewed on a video screen. The goal was simply to use computation to improve the usability of on-line text.

There was, however, another source of our interest in this project. Along with several colleagues, we have been studying a variety of issues in the human use of computers for textual information handling tasks. This has led to the discovery of some fundamental behavioral obstacles in such tasks, and to the invention of some new computer-based "cognitive tools" that can help to overcome them. We wanted to apply these new tools to improve the way people obtain and use information from books. We were especially interested in books used for learning or for reference, (as opposed to books that are read like novels) because these seem to pose the greatest need and opportunity for assist-

ing the user.

One relevant question that had been studied was why people frequently fail to find desired information in textual and other information retrieval devices [Furn83], [Furn], [Gome84], [Gome]. Very briefly, it was found that the principle cause of failure is that, when searching for something, users describe the things they want in different terms from those by which the system knows them. One remarkably effective solution, typically raising search success rates by a factor of four, from 20% to 80%, is just to greatly increase the number of names by which each information object can be reached. This is called *unlimited aliasing* or rich indexing. Several techniques for accomplishing it have been tested with favorable results in both laboratory experiments and prototype systems [Furn85], [Gome84], [Gome]. Full-text and adaptive indexing are easily automated techniques that seemed especially suitable for enhancement of existing on-line books.

Another information-handling aid that had arisen from this research was the *fish-eye viewer* [Furn85] that shows hierarchical structured information selectively according to a "degree-of-interest function". This function is based on nearness in the text to the point of focus and the *a priori* importance of higher levels in the hierarchy. Since there was also empirical evidence of the utility of this device for improving navigation within text, we wanted to apply a version of it too, to on-line books.

Thus, our goals overlapped at least the first two hypertext rationales listed above - improved search and navigation. Some partial and indirect solutions to the other problems that have motivated the hypertext vision also fall out of an attempt to provide a powerful and convenient delivery mode for textbooks and references. We will postpone further discussion of these until the details of the current implementation and design have been presented. In the discussion section at the end of the paper we will consider how the results of our effort merely to make good electronic books overlaps and differs from the accomplishments of hypertext projects as such.

WHAT SuperBook IS AND DOES

Here is a quick overview of the system. A narrative scenario of its use, with illustrations, and details of implementation follow.

The SuperBook software takes as input a text document prepared for a standard formatting package such as *TROFF*, *Scribe*, or *Interleaf*, and produces an enhanced delivery mode for the document. It analyzes the words and heading structure of the text and creates a data structure that allows pieces of text to be presented in a print-like format in one or more arbitrary-sized windows (see Figure 5), which can be scrolled, paged, highlighted, or selected by mouse. The full heading hierarchy, providing information about the location of the text in the document, is displayed above the text in each window, and is updated as necessary when scrolling or paging.

Appearing in a separate window is a *Table of Contents* of the text, constructed au-

tomatically by SuperBook from the formatting macros specifying hierarchically ordered headings. This dynamic *Table of Contents* shows varying levels of detail of chapters and sections, like *fisheye*. The user controls the degree of detail by opening up sections by mouse selections and displaying subheadings to any desired depth. Sections from the *Table of Contents* can be selected by mouse and displayed as text.

SuperBook builds a full-text index or concordance that is used to find all occurrences of any word, word stem, or boolean combination thereof. Search words are entered from the keyboard or by mouse selection from the screen. The *Table of Contents* is coordinated with the full-text index to show the distribution of a search word or phrase throughout the document. The number of occurrences of the word is posted alongside the title for each section in the *Table of Contents* providing extra orientation to the user concerning which sections are likely to be of interest.

There is also a facility for creating aliases. For words seldomly occurring or not occurring at all in the text, readers can add other words as index synonyms. The repertoire of aliases built up by a community of users improves the likelihood of a hit when doing word lookups. In addition, there is a facility to annotate the text. Users can open a workspace window, enter a comment or addition of any length, and, by a mouse click, leave an icon containing their initials and date in the margin of the visible text. The comment is stored out of sight, retrievable by mouse, and, if so desired, indexed by all its contained terms as part of the general index. Finally, there is limited provision for figure graphics. If the original text contains figures that can be automatically generated or are stored as bit-maps, they can be automatically displayed in a separate window.

NARRATIVE SCENARIO

SuperBook has been demonstrated on three documents to date - the *LSSGR*, the *S Language for Data Analysis*, and a Lisp tutorial. Here we will show a typical scenario using SuperBook on the *LSSGR*. The *LSSGR*, or *LATA Switching Systems Generic Requirements*, is a seven volume document produced by Bell Communications Research, describing in detail requirements for telephone and switching equipment for manufacturers and vendors.

The *LSSGR* contains about 4 Megabytes of text with a rich hierarchical structure of sections and sub-sections - up to seven nested levels. Since the paper version of the *LSSGR* consists of seven volumes, the information relevant to a query may be distributed over more than one volume, making it cumbersome to use. SuperBook on the other hand can easily keep track of all seven volumes in a single database and on a single screen.

The *LSSGR* in SuperBook on the SUN workstation initially comes up with four windows. These include a *Title* window, a *Table of Contents* window, a *Page of Text* window, and a *Word Lookup* window. Extra, overlapping *Page of Text* windows may be created as desired. The *Title* window displays the title of the book implemented and also

contains some general command buttons. The *Table of Contents* window initially shows only the highest level sections of the book, which are usually chapters. The *LSSGR* is divided into three functional sections and only these appear in the initial *Table of Contents* (see Figure 1).

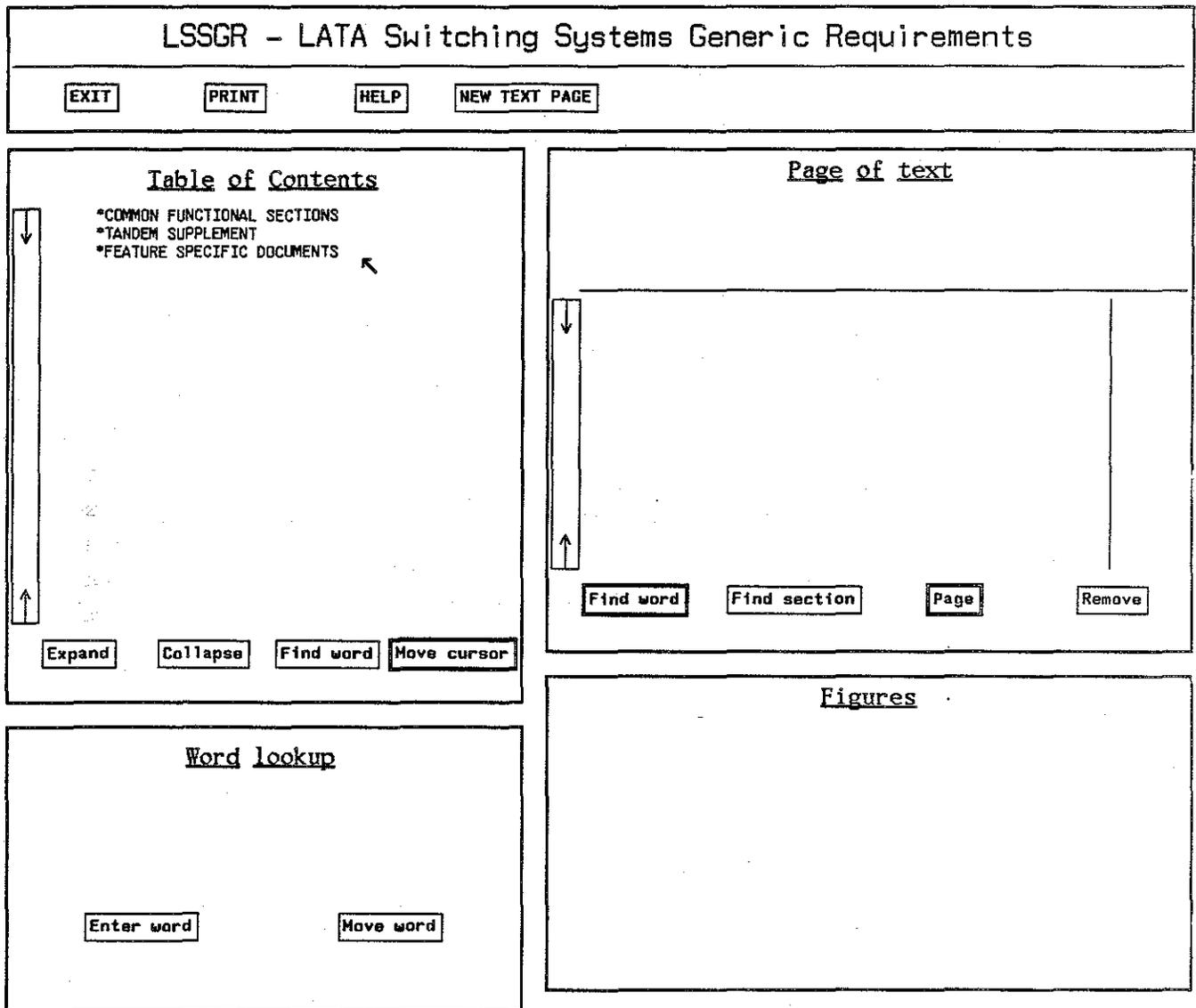


Figure 1. Initial configuration of the SuperBook version of the LSSGR document. The full book title appears at the top of the screen. The *Table of Contents* shows the highest level divisions of the book. The *Page of Text* and *Word Lookup* windows are initially empty.

The *Word Lookup* window maintains a list of the words or phrases searched so far in the book. In Figure 2, for example, the user entered the word *queuing*. The ending *-ing* is stripped off and all the different words throughout the entire document that begin with the root *queu-* are found and shown. In Figure 3 the user clicked on the **Find Word** button in the *Table of Contents* window whereupon SuperBook indicated the number of occurrences of the words matching *queu-* in each section of the *Table of Contents*.

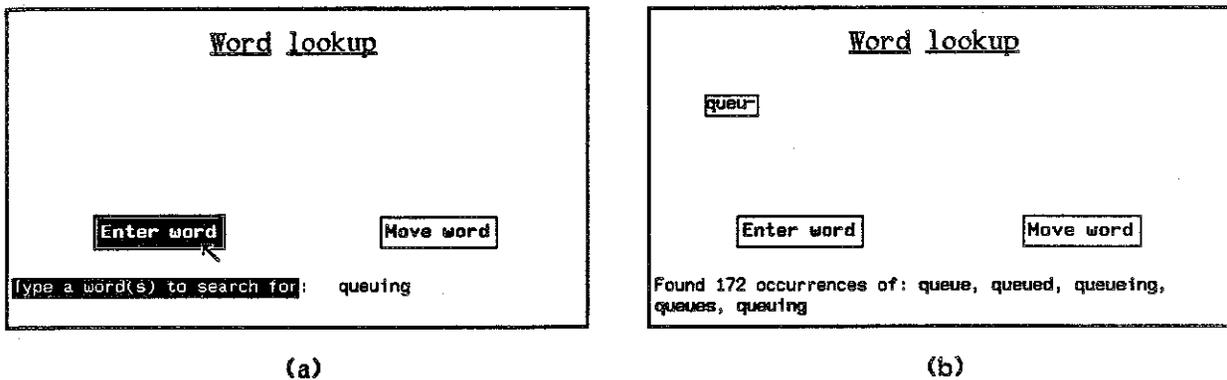


Figure 2. The user has typed *queuing* as a search word. SuperBook then lists the five forms of words beginning with the stem *queu-* that it found in the LSSGR, in 172 places.

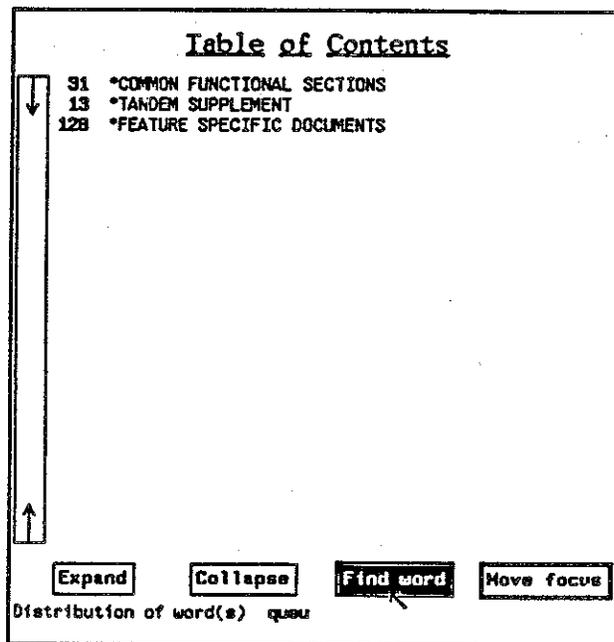


Figure 3. By clicking the mouse on **Find Word** in the *Table of Contents*, a breakdown of the 172 occurrences of *queu-* by section is shown.

Figure 4 shows how chapters or sections in the *Table of Contents* may be expanded into subsections. A section title is selected by clicking on it, and it is then expanded by clicking on **Expand**. The reader has decided to expand the section containing the highest frequency of the matches for *queu-*. Here we see the *FEATURE SPECIFIC DOCUMENTS* section of the *LSSGR* and one of the subsections, *OUTGOING FACILITY GROUP QUEUING* in the section *PRIVATE FACILITY FEATURES* being expanded. The word frequency for each newly displayed subsection is also posted.

Table of Contents	
31	*COMMON FUNCTIONAL SECTIONS
13	*TANDEM SUPPLEMENT
128	FEATURE SPECIFIC DOCUMENTS
	*81 RESIDENCE AND BUSINESS CUSTOMER FEATURES
184	*82 PRIVATE FACILITY FEATURES
	*84 CUSTOMER SWITCHING SYSTEM FEATURES
3	*18 COIN AND CHARGE-A-CALL
	*15 PUBLIC SAFETY
17	*28 MISCELLANEOUS
	*25 INTEROFFICE
	*38 CALL PROCESSING
	*31 SERVICE SWITCHING
	*35 SYSTEM MAINTENANCE
	*48 TRUNK, LINE AND SPECIAL SERVICE CIRCUIT TEST
4	*45 ADMINISTRATION

Expand Collapse Find word Move focus

Distribution of word(s) queu

(a)

Table of Contents	
31	*COMMON FUNCTIONAL SECTIONS
13	*TANDEM SUPPLEMENT
128	FEATURE SPECIFIC DOCUMENTS
	*81 RESIDENCE AND BUSINESS CUSTOMER FEATURES
184	82 PRIVATE FACILITY FEATURES
	*OUTWARD WIDE AREA TELECOMMUNICATIONS SERVICE (OUTWATS)
94	OUTGOING FACILITY GROUP QUEUING
44	*1.8 INTRODUCTION
1	2.8 USER PERSPECTIVE
43	*3.8 FEATURE REQUIREMENTS
	4.8 FEATURE FLOW DIAGRAM
8	5.8 GLOSSARY
	6.8 BIBLIOGRAPHY
1	*AUTOMATIC FLEXIBLE ROUTING
	*AUTHORIZATION CODES FOR AFR AND ACCOUNT CODES FOR
8	*MESSAGE DETAIL RECORDING TO CUSTOMER PREMISES
	*84 CUSTOMER SWITCHING SYSTEM FEATURES
3	*18 COIN AND CHARGE-A-CALL
	*15 PUBLIC SAFETY
17	*28 MISCELLANEOUS

Expand Collapse Find word Move focus

Distribution of word(s) queu

(b)

Figure 4. Expansion of sections in the *Table of Contents* window. The word frequencies in the subsections are shown as sections are expanded.

By clicking first on section *3.0 FEATURE REQUIREMENTS*, and then on **Find Word** in the *Page of Text* window, the first occurrence of *queu-* in section *3.0* is shown in the context of full text (see Figure 5). Notice the highlighting of the words matching *queu-*, and the higher level sections containing the text listed above the text itself, which provide an extra orientation to the user. To move through the text the user can scroll up or down with the scroll bar, or go forward or back a page with the **Page** button. To find the next occurrence of the current word the user can click again on **Find Word**, perhaps selecting another section from the *Table of Contents* first.

Table of Contents

31 *COMMON FUNCTIONAL SECTIONS
 13 *TANDEM SUPPLEMENT
 128 FEATURE SPECIFIC DOCUMENTS
 *81 RESIDENCE AND BUSINESS CUSTOMER FEATURES
 184 82 PRIVATE FACILITY FEATURES
 *OUTWARD WIDE AREA TELECOMMUNICATIONS SERVICE (OUTWATS)
 94 OUTGOING FACILITY GROUP QUEUING
 44 *1.0 INTRODUCTION
 1 2.0 USER PERSPECTIVE
 43 *3.0 FEATURE REQUIREMENTS
 4.0 FEATURE FLOW DIAGRAM
 6 5.0 GLOSSARY
 6.0 BIBLIOGRAPHY
 1 *AUTOMATIC FLEXIBLE ROUTING
 *AUTHORIZATION CODES FOR AFR AND ACCOUNT CODES FOR
 8 *MESSAGE DETAIL RECORDING TO CUSTOMER PREMISES
 *84 CUSTOMER SWITCHING SYSTEM FEATURES
 3 *18 COIN AND CHARGE-A-CALL
 *15 PUBLIC SAFETY

Expand Collapse Find word Move cursor

Distribution of word(s) queu

Page of text

FEATURE SPECIFIC DOCUMENTS
 82 PRIVATE FACILITY FEATURES
 OUTGOING FACILITY GROUP QUEUING
 3.0 FEATURE REQUIREMENTS
 3.1 FEATURE OPERATIONS

A call enters the switch on an incoming facility group or line. Either AFR or direct (code) access may be used to select the outgoing facility group. If an outgoing facility group for which the call is eligible (based, e.g., on the class of service of the incoming facility group) is available, the call should be routed and OFGQ plays no role. (The hunting in AFR or for direct (code) access should be applied to determine if an outgoing facility is available. Thus, for example, the hunting over Wide-Area Telecommunications Service (WATS) bands should be applied before an attempt is made to queue a call.) Otherwise, the call encounters the all-allowed-outgoing-facility-groups-busy condition. The switch should determine if the incoming facility group or line has been marked to prohibit queuing. If so, then

Find word Find section Page Remove

Looking for word(s): queu

Figure 5. Displaying text containing the search word queu in the Page of Text window. The desired section, 3.0 FEATURE REQUIREMENTS was first selected from the Table of Contents window.

In Figure 6 the user has clicked on the word *OFGQ* appearing in the *Page of Text*. This word is appended to the history list in the *Word Lookup* window and is searched just as if it were entered from the keyboard. A word distribution in the *Table of Contents* shows that it first occurs in section *1.0 INTRODUCTION*. When text containing its first occurrence in this section is displayed, we find the definition of this acronym (Figure 7).

Page of text

FEATURE SPECIFIC DOCUMENTS
 82 PRIVATE FACILITY FEATURES
 OUTGOING FACILITY GROUP QUEUING
 3.0 FEATURE REQUIREMENTS
 3.1 FEATURE OPERATIONS

A call enters the switch on an incoming facility group or line. Either AFR or direct (code) access may be used to select the outgoing facility group. If an outgoing facility group for which the call is eligible (based, e.g., on the class of service of the incoming facility group) is available, the call should be routed and OFGQ plays no role. (The hunting in AFR or for direct (code) access should be applied to determine if an outgoing facility is available. Thus, for example, the hunting over Wide-Area Telecommunications Service (WATS) bands should be applied before an attempt is made to queue a call.) Otherwise, the call encounters the all-allowed-outgoing-facility-groups-busy condition. The switch should determine if the incoming facility group or line has been marked to prohibit queuing. If so, then

Find word Find section Page Remove

Looking for word(s): queu

Word lookup

queu
OFGQ

Enter word Move word

Found 16 occurrences of: ofgq

(a)

(b)

Figure 6. The user has clicked on OFGQ in the text, causing a search to be performed on this word.

LSSGR - LATA Switching Systems Generic Requirements

EXIT
PRINT
HELP
NEW TEXT PAGE

Table of Contents

- *COMMON FUNCTIONAL SECTIONS
- *TANDEM SUPPLEMENT
- 10 FEATURE SPECIFIC DOCUMENTS
 - *81 RESIDENCE AND BUSINESS CUSTOMER FEATURES
 - 10 82 PRIVATE FACILITY FEATURES
 - *OUTWARD WIDE AREA TELECOMMUNICATIONS SERVICE (OUTWATS)
 - 10 OUTGOING FACILITY GROUP QUEUING
 - 7 **1.8 INTRODUCTION**
 - 1 2.8 USER PERSPECTIVE
 - 7 *3.8 FEATURE REQUIREMENTS
 - 4.8 FEATURE FLOW DIAGRAM
 - 1 5.8 GLOSSARY
 - 6.8 BIBLIOGRAPHY
 - *AUTOMATIC FLEXIBLE ROUTING
 - *AUTHORIZATION CODES FOR AFR AND ACCOUNT CODES FOR
 - *MESSAGE DETAIL RECORDING TO CUSTOMER PREMISES
 - *84 CUSTOMER SWITCHING SYSTEM FEATURES
 - *18 COIN AND CHARGE-A-CALL
 - *15 PUBLIC SAFETY

Expand
Collapse
Find word
Move cursor

Distribution of word(s) ofgg

Page of text

FEATURE SPECIFIC DOCUMENTS
82 PRIVATE FACILITY FEATURES
OUTGOING FACILITY GROUP QUEUING
1.8 INTRODUCTION
1.1 DEFINITION

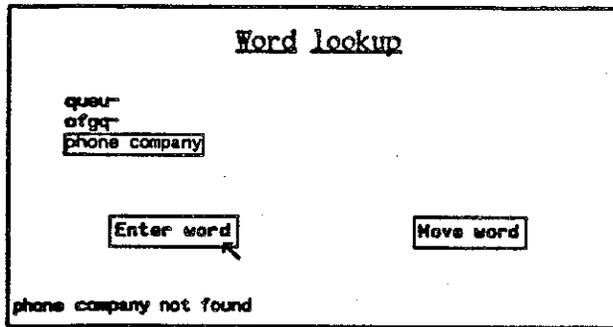
Outgoing Facility Group Queuing (OFGQ) is a per-outgoing-private-facility-group feature that improves the usage of the OFGQ customer's private-facility groups. The purpose of OFGQ is to allow a call to wait for a facility in one or more private-facility groups to become idle on a first-come, first-served basis (with optional priority treatment). Provided that certain conditions are met, OFGQ may be invoked by the switch after a caller is unsuccessful in finding an idle facility in an outgoing facility group. The search for the idle facility could have been initiated either by the caller dialing an access code for a particular outgoing private-facility group or by the caller using Automatic Flexible Routing (AFR). (Familiarity with AFR is assumed in this FSD.) The following capabilities are contained in OFGQ:

Find word
Find section
Page
Remove

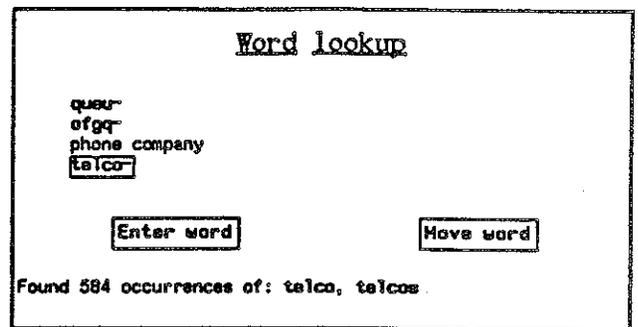
Looking for word(s): ofgg

Figure 7. Distribution of word OFGQ among sections in the Table of Contents and the first occurrence of OFGQ in the text.

To illustrate the adaptive index feature of SuperBook, let us suppose that the user wishes to search for places in the *LSSGR* that mention phone companies. In Figure 8 the user has entered the phrase *phone company* into the *Word Lookup* window, and SuperBook announced that it was not found. Later on, perhaps, the user will notice the term *TelCo* appearing in the text, clearly having Telephone Company as its meaning from its usage. The user may now search for *TelCo* to find all of its occurrences in the *LSSGR*. However, to avoid the frustration of another such failure, the user has decided to alias *phone company* with *TelCo*. This is done simply by placing the two phrases on the same line in the history of the *Word Lookup* window using the **Move word** button. When a word or phrase to be searched is placed on the same line as other words, SuperBook takes all of the phrases on the line as synonyms, searches for them all, and ORs together the resulting occurrence sets. In Figure 9 the user has aliased *phone company*, *TelCo*, and *telephone company* as well. Now, when the user (or someone else) enters *phone company*, SuperBook will come back with all occurrences of *TelCo* and *telephone company*.

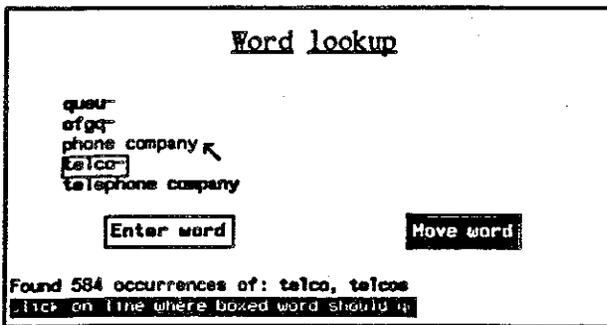


(a)

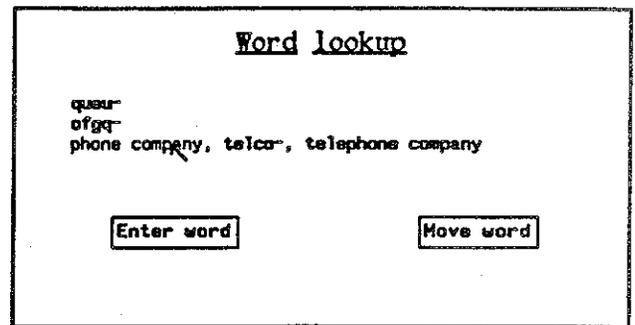


(b)

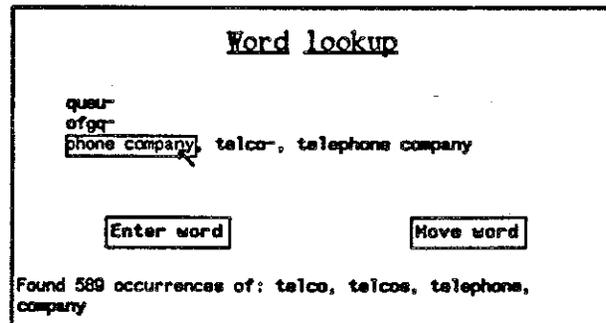
Figure 8. A search for phone company was unsuccessful, whereas Telco was found 584 times.



(a)



(b)



(c)

Figure 9. Defining synonyms in SuperBook: When words are moved to the same line in the word history, they will be treated as synonyms when doing a search. Now, when a user searches for phone company SuperBook looks up the synonyms telco and telephone company in the book.

Another customization feature of SuperBook is the facility for adding annotations to the text of the document. At any point in the text the user may add any annotation of his own, such as a reference, example, or explanation. This is done by clicking in the margin to the right of the text in the *Page of Text* window. A small editor window then appears on the screen, into which the user enters some text (Figure 10).

Page of text

FEATURE SPECIFIC DOCUMENTS
 02 PRIVATE FACILITY FEATURES
 MESSAGE DETAIL RECORDING TO CUSTOMER PREMISES
 3.0 FEATURE REQUIREMENTS
 3.1 FEATURE OPERATIONS

3.1 FEATURE OPERATIONS

3.1.1 MAIN FEATURE OPERATIONS

The switch should provide the standard originating and terminating treatment described in LSSGR Sections 5.2.1 and 5.2.2. In addition, the switch should collect and save MDR information during call processing. The switch should have the capability to collect MDR information during the call situations described in Sections 3.1.1.1 and 3.1.1.2 and should collect and save the MDR information described in Section 3.1.1.3 through 3.1.1.9. Call situations are divided into seven categories (I-VII) and the situations that fit into each category are described in Sections 3.1.1.1 and 3.1.1.2.

Find word Find section Page Remove

Looking for word(s): telco-, telephone company

ANNOTATION

See also Section 5.7.1

~

~

~

(a)

(b)

Figure 10. When the user clicks in the margin area to the right of the text a small editor window appears on the screen into which the user enters an annotation.

When finished, the editor window disappears, and an icon consisting of the user's login and the date appears in the margin of the *Page of Text* window, indicating the presence of the annotation (see Figure 11). This icon is scrolled along with the main text. To see the actual annotation one need only click on the icon causing the annotation text to appear in a separate pop-up window. Note how such notations can be used to create "hot" text, or to build new internal nodes, increasing the number of navigation routes.

Page of text

FEATURE SPECIFIC DOCUMENTS
 02 PRIVATE FACILITY FEATURES
 MESSAGE DETAIL RECORDING TO CUSTOMER PREMISES
 3.0 FEATURE REQUIREMENTS
 3.1 FEATURE OPERATIONS

3.1 FEATURE OPERATIONS

3.1.1 MAIN FEATURE OPERATIONS

The switch should provide the standard originating and terminating treatment described in LSSGR Sections 5.2.1 and 5.2.2. In addition, the switch should collect and save MDR information during call processing. The switch should have the capability to collect MDR information during the call situations described in Sections 3.1.1.1 and 3.1.1.2 and should collect and save the MDR information described in Section 3.1.1.3 through 3.1.1.9. Call situations are divided into seven categories (I-VII) and the situations that fit into each category are described in Sections 3.1.1.1 and 3.1.1.2.

tk1
7/24/87

Find word Find section Page Remove

Looking for word(s): telco-, telephone company

Figure 11. After leaving the editor an icon consisting of the user's initials and date remains, indicating the existence of an annotation.

Graphs and figures in the text will be displayed in windows of their own, which appear whenever references to them are made in the text, however, this feature is not fully implemented yet.

IMPLEMENTATION

SuperBook consists of a preprocessing phase and the browser itself. The preprocessor reads and analyzes on-line source text, written in some standard text markup language, and builds a formatted text data structure and full-text index. The hierarchical structure of the text, as indicated by the formatting macros of the markup language, is also represented in the data structure and index, and is used to construct the *Table of Contents*. SuperBook uses an interpretation module specific to the formatting package to properly format and structure the text. The preprocessing phase is fully automatic providing that the required interpretation module is given. Although the preprocessing may take several hours to run, it need be done only once for each document or major update.

To get SuperBook to accept a new text markup format, it is only necessary to write a translation program to convert the text to any of the "standard" formats that SuperBook already knows. The scope of documents available to SuperBook is quite large and growing, as interpretation modules are written making more formats recognizable. The number of existing on-line documents is large compared to the number of different text formats, and it is hoped that SuperBook will eventually be able accept already formatted text, without depending on structuring macros at all.

SuperBook at present is written mostly in Franz Lisp, which, while easy to modify and debug, runs rather slowly on a SUN. It would be possible to improve the speed by rewriting portions of it in C.

DISCUSSION

When SuperBook is considered in contrast to many other systems which are generally labeled hypertext we must ask the question - is SuperBook a hypertext system? Yes and No. Yes, at least in spirit, SuperBook is a hypertext system because it is a collection of tools explicitly designed to encourage the flexible exploration of ideas by making information more available. This goal was the original appeal of hypertext-like ideas e.g. [Bush45].

However, judged by its formal or architectural similarity to other recently developed systems, SuperBook may not be an example of a hypertext system. Those recently developed information browsing tools commonly designated as hypertext e.g. [Coll87], [Hala87], have data structures which directly support handcrafted author generated machine readable links between elements in an information space. In addition, these systems usually have sophisticated graphics tools and displays, to aid authors and

readers in the creation and modification of the structure connecting the information modules. But, are these features, which SuperBook lacks, necessary to accomplish the basic hypertext goal: a useful environment for the flexible exploration and creation of ideas? We think this is an open question.

Different approaches to the same problem. SuperBook attempts to accomplish the same ends as hypertext systems like ZOG [Robe81] and NoteCards [Hala87] by sophisticated key-word techniques applied to the full contents of a text, obtaining added intra-text relationships through annotation and adaptive aliasing, and by text navigation aids.

It is a clear goal of both SuperBook and other systems to give readers a tool to overcome the perceived limitations placed on them by the linear construction of standard books. Hypertext systems like those mentioned above do this by providing authors and reader with customizable spatial information structures, and the tools to traverse and construct them. We believe that this solution came about, at least in part, because these systems take as a starting point the preparation of new information sources and not, as SuperBook does, the recasting in electronic form of existing paper text and graphic material. Rather than developing new techniques to allow authors to build non-linear structure into a document, we attempted to develop ways to automatically expose the structure in a text and give users tools to explore it non-linearly.

It is important to point out that the usefulness of the information tools that result from either approach have not been evaluated to any significant degree. It is too early to tell if multi-window browsing tools of any underlying design will be valuable to people doing knowledge work. However, it worth noting that the key-word techniques used by SuperBook to increase the number of aliases to each information object have been shown to improve information retrieval performance [Furn83], [Gome]. It is also worth pointing out that McCracken and Akscyn [McCr84] report that ZOG data structures can cause the novice user to become lost and disoriented (see [Mant82] for a complete discussion). They also report that people rapidly learn to use ZOG and overcome early learning difficulties.

It is fairly clear that there is a wide range of applications for multi-window browsing systems. Some of those needs may be best met by systems whose underlying design resembles that seen in recent hypertext systems. New tools for information creation are an excellent example of such an application. On the other hand, systems like SuperBook are well suited to handling existing material and developing libraries of existing books which can be made available in one integrated format.

REFERENCES

- [Bush45] Bush, V., "As We May Think", *Atlantic Monthly*, no.176, pp. 101-108, Boston, 1945.
- [Coll87] Collier, G., "Thoth-II: Hypertext with explicit semantics". Submitted to Hyper-

text '87.

- [Furn83] Furnas, G.W., Landauer, T.K., Gomez, L.M., and Dumais, S.T., "Statistical semantics: Analysis of the potential performance of key-word information systems", *Bell System Technical Journal*, 62, 6, pp. 1753-1806, AT&T, New York, 1983.
- [Furn85] Furnas, G.W., "Experience with an adaptive indexing scheme", "Human Factors in Computer Systems", *CHI '85 Proceedings*, pp. 131-135, ACM, New York, 1985.
- [Furn86] Furnas, G.W., "Generalized Fisheye Views", in M. Mantei and P. Orbeton (eds.), "Human Factors in Computing Systems", *CHI '86 Proceedings*, pp. 16-23, ACM, New York, 1986.
- [Furn] Furnas, G.W., Landauer, T.K., Gomez, L.M., Dumais, S.T., "The Vocabulary Problem in Human-System Communication: an Analysis and a Solution", *CACM*, (in press).
- [Gome84] Gomez, L.M. and Lochbaum, C.C., "People can retrieve more objects with enriched key-word vocabularies. But is there a human performance cost?", in B. Shackel (Ed.) *Human-Computer Interaction - Interact '84*, pp. 257-261, North-Holland, Amsterdam, 1984.
- [Gome] Gomez, L. M., Lochbaum, C. C., Landauer, T. K., "All the Right Words: A Study of Human Performance and Key-Word Information Retrieval", (in preparation).
- [Hala87] Halasz, F., Moran, T., Trigg, R. "Notecards in a Nutshell", *CHI + GI '87 Proceedings*, ACM, New York, 1987.
- [Mant82] Mantei, M. M., "A study of disorientation behavior in ZOG", Ph.D. dissertation, University of Southern California, 1982.
- [McCr84] McCracken, D.L., Akscyn, R.M., "Experience with the ZOG human-computer interface system", *Int. J. Man-Machine Studies*, 21, pp. 293-310, Academic Press, New York, 1984.
- [Robe81] Robertson, McCracken, G.D., Newell, A., "The ZOG Approach to Man-Machine Communication", *Int. J. Man-Machine Studies*, vol. 14, pp. 461-488, Academic Press, New York, 1981.

User interface design for the Hyperties electronic encyclopedia

Ben Shneiderman

Department of Computer Science,
Human-Computer Interaction Laboratory, and
Institute for Advanced Computer Studies

University of Maryland, College Park, MD 20742

The Dream

Printed books were an enormous stimulus to science, culture, commerce, and entertainment. Electronic books and hypertext systems may produce a similar stimulus in the next century, but current designs are poor. Typical screens are too small, too slow, too complicated, and too hard to read. With careful attention to the user interface and the underlying technology, we have a chance to create a new medium that is potentially more attractive and effective than printed books in many situations.

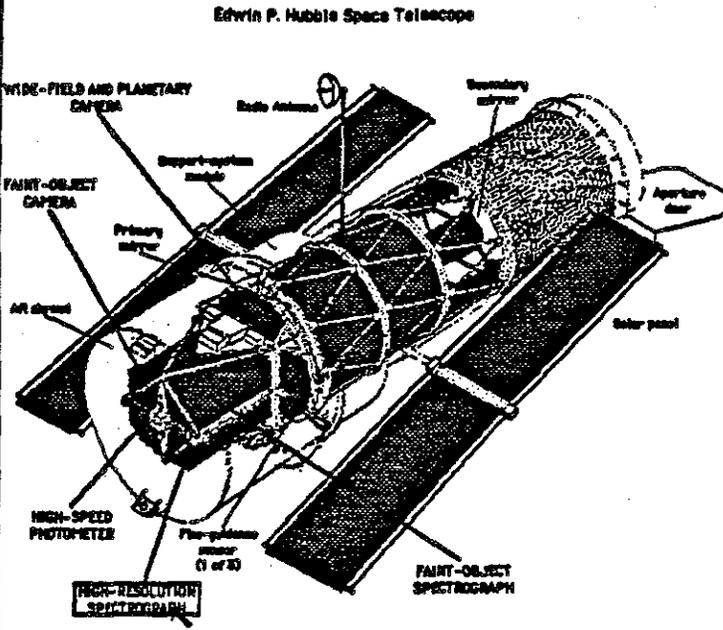
Electronic books can have color, animation, sound, rapid access, compactness, rapid traversal and search, user annotation, electronic dissemination and updating, dynamic text to reflect the user's needs, and other features yet undreamed of.

The Past

Hyperties has been under development at the University of Maryland since Fall 1983. It was originally called TIES (The Interactive Encyclopedia System), but the new name was chosen to indicate the close relationship with hypertext concepts. Hyperties allows users to explore textual and graphic information resources in an easy and appealing manner. They merely touch (with their fingers or with a mouse or by pressing arrow keys to move a light bar onto) highlighted topics or objects that interest them and a brief definition appears at the bottom of the screen. The users may continue reading or ask for the full entry about the selected topic. An article about a topic may be one or more screens long. As users traverse articles and graphics, Hyperties retains the path and allows easy reversal, building confidence and a sense of control. Users can also select articles and graphics from an index.

Hyperties is attractive for instruction (and entertainment) because the author's ideas and writing style are the focus of attention. Through careful human factors design, the computer aspects have been trimmed to let the author communicate to the students and to allow the students to control their learning. Hyperties has a defined instructional strategy - articles and graphics linked by embedded menus - that simplifies design, permits strong support for authoring, and enables use of previously written materials.

Hyperties can be an addition to a museum exhibit, a browser for information in libraries, a tool for diagnostic problem solving, an environment for novels or mysteries, an online help strategy, a browser for computer program text, a format for cookbooks or self-help manuals, or a way to explore cross referenced materials such as legal documents or an annotated Bible.



Imagine trying to see the clouds from the bottom of a muddy pond. That is how astronomers describe their view of the stars and planets through the Earth's atmosphere. As advanced as astronomical technology has become, our capabilities will be forever limited by the turbulence and brightness of our atmosphere. Even the finest ground observatories, such as the one at Mt. Palomar, California, are restricted by these conditions. In addition, the selective absorption of the atmosphere, which lets in visible light and radio waves emitted by stars and planets, but excludes most other forms of energy, limits our knowledge of celestial bodies.

To open the universe to observation in infrared, ultraviolet, x-ray, gamma-ray, and cosmic ray energies, NASA launched numerous satellites, each helping to explain different processes behind astronomical phenomena. But, to date, the value of these orbiting observatories has been limited by their relatively small size and limited spectral capability.

Now, for the first time, a ground-sized observatory will be placed in orbit to view the universe in visible and ultraviolet light unobscured by Earth's atmosphere.

Called the Edwin P. Hubble Space Telescope, the new observatory is a NASA-wide and international cooperative effort. Its name honors Edwin P. Hubble (1889-1953), who discovered that the universe extends far beyond the Milky Way galaxy.

The Hubble Space Telescope will weigh about 25,888 pounds (11,388kg) and will have a length of 43 feet (13.1 m) and a diameter of 14 feet (4.28 m). Its major components are an optical telescope assembly, five scientific instruments, and a support systems module.

Austria(see map) holds a special place in the history of the Holocaust. Situated between Eastern and Western Europe, possessing a vibrant and culturally creative Jewish community on the eve of World War II, Austria had also provided the young Adolf Hitler, himself an Austrian raised near Linz, with important lessons in the political uses of antisemitism. Leading Nazis came from Austria: the names of Adolf Hitler, Adolf Eichmann, who organized the deportations of the Jews to the death camps, and Ernst Kaltenbrunner, the head of the Reich Main Office for Security, 1943-45, readily come to mind. As

Linz - city in northern Austria; childhood home of Adolf Hitler and other leading Nazis

Above: SUN 3 version shows two pages with text and graphics items selectable by mouse to obtain more information.

Left: PC version enables touchscreen or arrow key selection of highlighted items (embedded menus) in text only, but graphics can be shown.

Hyperties

Hyperties consists of:

- Browser
- Authoring tool for composing new articles and editing
- Databases for browsing.

Databases already created include:

- Austria and the Holocaust (106 articles)
- Adele Stamp Student Union (42 articles)
- Hyperties Tour (68 articles)
- FULCRUM Project (30 articles)
- Online Maintenance Manual (52 articles)
- Managing your Credit (23 articles)
- David Seymour (1911-1956): Photographer (30 articles)
- Wines (40 articles)
- UM Office of Minority Student Education (41 articles)
- Introduction to the Computer Science Department (76 articles)
- Introduction to Online Database Searching at the National Agricultural Library (178 articles)

Hyperties is appealing to authors because of the explicit instructional model, the reduction of computer-related concepts, the focus on content, and the lively user interface. It allows authors to create a network of conceptual knowledge in which concepts are linked associatively and the learner is free to explore pathways based on their needs and interests. There is a great sense of satisfaction in composing articles and seeing the linkages come to life as they are used by students in novel ways.

The Hyperties authoring software guides the author in writing a title, brief definition (5-35 words), text (50-1000 words, typically), and synonyms for each article title. The author marks references in the text by surrounding them with a pair of tildes. Hyperties collects all references, prompts the user for synonym relationships, maintains a list of articles, and allows editing, addition, and deletion of articles. The author tool displays TO/FROM citations for each article and allows authors to keep notes on each article. A simple word processor is embedded in the authoring software, but users can create articles on their own word processor, if they wish. There are no commands to memorize, operations are done by selection from options on the screen.

Hyperties runs on a standard IBM PC (monochrome or color) and on IBM PCs, XTs, or ATs with or without touchscreens. We are attracted to the possibility of eliminating the keyboard while still providing substantial exploratory power. Hyperties was first written in APL and has been rewritten in the C programming language. Dan Ostroff, a graduate student in computer science, did the implementation and a major portion of the user interface design. Janis Morariu and Charles Kreitzberg contributed substantially to the design. Jacob Lifshitz, Susan Flynn, Richard Potter, William Weiland, and Catherine Plaisant-Schwenn did major maintenance and enhancements during 1985-87. The SUN 3 version with multiple windows and touchable graphics (graphic embedded menus) was implemented by Jacob Lifshitz and William Weiland.

Empirical Studies

More than a dozen experimental studies have been conducted to test out design alternatives and observe user behavior. Over 400 subjects participated in these controlled experiments. In addition, more than three hundred novices and experts have tried and commented informally on the current design.

In the study comparing the arrow keys (maybe better termed "jump" keys because the cursor would jump to the closest target in the direction pressed) to the mouse, the arrow-jump keys proved to be an average of 15% faster and preferred by almost 90% of the subjects. We conjecture that when there are a small number

of targets on the screen and when arrow-jump keys can be implemented, they provide a rapid, predictable, and appealing mechanism for selection.

In an early study conducted by David Powell, subjects traversed a database with 42 articles about the University of Maryland Student Union. The embedded menus technique reduced the number of screens viewed when compared with an explicit menu strategy. There were significant reductions in the times for task performance, and the subjective preference was strongly for the embedded menus.

The embedded menus idea was also used for two experiments with online maintenance manuals for electronic equipment. A tree structured and linear form of a 52 page maintenance manual was prepared for screen presentation and in paper form. Experimental subjects had to perform 12 tasks using one of the manuals. Significant differences were found showing that time was reduced using the paper versions. No significant differences were found between the tree and linear versions for speed or error rates. When a pruning algorithm was applied to the text to allow users to trim text unrelated to their task, the time was cut in half. This latter experiment used only the computer condition and demonstrated one of the advantages of screens over printed text. This is important, since for many applications printed manuals are still easier to use and approximately 30% faster to read than computer displays.

Four selection strategies (mouse, arrow-jump, touchscreen, and numbered keys) were compared in a study in which each of 24 subjects used each strategy to follow prescribed paths through the database. The touchscreen resulted in the fastest time and highest preference, but also the highest error rate. Improved strategies for touchscreen and the mouse were proposed.

Two studies were conducted to better understand how novice users traverse a database when seeking to answer historical fact questions. Judin Sukri compared a paper version of the Austria database (about 150 pages) with the Hyperties version. For simple fact queries paper was faster but Hyperties matched paper with more complex queries that required information from several articles. Subjects strongly preferred Hyperties even though it may have taken more time. Steve Versteeg compared performance with Hyperties versions showing 9, 18, and 34 lines of text (all single spaced) on the Enhanced Graphics Adapter display. The dense 34 line version was least preferred and performance times did not significantly differ across versions. Anecdotal evidence from both these studies helped shape our understanding of how people use Hyperties.

A study with 24 subjects compared the speed, accuracy, and user satisfaction of three touchscreen strategies designed for Hyperties. The "lift-off" strategy that enabled users to drag a cursor just above their fingers, produced statistically significantly lower error rates than the traditional "land-on" strategy. The lift-off strategy is implemented on the Microtouch touchscreen. We also have implemented Hyperties to run on the IBM Infowindows system.

On May 21, 1986 a version of Hyperties was installed in downtown Washington, DC in conjunction with an exhibit of the photographs of David Seymour. On November 14, 1986 a similar version of Hyperties was installed in the International Center of Photography at 94th Street and Fifth Avenue in New York City with another exhibit of David Seymour's photos. We collected usage data to understand reading patterns for patrons of these exhibits and are currently preparing a report. Index usage was much heavier than expected.

Studies currently being carried out focus on user strategies of using the index versus using the embedded menus, the advantages of four large versus forty-five short articles, the merits of three color coding styles, the impact of varying margin sizes and vertical spacing, and the impact of larger windows. Dependent variables include time to find answers to factual questions, comprehension scores on questions presented after reading, and subjective satisfaction.

Current Directions

In October 1987 we completed version 2.3 with revisions to many messages, repairs to some bugs, and many small improvements to the author tool. It also enables users to set their own colors for text

highlights and the cursor. Current design efforts focus on importation of large databases, browsing of computer programs, videodisc usage, and integration with other software. We have arranged for input from an IBM scanner in the proper format to support CGA and EGA graphics on the IBM PC. Bit-mapped display images for the SUN 3 workstation are generated on a Macintosh with Thunderscanner.

An advanced browser with string search, various bookmarks, multiple windows, user annotation, printing, etc. is being designed and parts are being implemented. An advanced authoring tool with automatic reference marking and sophisticated editing/formatting is planned.

The SUN 3 versions support our exploration of highlighting techniques for touchable graphics. In the current version the selectable regions pop up as the mouse cursor passes over them. This is a good start and makes for a dramatic experience, but some mechanism seems necessary to enable users to reveal all selectable regions without a great deal of mouse exploration. Possibly a single key or a on-screen button would cause all selectable regions to become highlighted. Highlighting might be done by blinking, color coding, hash marks, or various borders.

The multiple window strategies include a traditional independent arrangement of windows and a simple panning motion in which selections are always made in one window and older pages migrate to other windows in an orderly predictable manner. A more flexible strategy is to enable users to indicate which window is to receive the next page of text or graphics. Then backups are done by each window independently. An empirical evaluation of the benefits and problems is underway.

Availability

The University of Maryland has contracted with a software development firm to support commercial applications and distribution of Hyperties. The IBM PC version of Hyperties 2.3 is available, but the SUN 3 version is still in the research and development phase. To find out more, contact:

Charles Kreitzberg, President (609) 799-5005
Cognetics Corporation
55 Princeton-Hightstown Road
Princeton Junction, NJ 08550

Reports

- 1) Ewing, John, Mehrabanzad, Simin, Sheck, Scott, Ostroff, Dan, and Shneiderman, Ben, An experimental comparison of a mouse and arrow-jump keys for an interactive encyclopedia, *International Journal of Man-Machine Studies* 24, 1, (January 1986), 29-45.
- 2) Koved, Larry and Shneiderman, Ben, Embedded menus: Selecting items in context, *Communications of the ACM* 29, 4, (April 1986), 312-318.
- 3) Koved, Larry, Restructuring textual information for online retrieval, Unpublished Masters Thesis, Department of Computer Science, University of Maryland Technical Report 1529 (CAR-TR-133), (July 1985).
- 4) Lifshitz, Kobi and Shneiderman, Ben, Window control strategies for online text traversal, Unpublished Technical Report, Department of Computer Science, University of Maryland, College Park, MD 20742 (July 1987).
- 5) Marchionini, Gary and Shneiderman, Ben, Finding facts and browsing knowledge in hypertext systems, *IEEE Computer*, (January 1988) (to appear).

- 6) Morariu, Janis and Shneiderman, Ben, Design and Research on The Interactive Encyclopedia System (TIES), *Proc. 29th Conference of the Association for the Development of Computer Based Instructional Systems*, (November 1986), 19-21.
- 7) Ostroff, Daniel M., Selection Systems: Interactive Devices and Strategies, Unpublished Masters Thesis, Department of Computer Science, University of Maryland, College Park, MD, (May 1986), 161 pages.
- 8) Ostroff, Daniel M. and Shneiderman, Ben, Selection devices for users of an electronic encyclopedia: An empirical comparison of four possibilities, University of Maryland Computer Science Center Technical Report 1910, (September 1987).
- 9) Potter, Richard L., Weldon, Linda J., and Shneiderman, Ben, Making touch screens effective: An experimental evaluation of three strategies, University of Maryland Computer Science Center Technical Report 1920, (September 1987).
- 10) Shneiderman, Ben, User interface design and evaluation for an electronic encyclopedia, *Proc. of the 2nd International Conference on Human-Computer Interaction*, Honolulu, HI, August 1987. In G. Salvendy, Ed., *Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems*, Elsevier Publishers, Amsterdam, (1987), 207-223.
- 11) Weldon, L. J., Mills, C. B., Koved, L., and Shneiderman, B., The structure of information in online and paper technical manuals, *Proc. Human Factors Society - 29th Annual Conference*, Santa Monica, CA, (1985), 1110-1113.

A Hypertext Writing Environment and its Cognitive Basis*

John B. Smith, Stephen F. Weiss, & Gordon J. Ferguson

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
919-962-1792

Abstract

WE is a hypertext writing environment that can be used to create both electronic and printed documents. It is intended for professionals who work within a computer network of professional workstations. Since writing is a complex mental activity that uses many different kinds of thinking, WE was designed in accord with an explicit cognitive model for writing. That model raises several important questions for both electronic and printed documents. The paper includes a discussion of the underlying cognitive model, a description of WE as it currently exists and as it will be extended in the near future, as well as a brief outline of experiments being conducted to evaluate both the model and the system. It concludes by re-examining some of the issues raised by the cognitive model in light of WE, especially the role of constraints in hypertext systems.

1. Introduction

Hypertext is a form of electronic document in which data is stored as a network of nodes connected by links. Nodes can contain text, source code, graphics, audio, video, or other forms of data. Hypertext documents are normally meant to be written, stored, retrieved, and read within a computing environment. Thus, they spend their entire life on-line rather than on paper.

We are building a system that differs from most hypertext systems. It regards the network or directed graph form of information as one (early) stage in the development of a document rather than as its final form. Our system, which we call the Writing Environment or WE for short, helps writers transform loose associative networks of ideas into a hierarchical structure and then write a document in accord with that structure. The product that results can remain in electronic form but it can also be printed out to produce a paper document. Thus, the system can

* Parts of this research were sponsored by The National Science Foundation (Grant IRI-8519517), The Army Research Institute (Contract MDA 903-86-C-0345), and The International Business Machines Corporation (SUR Project 423).

be used both as a conventional hypertext system and as an authoring system with advanced graphical, direct manipulation structure editing capabilities.

Supporting both electronic and conventional paper forms of documents is a key aspect of WE. While electronic documents may eventually replace paper ones, that day is not at hand. Even in organizations in which professionals work within a network of workstations, paper documents continue to be important. Many users prefer to edit on paper rather than on screen. Most internal documents – memoranda, proposals, reports, etc. – must be printed for upper management to read them. And most documents that go outside the organization still go out through the mails or, more likely, Federal Express, than through a network. Thus, in building a system that supports both electronic and printed forms of documents, we have attempted to provide the best of both worlds.

A second major concern of our research group is the relation between WE and the cognitive processes of its users. We are particularly interested in the cognitive strategies writers use to transform information in one form into another. Consequently, a second line of research we are carrying out is a series of experimental studies to first map and then differentiate between the strategies used by expert vs. novice writers and those that lead to effective vs. ineffective documents.

This interest in the relation between cognitive process and system functions is shared with a number of hypertext developers. From the beginning, those working on hypertext systems and concepts have been keenly interested in the relation between thinking and computing. Vannevar Bush called his theoretical system *memex* and saw it as an environment that would enhance the thinking of knowledge workers [Bush, 1945]. Doug Engelbart called the first actual hypertext system *The Augmented Human Intellect System* [Engelbart, 1968]. Another, more recent system is called *Knowledge Garden* [Thompson & Thompson, 1987].

While the hypertext systems that are emerging offer many new opportunities for structuring and using information, they also raise a number of new questions concerning how best to create and use those resources. Much of the work we have done to understand and model the cognitive processes of writers applies equally well to the authors and users of hypertext “documents”. Hypertext authors must still transform inchoate ideas into coherent structures that can be comprehended as well as traversed. Users of hypertext documents must still understand what they read (or see, or hear, . . .) and must construct relations between new information and old, one idea and another. Thus, a second part of our discussion will be a consideration of the cognitive processes that underlie WE and that apply to other hypertext systems, as well.

In describing our work, we will look first at the cognitive basis for WE that includes a Cognitive Framework for written communication. In doing so, we will point out issues that have long-term implications for hypertext systems. Next, we describe WE. Following a brief discussion of some of the experimental studies we

are conducting to evaluate both the cognitive model and the system, our discussion concludes by reconsidering several of the questions raised by the Cognitive Framework in light of the description of WE.

2. Cognitive Basis for WE

In this section, we discuss the cognitive processes involved in writing. We describe those processes, first, as they are used for conventional paper documents and, then, for electronic or hypertext documents.

2.1. Cognitive Modes

Writing is a complex process that draws on many different cognitive skills. Not just translating ideas into words but retrieving information from the writer's memory or from external sources, identifying associative relations among ideas, drawing inferences and making deductions, building larger hierarchical structures, as well as reading, analyzing, and rewording during the editing process. Some writers even report using visual and kinesthetic thinking.

We view these processes as constituents of a set of *cognitive modes*. A mode consists of three components: one or more cognitive processes, a product produced and/or operated on by those processes, and a set of rules that govern the kinds of products that can be produced within the mode and the relations that can exist among the parts of the product(s). Writers use different cognitive modes to produce different forms of information or to transform one intermediate product into another.

For an intuitive sense of modes, consider the following examples. During early exploratory thinking, many writers adopt a mode of thinking in which the primary purpose is to identify ideas and data that may be included in the document and to consider various relations among them. The tenor of exploratory thinking is often relaxed and creative as the individual generates and considers alternative possibilities for the document. However, the mode of thinking used for organizing the content of the document is different. As the writer shifts to building a single integrated structure, he or she is likely to work with more focused attention and a stronger sense of purpose. Writing, *per se*, and editing involve still other modes of thinking.

2.2. Cognitive Framework for written communication

Figure 1 shows the flow of information through the different cognitive modes used for conventional written communication. The model includes both reading and writing. The areas of the "hourglass" denote different cognitive modes. The modes are shown across the top of the figure, the products along the bottom; the tapered areas of the hourglass form, itself, indicate relative differences in the constraints



0 0 0 0 0

Invited Panel on Systems

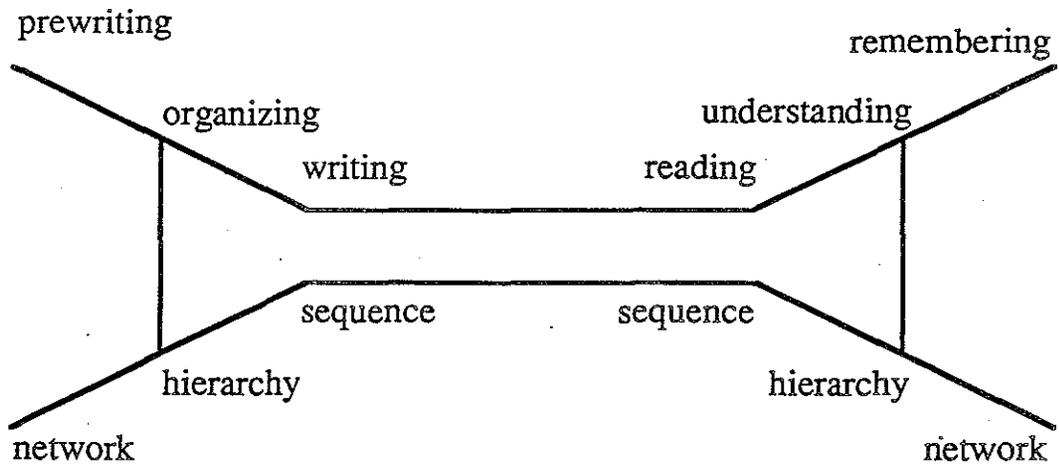


Figure 1

Cognitive Framework for Written Communication

imposed by the different rules for the various modes. (A smaller area implies more constrained options; a larger area, more relational possibilities.) We can now define more precisely the processes, products, and rules for the major modes used for writing.

During the early exploration phase of writing, represented on the left of Figure 1, the writer retrieves potential content from long-term memory or from external sources, considers possible relations among ideas, and, perhaps, groups related ideas and constructs small hierarchical structures. In that mode, the underlying rules are those associated with a network: any idea can be related to any other idea through simple association. Thus, the intermediate product is a network or directed graph of ideas.

Organization is the task of constructing a integrated structure for the document. For many documents, particularly those written by professionals, that structure will be a hierarchy. The product is, thus, a hierarchical structure, and the rules are those that govern hierarchies. That is, each concept or node in the hierarchy can be subordinate to at most one other concept/node, but it may be superordinate to many concepts/nodes. Building such a structure requires a different set of cognitive processes from those used during exploration. The critical one is the process of abstract construction that includes perceiving subordinate/superordinate relations, comparative levels of abstraction, sequencing, proportion, and balance. This mode is shown as smaller than that for exploration since a hierarchy is a restricted form of network and is, thus, more constrained.

Writing, itself, involves still a different set of cognitive processes. Here, the primary task is encoding the abstractions of content and the relations of the hierarchical structure into a sequence of words, drawings, or other explicit forms. The structure of the encoded text is linear and represents a path through the hierarchy. Consequently, it is even more constrained than organization mode.

Editing is not shown in the Figure, but would represent still further constraining of the linear sequence and would include additional reading and analysis, as well as encoding, processes.

Reading, shown on the right half of the Figure, employs an analogous set of processes and forms. Whether the reader reads the document from beginning to end or jumps around from place to place, when that reader settles to read, the text that is read or decoded is a linear sequence of words. The text that is comprehended, however, is a hierarchy. That is, the reader sees that several points do, indeed, add up to the conclusion drawn by the author or that a generalization is supported by the facts or argument cited [Kintsch, 1974; Meyer, 1975; Kintsch & van Dijk, 1978]. What is remembered, though, is that portion of the text hierarchy that is integrated into the network of long-term memory.

Thus, both writing and reading involve a series of transformations in which different cognitive processes transform information in one structural form into a

different structural form. For writing, that dynamic is network to hierarchy to linear sequence. For reading, the dynamic is reversed. (A more thorough discussion of cognitive modes can be found in [Smith and Lansman, 1987].)

2.3. Implication for printed documents

Central to both writing and reading is the hierarchical form of information. Perhaps that is not surprising since research in reading comprehension has shown scientifically what many writers have known intuitively: that hierarchy is an optimal form for most expository documents. Consequently, features that highlight a document's (hierarchical) structure increase its comprehensibility.

More specifically, thematic titles presented prior to a well-structured text significantly increase free recall of the content of that text [Schwartz & Flammer, 1981]. Within a text, advance organizers – passages containing the main concepts of a text or section of text but at a higher level of abstraction – positively affect comprehension [Ausubel, 1963]. Hierarchical texts in which the structure is signaled or cued are comprehended more effectively than texts in which the structure is not signaled [Meyer, Brandt, & Bluth, 1980]. At the paragraph level, inclusion of a topic- or theme-sentence in the initial position, rather than in an internal position or not at all, results in more accurate comprehension [Kieras, 1980; Williams, Taylor, & Ganger, 1981].

Consequently, for efficient, effective communication, a writing environment should support and encourage development of documents with these characteristics.

2.4. Implication for hypertext documents

This research has significant implications for developers of hypertext systems. Readers of hypertext documents are likely to have problems comprehending what they read similar to those of readers of conventional documents. The same features that facilitate comprehension there are also likely to apply to electronic documents – a well-defined structure that is clearly signaled, advance organizers such as overviews and descriptive titles, and topic statements within individual paragraphs or content units. All of these help the reader develop a high-level understanding of the document's content and purpose that serves as a framework in which to understand and attach its details.

The underlying model for most hypertext systems is a directed graph in which content units are associated with the nodes and the sequences in which the reader may access them determined by the links. However, a *network* of information has properties very different from those of a *hierarchy*. By definition, a hierarchy addresses a single, high-level concept or purpose. Thus, it is well-suited for writers who wish to argue a single point or produce a specific action by their document. A network has no such central thrust. Rather, it is an *environment* in which different

readers may immerse themselves for different purposes and with different expected results. Thus, the emphasis is on the experience of the reader rather than any specific motivation or action. We can easily imagine new forms of entertainment, new literary genres, or even bodies of research materials with directed graph structures. But we cannot foresee purposeful, action-oriented communications in this form. Hierarchical documents, on the other hand, provide the reader with a sense of the whole by including high-level overviews that describe what will follow. Structural information of this sort does not exist in a directed graph. Most network-based hypertext systems have ignored the issue of global structure. Instead, they simply provide for each unit of information the links in and the links out. And most readers quickly get lost in the tangle.

While visual tools for helping readers grasp large graph structures are promising, (see Figures 5 and 6, below), the issues of purpose and focus are inherent. In the section that follows, we will describe the WE system, pointing out as go along how it has addressed these issues. After that, we will return to these same concerns, suggesting a somewhat different perspective of hypertext that may help resolve some of these problems.

3. Description of WE

WE was designed to be congruent with the cognitive theory of writing outlined above. In describing the system, we will first discuss its multimodal design, then the function it provides for moving intermediate products from one system mode to another, and, finally, several special features including a zoom and roam function for searching a large graph structure and controlling the display, WE's interface with an underlying database, and print options. We will also describe WE's hypertext features and several extensions we plan to make to the system in the near future.

3.1. Modes

WE supports each of the major phases of writing in a separate window or system mode. The rules that underline each cognitive mode are reflected in the operations WE supports in the corresponding system mode. WE's structural modes, shown on the left of Figure 2, support representing information units as nodes, moving these nodes from one place to another, and defining relationships among them in the form of directed links. WE's encoding and editing modes, shown on the right side of the screen, only permit manipulation of the content (currently, text) associated with these nodes; structural operations are not allowed in these. A more detailed explanation of each of WE's modes follows.

The user interface provides direct manipulation of visual objects. Objects are selected by pointing with a mouse. Pressing a mouse button provides a pop-up menu specific to the type of object selected. Thus, user operations are organized around a taxonomy of visible object types.

Writing Environment emptyWS		Work Space		Holding Areas		System 10 September 1987		
NETWORK MODE: Net a			View Control	Display/Print	TEXT MODE		View Control	Display/Print
					X			
TREE MODE: Tree a			View Control	Display/Print	EDIT MODE		View Control	Display/Print

Figure 2

WE: Default Screen Layout

3.1.1. Network Mode

Network mode, shown in the upper left quadrant of Figure 2, is intended to support the early exploratory phases of document development. It is also the mode normally used for hypertext. The cognitive processes, described above, for the corresponding cognitive mode are retrieving potential concepts from long-term memory and/or from external sources, representing these concepts in tangible form, clustering them into related groups, defining specific relations or associations between pairs of concepts, and constructing small hierarchical structures.

The system functions for network mode were designed to support the corresponding cognitive processes as directly and as unobtrusively as possible. To represent a concept, the user may point anywhere in the visual space of WE's network mode and select *create node* from the menu. He or she is then prompted for a brief title to label the node. As the set of nodes/ideas grows, the user's cognitive orientation is likely to shift to building small clusters of ideas. The *move* option, selected from the node menu, can be used to gather concepts into spatial groups. To make relations between nodes explicit, the user may link them and give the link a title. The writer who thinks of links as indicating a super/subordinate relation may use this options to build small hierarchies. Figure 3 shows a network constructed by exploring the cognitive concepts related to WE.

As patterns of nodes emerge on the screen, they produce a similar change in the pattern of concepts in the writer's mind. At some point, the author is likely to shift from exploring the ideas and relations inherent in the data and his or her mind to constructing a single, integrated structure for the document. Thus, the writer's intention shifts from possibility to commitment. At this point, he or she may pause to tidy up the exploratory clusters in preparation for moving into a different mode of thinking and working.

3.1.2. Tree Mode

Tree mode helps the user build a single, integrated hierarchical structure for the document. Noting super - and subordinate relations as was done during exploration is frequently almost a reflexive cognitive process, but constructing a large integrated hierarchical structures is not. It requires additional processes. The writer must think on a broader scale, noting relations among not just small groups of concepts, but whole substructures of ideas. He or she must also note parallel relations among similar configurations as well as balance the overall structure. In short, organization is a process of conscious, deliberate construction.

WE represents the hierarchy as a tree, as seen in the lower left quadrant of Figure 2. Figure 4 shows an expanded tree mode. The constraints for tree mode are different from those of network mode. It is no longer possible to create isolated nodes; new nodes can be created only in relation to the tree. To add a node, the writer first selects a node in the tree. He or she can then add a 'child' (subordinate

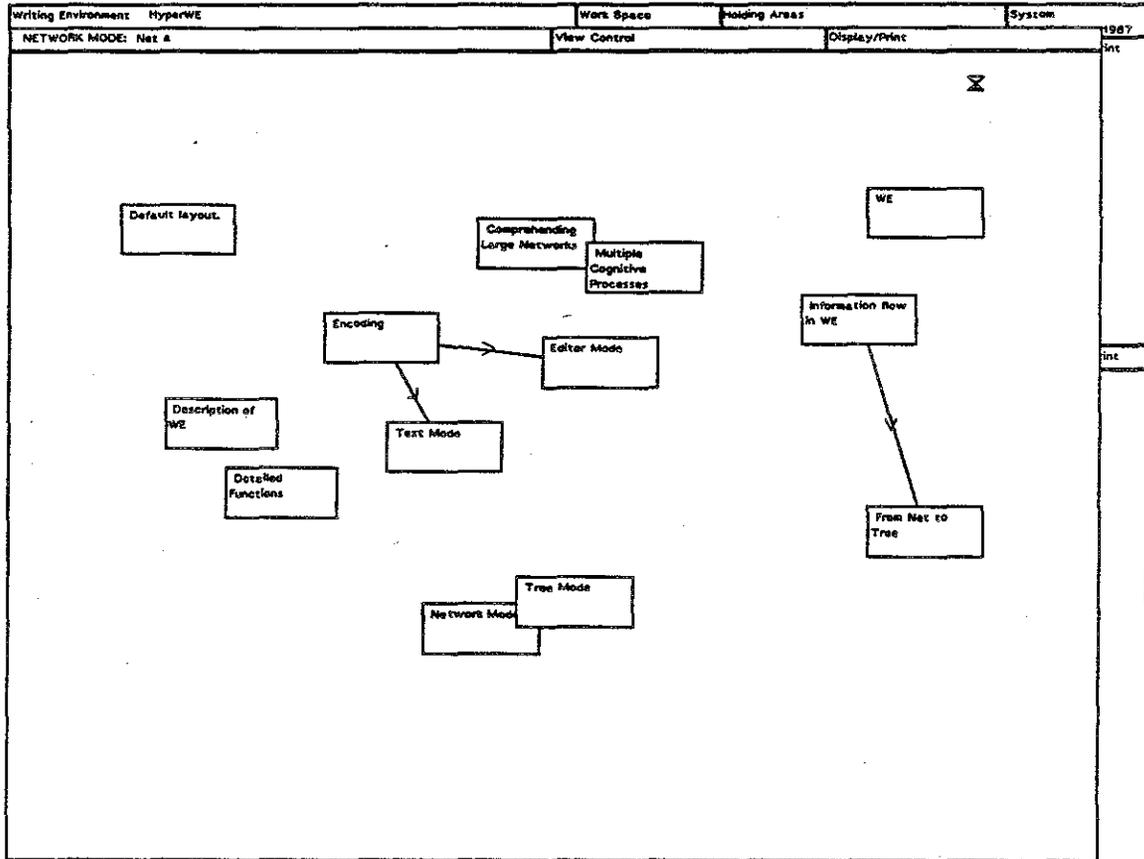


Figure 3
WE: Network Mode

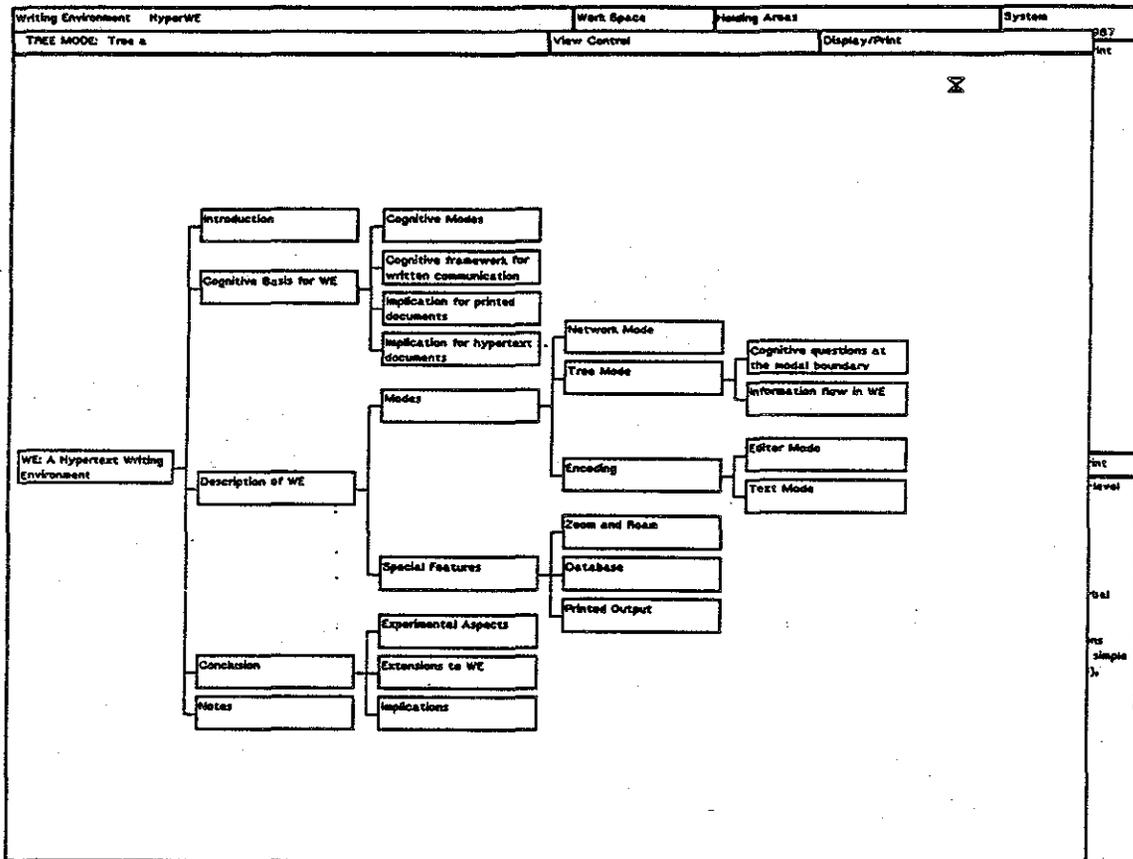


Figure 4
WE: Tree Mode

concept), a 'parent' (superordinate concept), or a 'sibling' (parallel concept). Nodes may be moved from one place to another in the tree. In fact, entire subtrees (a node and all its descendants) may be manipulated: moved, deleted, or made the focus of display. But neither nodes nor branches may be moved out of the tree and remain in tree mode, since the rules of this mode constrain the product to a single hierarchical structure, not a forest.

One of WE's strength is its support of information flow across the modal boundary between network and tree modes. Moving concepts is simply a matter of copying and pasting nodes. Both operations may be done to and from either tree or network mode simply by selecting the node to be moved in one mode and then pointing to the position where it is to be pasted in the other mode. Moving structures of nodes is done the same way. If the structure in Network Mode is hierarchical, the operation is straight forward. If it is not a hierarchy - e.g., a graph containing a cycle - WE transforms the graph into a hierarchy by applying a depth first algorithm that breaks links that cross the hierarchy.

A *subtract tree* operation in network mode provides a form of negative information flow. When a branch of the hierarchy is selected in tree mode, the subtract tree operation removes from the display in network mode all the nodes contained in the branch. Thus, only those ideas/nodes will remain displayed in network mode that have not yet been integrated into the document's hierarchical structure.

3.1.3.1. Editor Mode

Editor mode, shown in the lower right quadrant of Figure 2, provides access to a standard text editor. It is used to encode the concept represented by a node into text. In the current system, that editor is the Smalltalk text editor. In future extensions of WE, the system will support additional text editors as well as editors for other kinds of data, such as graphics, sound, and video. At that time, the editor invoked will be keyed to the data type of the particular node.

To begin writing, the user points to the node in either tree or network mode and selects the edit option on the menu. This transfers control to editor mode. Text may then be keyed in, deleted, and so on. The user leaves editor mode simply by moving the cursor from that area of the screen into any of the other mode windows.

3.1.4. Text Mode

In text mode, shown in the upper right quadrant of Figure 2, the document is presented in linear form much as it would appear in printed form. Text mode constructs a representation of the continuous document by stepping through the tree - from top to bottom, left to right - inserting node labels as section headings, followed by the blocks of text associated with the nodes. A scroll bar permits the writer to move forward and backwards through the document as a whole (the path through the tree). A long-term goal is to make the representation identical to final

formatted output. Currently, text mode provides three editing regions within its window. As the tree is traversed using the scroll bar, the blocks of text associated with the various nodes are moved into the three areas of the window.

Within each area, a second scroll bar permits the user to move through the text for the individual node displayed there. Thus, by scrolling to the bottom of one section and the top of the following section, the writer can see how the text for the two nodes fits together. The writer can edit the text for each node using the Smalltalk editor, just as in editor mode. Text can also be moved from one area/node to another, and the section headings (node label) can be edited, as well. However, the node itself can't be deleted or edited structurally from text mode. This can be done only from tree mode.

3.2. Special Features

WE provides several additional featuring that are not, strictly speaking, part of the writing process. These include a zoom and roam option for managing a group of nodes too large to fit on a single screen, an interface for a supporting database, and options for printed output.

3.2.1. Zoom and Roam

Navigation through the two dimensional space of computer displays has typically involved some form of scroll bar. Unfortunately, these do not present any overview of the space being explored. WE uses a different technique, called roaming, that was originally developed by other members of our research group [Beard & Walker, 1987]. The user can invoke the roam and zoom display from either network or tree modes: the system will then display in a pop-up window a very small representation of the entire graph or hierarchical space with the area of the current display indicated by a box, (see Figure 5). This box can be directly manipulated to change the scale or position of what is then displayed in the mode window. Figure 5 shows a stretch box and Figure 6 shows the resized network mode that was produced as a result.

3.2.2. Database

WE is intended to be used in conjunction with an object oriented database system in which all structural information is stored. To support this interface, WE uses low-level data objects that correspond with database operations. These objects are currently of three types: structures, nodes, and links. Structures are typed, named sets of links (and, by implication, associated nodes). The type indicates whether the structure is a graph, hierarchy, or path. This information is used by the system to determine the operations that can be performed on the particular structure. Each node is also viewed as a typed object. Associated with it are various attributes that identify the type of content "within" the node and, thus, bind it to

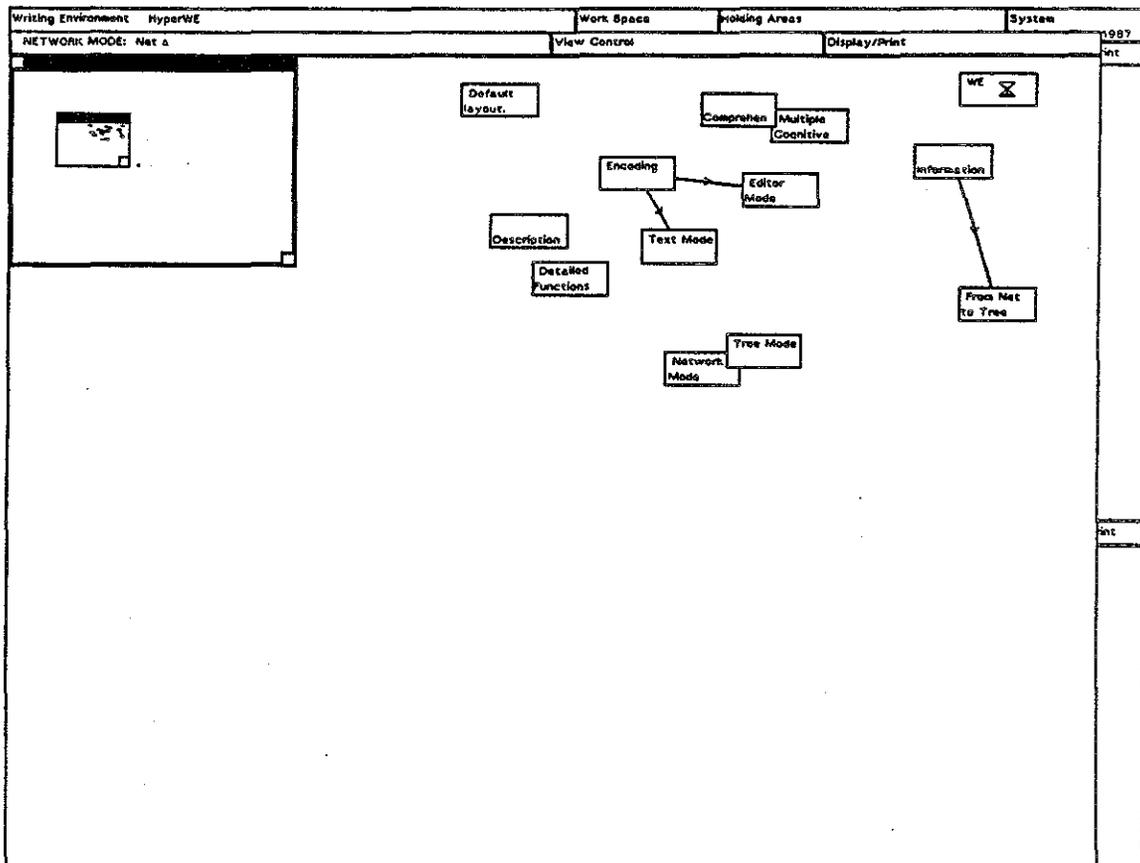


Figure 5

WE: Roam and Zoom View of Network Mode

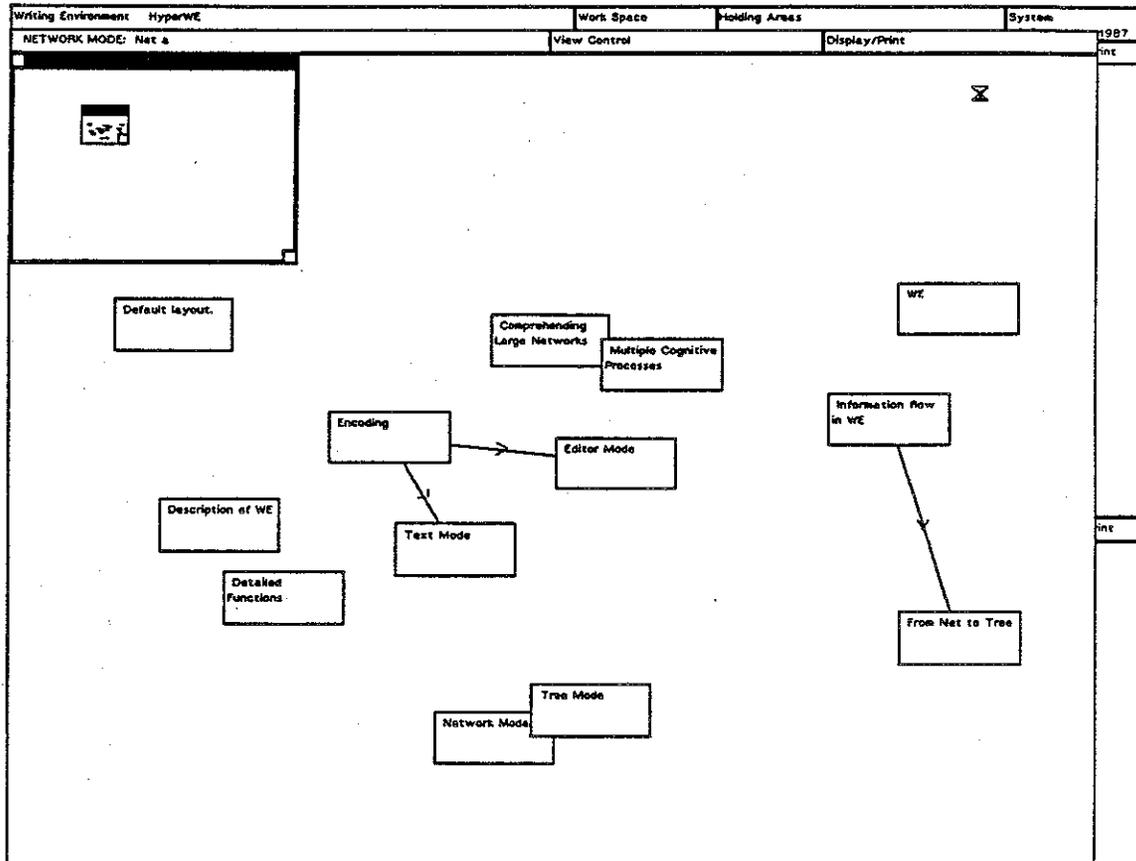


Figure 6

WE: Resized View of Network

a particular editor/display program; its spatial dimensions in network mode space; and both its associative and hierarchical links. Links are attributed pairs of node identifiers that define the directed arc. Attributes indicate the structure of which the link is a part and additional system information.

Currently, the database is confined to a single document, but we will extend its definition to permit teams and departments to store collections of documents and other kinds of data. Thus, future users will be able to search the database for information relevant to a current project. Once a usable node or structure is found, it can be imported into the environment and included in the structure being developed.

3.2.3. Printed Output

WE produces output for a laser printer, although actual formatting is done by TeX using commands inserted into the text by WE. The detailed mechanics of the printing process are, of course, installation dependent. However, two levels of intervention are available to users. First, the TeX files, themselves, can be saved and modified as necessary. Some sections of this document were prepared using WE, others with a conventional editor; the two groups were integrated in this way. Second, the TeX macros that format the headings, select the fonts and spacing between sections, and so on, are stored in a dictionary and can be changed by the users.

3.3. Extensions to WE

WE is an evolutionary system. We will continue to enhance it, providing additional functions and additional capability with respect to the size and number of documents. High on the list of priorities are functions to help users manage multiple drafts of single documents. Most of these extensions will be implemented in terms of the database design when WE is merged with an underlying object-oriented database system.

A more fundamental enhancement will allow WE to handle distributed writing. There are many (perhaps most) projects that are too big for a single person. The next major step in the development of the Writing Environment will be to support collaborative writing. A group of writers, possibly widely separated geographically, will be able to work together to produce a single product. Each user will be able to see a single shared workspace and will be able to manipulate the workspace under a managed collaborative paradigm.

A longer term goal is to merge another system we are developing – MICROAR-RAS [Smith, Weiss, & Ferguson, 1986], an advanced full-text retrieval and analysis system – with WE. This will provide fast, flexible content-based searches as well as other analytic functions.

While we have chosen to characterize the system in terms of writing, it is actually a more general tool. It is useful in many other information management applications where the major steps are:

- conceptualization – the generating of ideas;
- organization – imposing structure on those ideas;
- modification – refining the ideas and structures;
- and linearization – defining linear paths through the structure.

Such applications include designing a building, planning a logistical operation, or writing a large computer program. In each case, the process begins with the creation of a graph structure or hypertext of content units. The nodes represent the individual components of the operation, and the links represent the dependencies. Implementing the operations requires that the hypertext be linearized, for example, along a single time line. WE will provide tools to develop additional modes tailored to particular applications.

4. Experimental Studies

4.1. Protocol Tracker and Cognitive Grammar

In addition to serving as a tool for writing, WE can also be used to observe how people write. We have implemented an on-line tracker that captures a user's interactions with the system. That information is represented as a sequence of symbols, with attributes, that constitutes a history of the session. We have also built a replay function that permits us to replay the session in time proportional to the original session, in uniform time, and in manually controlled steps.

We are developing a more powerful tool for analyzing these data. To be comprehensible, the low level data must be transformed into symbols that are more general and more indicative of the user's strategy. That is, users think in terms of high level conceptual phrases but they enact those phrases as a sequence of lower level operations. For example, the conceptual phrase might be to create a cluster of ideas. This is accomplished in WE by creating a set of nodes, labeling them appropriately, moving them near one another, and, perhaps, linking them together. We are developing a cognitive grammar by which low level operations can be mapped to a small set of conceptual phrases and higher level constructs. The parse trees produced by the grammar will provide insight into the user's overall writing strategy. By comparing the strategies of different classes of users (for example, expert technical writers vs. novice writers), we hope to develop more effective and efficient writing methodologies and tools.

5. Conclusion

We conclude by returning to some of the questions and issues raised at the beginning of this paper. While the processes of reading and writing conventional documents have been studied in considerable detail, we still have a very limited understanding of the cognitive processes and strategies that produce effective information transfer. We know even less about such communications for electronic documents. A major line of research that should go hand in hand with the development of hypertext and other electronic document systems is formal, controlled experimental studies of users' interactions with these systems followed by actual-use studies to confirm results. We are committed to this approach as an integral part of the development method for WE; we know of at least one other research group (Xerox PARC) that, we believe, shares this concern. But this is a large and complex area of inquiry that will require additional researchers as well.

In many respects, hypertext is a state of mind. It has been described frequently as a tool to enhance the user's mental abilities, as an environment in which to think, etc. It is essential, however, to remember that human beings don't exist in only one state of mind. We use multiple cognitive modes for different intellectual tasks and purposes. (Figure 1 showed the organization of those modes for written communication.) But, as suggested above, hypertext in its fundamental form – a directed graph of information components – is most consistent with one particular mode of thinking – exploration. Exploratory thinking usually occurs early in the development of a set of ideas. Such thinking is an integral part of the overall cognitive process not just for writing but for many forms of productive, professional work. But it is an end in itself for only certain situations. Great for an aesthetic experience – James Joyce, or more likely, John Fowles, would have loved it as a literary medium. Great for an undirected, free-flowing learning experience, analogous to spending an evening browsing through an encyclopedia. But as a tool for professionals, hypertext, we believe, will become a supporting utility over which more constrained applications will be developed rather than the primary application system, itself. To be truly effective, hypertext applications must match additional power with additional control and structure. In the long term, constraints may turn out to be more important than raw power.

Looking further into the future to a time when large distributed databases of hypertext documents will exist, we don't see (or don't want to see) a flat, hyperplane of spaghetti. Rather, we believe that out of that hyperplane will emerge peaks of understanding and purpose created by professionals using powerful new tools. These peaks will be criss-crossed, to be sure, by multiple paths and relations, sometimes visible, sometimes not. But each peak will be supported by a single, integral hierarchical structure.

We share the enthusiasm for hypertext that is growing daily. But we hope that trail-blazers will think about *where* they are going in addition to *how* to get there. And that those that follow them will do so for purpose as well as for possibility.

6. Acknowledgments

A number of individuals and organizations have contributed to the work described here. We wish to thank our sponsors for both their financial support and the advice and encouragement provided by their program officers. These include The National Science Foundation, The Army Research Institute, and The IBM Corporation. We also wish to thank our faculty colleagues, Profs. Marcy Lansman (Psychology) and Jay Bolter (Classics), for their contributions to the ideas described here. We also wish to thank the following graduate students who have helped to develop WE: Paulette Bush, Yen-Ping Shan, Irene Jenkins (Psychology), Valerie Kierulf, and Greg Berg (Psychology).

7. Notes

Ausubel, D. P. (1963). *The Psychology of Meaningful Verbal Learning*. New York: Grune & Stratton.

Beard, D. V. & Walker, J. Q. (1987). *Navigational techniques to improve the display of large two-dimensional spaces*. Chapel Hill, NC: UNC Department of Computer Science Technical Report 87-031.

Bush, V. (1945). As we may think. *Atlantic Monthly*, 176(1), 101-108.

Engelbart, D. & English, W. (1968). A research center for augmenting human intellect. *Proceedings of 1968 FJCC*. Montvale, NJ: AFIPS Press, pp. 395-410.

Kieras, D. E. (1980). Initial mention as a signal to thematic content in technical passages. *Memory and Cognition*, 8(4), 345-353.

Kintsch, W. (1974). *The representation of meaning in memory*. Hillsdale, NJ: Erlbaum Associates.

Kintsch, W. & van Dijk, T. A. (1978). Toward a model of text comprehension and production. *Psychological Review*, 85, 363-394.

Meyer, G. J. F. (1975). *The organization of prose and its effects on memory*. Amsterdam: North Holland Publishing Company.

Meyer, G. J. F., Brandt, D. M., & Bluth, G. J. (1980). Use of top-level structure in text: key for reading comprehension of ninth grade students. *Reading Research Quarterly*, 1, 72-103.

Schwartz, M. N. K. & Flammer, A. (1981). Text structure and title-effects on comprehension and recall. *Journal of Verbal Learning and Verbal Behavior*, 20, 61-66.

Smith, J. B. & Lansman, M. (1987). *A theoretical basis for a computer writing environment*. Chapel Hill, NC: UNC Department of Computer Science Technical Report 87-032.

Smith, J. B., Weiss, S. F., & Ferguson, G. J. (1986). *MICROARRAS: An overview*. Chapel Hill, NC: UNC Department of Computer Science Technical Report 86-017.

Thompson, B. & Thompson, B. (1987). KnowledgePro. Software distributed by Knowledge Garden, Nassau, NY.

Williams, J. P., Taylor, M. B., & Ganger, S. (1981). Text variations at the level of the individual sentence and the comprehension of simple expository paragraphs. *Journal of Educational Psychology*, 73(6), 851-865.

Argumentation

Constraint-Based Hypertext for Argumentation

Paul Smolensky^{1,3} Brigham Bell¹ Barbara Fox^{2,3} Roger King¹ Clayton Lewis^{1,3}

¹Department of Computer Science, ²Department of Linguistics &
³Institute of Cognitive Science
University of Colorado, Boulder, CO 80309

ABSTRACT

In this paper we describe a hypertext system we are developing for the support of reasoned argumentation: the EUCLID project. We use the project to address two general problems arising with hypertext: the problems of controlling user/document interaction, and the problem of controlling the screen. We suggest that guiding users' interaction with hypertext is difficult because of the unique form of discourse that hypertext represents, and that structuring user/document interaction can be achieved through specializing to a particular type of material and designing the hypertext system to respect the particular discourse structure characteristic of that material. EUCLID's design is tuned to the structure of reasoned discourse. The problem of screen management in EUCLID is a serious one, because our presentation of complex arguments requires mapping the complex logical relations between parts of realistic arguments onto complex spatial relations between items in the display. We describe a general system we are developing which provides this high degree of control for hypertext screen management. This system represents a constraint-based approach to hypertext, in which the items from the underlying database that are to be displayed may each contribute a number of constraints on the layout; a general constraint-satisfier then computes a screen layout that simultaneously satisfies these constraints. Each time an item is to be added to or deleted from the screen, the constraint set is adjusted and the screen layout is recomputed; thus the spatial relationships on the screen provide at all times a veridical representation of the underlying relations between displayed database items. This kind of strong screen control is demanded by hypertext applications which, like ours, are fine grained: the number of nodes and links being displayed number in the hundreds.

0. INTRODUCTION

This paper addresses two general problems arising in hypertext systems: the problem of controlling the interaction between user and document, and the problem of controlling the screen.

Hypertext is a new form of communication that introduces a new challenge in controlling interaction between document and user. This challenge arises because hypertext falls in a difficult no-man's-land between the traditional discourse media of ordinary text and conversation. In ordinary text, the reader has virtually no influence on the presentation of information, so the author has complete control over the organization of material, and an obligation to design an organization that will be satisfactory for the reader. In ordinary conversation, the interlocutor has great influence on the flow of information, but the speaker is physically present to make real-time decisions about how to dynamically structure that information. In

hypertext, the reader has tremendous control over the flow of information, but the author is not there to provide real-time guidance. There is thus a tendency for users of hypertext systems to be denied the kind of guidance possible in ordinary text or conversation, guidance that is generally necessary for a user to effectively absorb information from a source whose structure is complex and not thoroughly familiar. In short, hypertext systems often set up an interaction between user and document that is poorly structured, and users simply get lost.

One approach to structuring hypertext interaction is to focus on a certain type of material, and base the hypertext interaction on a study of the discourse structure characteristic of that type of material. In the work reported here, we have focussed on material that is characterized by a high degree of *logical structure*: reasoned argumentation. By focussing on this *reasoned discourse*, it is possible to develop a hypertext system designed to support user-computer interaction that revolves specifically around logical structure. Section 1 of this paper reports on the hypertext system we are developing to support reasoned discourse: the EUCLID system.¹ A fuller description of this system may be found in [Smol88].

The second general hypertext problem we address is that of controlling the screen. In presenting large complex arguments, it is important to design a well-organized screen in which many logical relationships are implicitly encoded in the spatial relations between items on the screen. This will be elaborated in some detail below, but to take a simple example, it is often useful to place opposing arguments side-by-side, using a representation in which the logical opposition of the arguments is implicitly encoded in their relative spatial locations. Such implicit notations are often desirable; they are absolutely necessary in situations where explicitly indicating all relationships (eg., by labelled arrows) would create a display that is hopelessly cluttered.

The demands of creating the highly structured displays needed to portray complex arguments has led us to an approach to screen management we call *constraint-based display*. The basic idea is that every item and relationship to be displayed contributes a certain number of constraints on the screen layout; the display system then designs an appropriate display by simultaneously satisfying all these constraints. We are developing a general constraint-based hypertext system for displaying items selected from a large network database, and applying this general system specifically to our argumentation domain.

Section 1 of this paper is a description of the EUCLID system for supporting reasoned discourse. This motivates the constraint-based approach to hypertext, and gives a concrete arena in which to then discuss the constraint-based approach in Section 2. The project is in its early phases: much of the system to be described has already been implemented; the remainder has been designed and is currently being built.

1. REASONED DISCOURSE AND THE EUCLID PROJECT

Spoken language, writing, and mathematical notation and proof are symbolic systems that have profoundly affected human reasoning capacity. Modern computers are powerful, active symbolic systems with the

1. The environment EUCLID is unrelated to the programming language Euclid [Lamp77, Holt83].

potential, we believe, to provide significant further advances in human reasoning ability.

In this Section of the paper, we describe a tool we are developing for helping people create and assess reasoned arguments and communicate these arguments to others. The tool provides reasoners with a *language*, ARL, for expressing their arguments in a clear, precise, and relatively standardized fashion. The medium in which this language is realized is a computer environment we call EUCLID.

In EUCLID, the computer plays a role analogous to acoustic or print media in verbal or written argumentation: it provides a medium—an extremely powerful one—for supporting logical discourse among human users. We are *not* proposing to use computer reasoning to replace human reasoning. Our goal is to give users the expressive and analytic power necessary to elevate the effectiveness of their own reasoned argumentation.

1.1. The goal: Enhancing reasoned discourse

A central activity in theoretical research is the construction of reasoned arguments supporting theoretical conclusions. The problem we address is a practical one: how can this activity be effectively supported? While our focus is on argumentation of the type found in research papers (for we are developing our tool while using it ourselves in our work), we also consider, to a lesser extent, related forms of argumentation. Other examples of *reasoned discourse*, in addition to research papers, include policy advocacy for decision making (e.g. reasoned letters to the editor), pedagogy in theoretical disciplines such as linguistics and physics, and, to a certain extent, reasoned argumentation in everyday conversation.

The domains of analysis we have in mind are ones that are not strictly formal, so that mathematically rigorous proofs are not possible. Our goal is to enhance reasoned discourse that now occurs in natural—not formal—language.

We take the goal of enhancing reasoned discourse to integrally incorporate both support of explicit discourse processes (like reading and writing) and also support of reasoning itself. A clear distinction between cognition and communication is particularly problematic in the area of reasoning. Argumentation is the construction of a symbolic structure intended to persuade through conformity to certain social conventions: reasoning is intrinsically a discourse phenomenon. The point is underlined by a substantial body of research on writing which suggests that what writers most need support for is the *planning* of documents: the main problem is deciding exactly what to say, and devising an overall presentation plan [Flow80, Greg80, Kell85a, Kell85b, Kell88]. Even in the writing of few-paragraph business letters, people spend two-thirds of their time planning [Goul80]. In the domain of reasoned discourse, it is clear that planning—laying out the line of argumentation—is a crucial and almost completely unsupported activity. The process of planning a research paper—the working out of the claims to be made and the arguments to support them—is essentially the process of carrying out the theoretical component of the research itself.

Our goal, then, is to provide a tool to facilitate reasoning and enhance reasoned discourse, particularly writing.

1.2. The tool

In this section we specify the functionality that ARL and EUCLID are intended ultimately to provide. Presently, only part of this functionality has been implemented.

1.2.1. ARL: An Argumentation Representation Language

Our fundamental hypothesis is that in constructing an argument, two kinds of knowledge are brought to bear: knowledge of the subject domain, and knowledge of argumentation per se. These respectively manifest themselves as argument content and argument structure. A *general purpose* argumentation tool helps the user by virtue of its knowledge of argument structure, not argument content.

Drawing a clear line between structure and content is so crucial to this research that we find it useful to give that line a concise name: *the Divide*. Content information is *below* the Divide; structure information is *above* the Divide. Examples of assertions below the Divide are:

- Lower interest rates lead to bull markets.
- Linguistic principle *X* is universal.
- Approach *Y* to knowledge representation is seriously flawed.

Above the Divide we find statements such as:

- Claim C_1 supports claim C_2 .
- Claim C is the main point of argument A .
- Claim C is made by author S .
- Claim C_1 made by author S_1 contradicts claim C_2 made by author S_2 .
- Term T is used by author S_1 to mean phrase D_1 but by author S_2 to mean phrase D_2 .

This latter sort of information is often not explicitly stated in text, but in it lies the structure that characterizes reasoned discourse. (The crucial importance of information above the Divide has also been emphasized by Zukerman and Pearl [Kuck85]. In the tutoring context, they have studied how such information is introduced through natural language expressions they call *meta-technical utterances*.)

Information below the Divide involves terms and predicates that vary completely from one domain of argumentation to another. But information above the Divide involves a reasonably constant vocabulary: the examples above use the terms *claim C*, *argument A*, *author S* and the predicates *supports*, *main-point*, *asserts*, *contradicts*. This vocabulary is characteristic of reasoned discourse in any domain.

ARL offers a set of primitive term-types and primitive predicates for formally describing argument structure, such as those mentioned in the previous paragraph. In addition, it incorporates high-order structures formally defined by combining simpler ones: for example, high-level standard schematic structures for arguments, arguments by analogy, allegations of misrepresentations, and argument refutations.

To give users the expressive power needed in real argumentation, ARL must let users extend the language's set of primitives and must provide the machinery for them to formally create their own high-

order constructs.

The ARL statement corresponding to "Claim C_1 supports claim C_2 " uses the formal predicate **supports** to relate two entities that have formal type **claim**. The content of each claim is not expressed formally, but *informally*, in natural language. For example, the content of C_1 might be "Lower interest rates lead to bull markets." Thus ARL is a *semi-formal* language: argument *structure* information (above the Divide) is represented *formally*, while argument *content* (below the Divide) is represented *informally*. The computer has access to the semantics of the formal information, but only the user has access to the semantics of the informal information.

We believe that the notion of semi-formal language is potentially of great value to the design of effective joint human/computer systems: its applicability extends beyond joint human/computer reasoning, to include nearly any joint human/computer activity.

1.2.2. EUCLID: An Environment for User Construction of Logical Informal Discourse

A formal representation of the structure of the argument contained in a theoretical research paper is too large for anyone to explicitly represent without the help of a data manager. The computer environment EUCLID keeps track of ARL representations, allowing users to select portions of the argument to be displayed on a high-resolution graphics terminal. In addition to displaying ARL structures, EUCLID allows users to add new structures, and modify old structures. Users can state that they want to create a new instance of a higher-level construct (say an analogy), whereupon EUCLID prompts the user for the necessary inputs and manages the details of creating the necessary data structures. Information is displayed in ways specially designed to show the various argument components. For example, there are special displays for analogies, refutations, or retrieval requests like "show all claims whose validity depends on this one." Along with the capability to create new types of argument structures, users have the capability to specify new ways of displaying them. Users also have considerable choice among alternative types of displays.

An argument created in EUCLID may contain full pieces of text: EUCLID is intended to provide a unified environment for working out an argument *and* expressing it in text. A specific example will be illustrated in the next section; to give the general idea, reading a "journal article" in EUCLID might proceed like this. After reading the abstract, the user decides what further information is of most interest: an experimental procedure, a source for a "fact," a theoretical argument, the theoretical assumptions. The requested information is retrieved and the screen is reconfigured to incorporate the new information, which is displayed in a manner reflecting its information type. The retrieved information might be a piece of text, like a section of a paper, or perhaps a table or graph, or even a running program. In one unified ARL datastructure is contained both the underlying logical structure and the pieces of text that present the argument. EUCLID is a hypertext system specially tailored for logical material—reasoned discourse.

1.2.3. An example: The Chinese room debate

As an example of how EUCLID might look to a user reading an argument, we will consider an argument that has been our testbed: the "Chinese room" argument of John Searle [Sear80]. This argument claims to show that instantiating an AI program—even one that could answer questions indistinguishably from a human and thereby pass the Turing test—cannot be sufficient grounds for saying that a machine "understands" in

the full sense of the word. The core of the argument is the following analogy. A Chinese story is slipped under the door of a closed room, and then Chinese questions about the story are slipped in. Back under the door come Chinese answers to the questions, indistinguishable from those of a native speaker. As it happens, inside the room is Searle himself, working away at copying Chinese symbols he doesn't understand from big books under the guidance of a complex set of English instructions. According to Searle's analogy, the Chinese characters are to the Searle in the room as English is to a question answering computer: completely meaningless forms being manipulated without any understanding.

The commentary from numerous cognitive scientists that was published with the article reveals a tremendous diversity of outlooks, and appears to evidence a considerable amount of confusion about just what Searle's argument is. The argument is still highly active today, meriting an entire session of the 1986 meeting of the Society of Philosophy and Psychology. Our goal is to use EUCLID to delineate, as clearly as possible, the positions taken by the numerous participants in the published debate; in the process, the expressive adequacy of ARL and the usability of EUCLID will be challenged by a truly worthy argument.

Figures 1 through 6 illustrate how EUCLID might be used to study an ARL representation of the Chinese room debate. We imagine the user has read the text, and is ready for an analysis. Figure 1 gives a tabular display of the top level of Searle's argument. This is a relatively clean display, in which a lot of relational information is implicit in the arrangement of items on the screen. Figure 2 shows the relationships explicitly. The left side of the diagram are claims and arguments that Searle attributes to his opponents, those accepting the position of "strong AI." On the right side of the diagram are the claims and arguments that Searle accepts. At the very top of the left side is the main claim of the strong AI view. Immediately beneath the strong AI position are three arguments supporting it, which Searle attributes to his opponents; to the right of each one is Searle's counter-argument. Below these counter-arguments are Searle's arguments in favor of his own position, the main claim of which is stated at the top of the right column. To the left of Searle's arguments are refutations of them which he attributes to his opponents and to the right of these are his counter-arguments.

A user facing the austere display of Figure 1 might request that the implicit relationships be made explicit, giving rise to Figure 2. Next we suppose the user to have selected "The Chinese Room" for further information about Searle's key argument. This selection leads to a new display, Figure 3, which expands upon the selected item. (Note that the new display is coherently displayed without intervention by the user.) Figure 3 shows the form of the Chinese room argument: it is an analogy, and is displayed in an appropriate form. On the left side of the Chinese Room display are the elements of the Chinese room domain; on the right side are the corresponding elements of the AI system. Searle's analogy is a mapping that carries elements and claims from the Chinese Room domain into elements and claims of the AI domain.

Having been shown the explicit form of the Chinese Room analogy, we next suppose that the user wishes to consult the text to check the accuracy of the analysis given for the analogy. Figure 4 shows the text separated into pieces that are explicitly connected to components of the analogy analysis. Deciding this representation of the relationships is too messy, the user requests a simpler representation. Figure 5 shows the text, unbroken, next to the analogy analysis. After selecting a particular item in the analysis, the

Strong AI

Searle

An AI program
(running on a von Neumann machine)
that can pass the Turing test lacks no
important element of understanding

An AI program that can pass the Turing test
lacks an important element of understanding
that would be present if the program were
implemented on a machine with the causal
powers of the brain.

The argument from information processing

The argument from formality

The argument from behavior

Just behaviorism

The argument from implementation independence

Just modern-day dualism

The systems reply

The argument from formality:
The Chinese room

The robot reply

The internal Chinese room

The internal Chinese room
+ peripherals

The brain simulator reply

The argument from water pipes

The argument from lactation

Figure 1. An example EUCLID screen showing the high-level argument structure in an analysis of the Chinese room debate. Figures 1 through 6 are Macintosh™ mock-ups from the system design. (Figures 7 through 10 show screens from an implementation.)

Strong AI

Searle

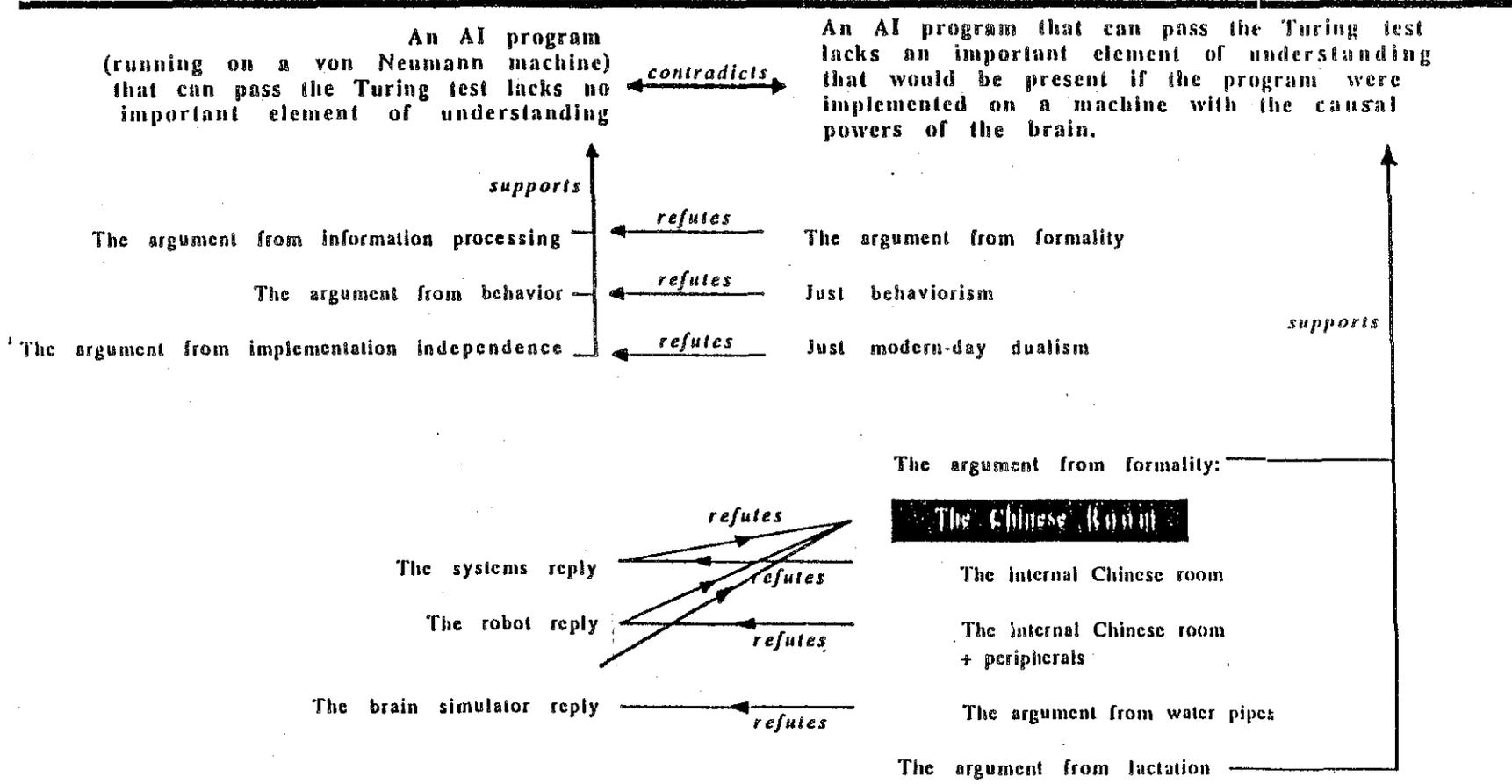


Figure 2. The screen of Figure 1 after relations that were implicitly represented by spatial relationships have been explicitly represented by labelled arrows. One node on the screen, "The Chinese Room," has been selected for expansion in place.

Strong AI

Searle

An AI program (running on a von Neumann machine) that can pass the Turing test lacks no important element of understanding

An AI program that can pass the Turing test lacks an important element of understanding that would be present if the program were implemented on a machine with the causal powers of the brain.

contradicts

supports

supports

The argument from information processing

refutes

The argument from formality

The argument from behavior

refutes

Just behaviorism

The argument from implementation independence

refutes

Just modern-day dualism

Text: Searle's article, pp. 417-8

Suppose that I'm locked in a room and given a large batch of Chinese writing. Suppose furthermore (as is indeed the case) that I know no Chinese, either written or spoken, and that I'm not even sure that I could recognize Chinese writing as Chinese writing distinct from, say, Japanese writing or meaningless squiggles. To me, Chinese writing is just so many meaningless squiggles.

Now suppose further that after this first batch of Chinese writing I am given a second batch of Chinese script

together with a set of rules for correlating the second batch with the first batch. The rules are in English, and I understand these rules as well as any other native speaker of English.

They enable me to correlate one set of formal symbols with another set of formal symbols, and all that "formal" means here is that I can identify the symbols entirely by their shapes.

Now suppose also that I am given a third batch of Chinese symbols

together with some instructions, again in English, that enable me to correlate elements of this third batch with the first two batches, and these rules instruct me how to give back certain Chinese symbols with certain sorts of shapes in response to certain sorts of shapes given me in the third batch.

... Suppose also that after a while I get so good at following the instructions for manipulating the Chinese symbols and the programmers get so good at writing the programs that from the external point of view ... my answers are absolutely indistinguishable from those of native Chinese speakers.

... It seems to me quite obvious in the example that I do not understand a word of the Chinese stories.

The argument from formality:

The Chinese Room

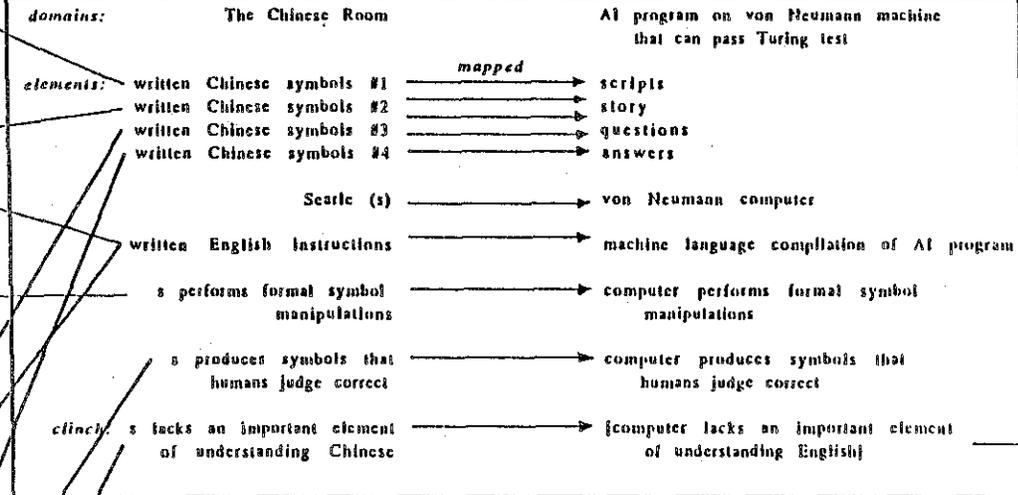
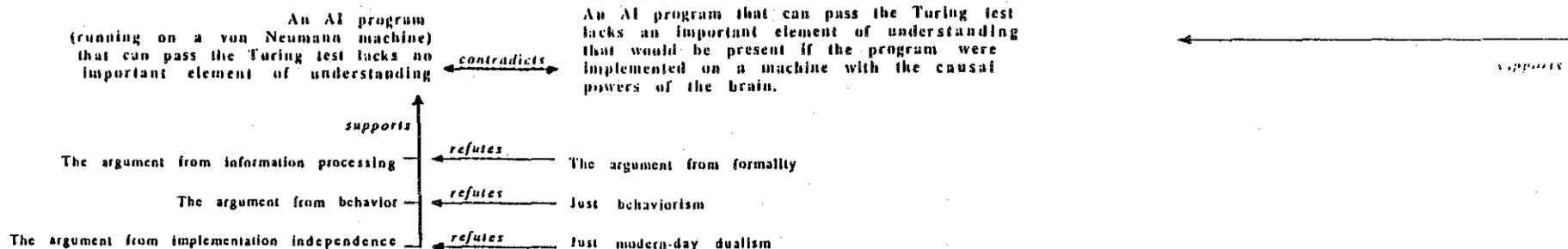


Figure 4. The screen of Figure 3 after a request for the original text underlying the individual elements in the analysis of the Chinese Room argument.

Strong AI

Searle



Text: Searle article, pp. 417-8

Suppose that I'm locked in a room and given a large batch of Chinese writing. Suppose furthermore (as is indeed the case) that I know no Chinese, either written or spoken, and that I'm not even sure that I could recognize Chinese writing as Chinese writing distinct from, say, Japanese writing or meaningless squiggles. To me, Chinese writing is just so many meaningless squiggles. Now suppose further that after this first batch of Chinese writing I am given a second batch of Chinese script together with a set of rules for correlating the second batch with the first batch. The rules are in English, and I understand these rules as well as any other native speaker of English. They enable me to correlate one set of formal symbols with another set of formal symbols, and all that "formal" means here is that I can identify the symbols entirely by their shapes. Now suppose also that I am given a third batch of Chinese symbols together with some instructions, again in English, that enable me to correlate elements of this third batch with the first two batches, and these rules instruct me how to give back certain Chinese symbols with certain sorts of shapes in response to certain sorts of shapes given me in the third batch. . . . Suppose also that after a while I get so good at following the instructions for manipulating the Chinese symbols and the programmers get so good at writing the programs that from the external point of view . . . my answers are absolutely indistinguishable from those of native Chinese speakers. . . . it seems to me quite obvious that I do not understand a word of the Chinese stories.

The argument from formality:

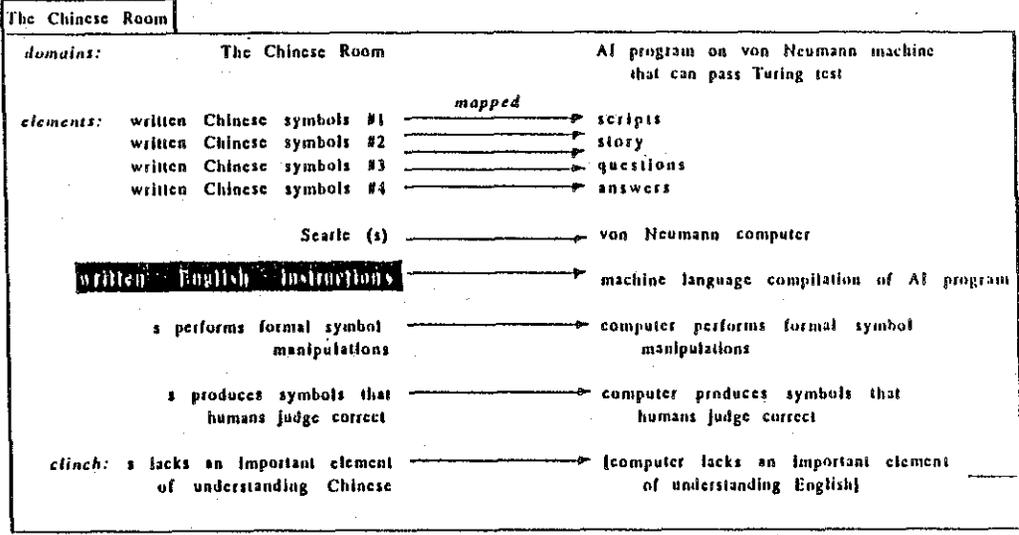


Figure 5. The screen of Figure 2 after a request for the text underlying a single selected element of the analysis. This display indicates the underlying text differently from the display of Figure 4.

Strong AI

Searle

An AI program
(running on a von Neumann machine)
that can pass the Turing test lacks no
important element of understanding

An AI program that can pass the Turing test
lacks an important element of understanding
that would be present if the program were
implemented on a machine with the causal
powers of the brain.

The argument from information processing

The argument from formality

The argument from behavior

Just behaviorism

The argument from implementation independence

Just modern-day dualism

The argument from formality:

The Chinese Room

domains:

The Chinese Room

AI program on von Neumann machine
that can pass Turing test

elements:

written Chinese symbols #1
written Chinese symbols #2
written Chinese symbols #3
written Chinese symbols #4

scripts
story
questions
answers

Searle (s)

von Neumann computer

written English instructions

machine language compilation of AI program

s performs formal symbol
manipulations

computer performs formal symbol
manipulations

s produces symbols that
humans judge correct

computer produces symbols that
humans judge correct

clinch: s lacks an important element
of understanding Chinese

{computer lacks an important element
of understanding English}

The systems reply

The internal Chinese room

The robot reply

The internal Chinese room
& peripherals

The brain simulator reply

The argument from water pipes

The argument from lactation

Figure 6. The screen of Figure 2 after elimination of the arrows explicitly representing the relations of the displayed items; the relations are indicated only implicitly, in the spatial relations on the screen of the displayed items.

corresponding parts of the text become underlined.

After studying the analysis side-by-side with the text, the user decides to accept the analysis for the time being and proceed. A request to remove the text and the explicit relationships gives the relatively clean screen of Figure 6, from which it is now reasonable to proceed by selecting another part of the overall argument for analysis.

The displays of Figures 1 through 6 are mockups towards which we are currently striving. The displays of Figures 7 through 10 are screens from the current EUCLID implementation on the Symbolics™ lisp machine. Figure 7 shows the top level of the argument; Figure 8 shows an expanded view in which "the argument from information processing" and "the Chinese Room argument and replies" have both been expanded in place. Figure 9 shows the screen after attention has been shifted to the Chinese Room argument, which now fills the screen; "the systems reply" and "the internal Chinese room" have now been expanded out. In Figure 10, we have returned to the top level of the argument; the part of the argument in which Searle refutes his opponents has been reduced to a fairly compact form, and the part in which he supports his own position has been expanded. All the operations that have been performed by the user are simple menu selections, indicating what information to add or delete; all screen management has been done by EUCLID.

The displays of Figures 7 through 10 are generated in a top-down, recursive fashion, in which large datastructures called *display templates* specify the overall organization of the large portion of the screen occupied by a particular argument; when imbedded arguments are displayed, control is governed by the templates for the imbedded arguments. The displays of Figures 1 through 6 demand an additional degree of screen control relative to those of the existing implementation; switching to the use of arrows to indicate relationships, as in going from Figure 1 to Figure 2, requires a degree of local control that is cumbersome for the current top-down, template-driven display system. To achieve the more local screen control required by Figures 7 through 10, we are currently re-implementing EUCLID using the constraint-based display approach to be described in Section 2.

1.2.4. Creating arguments

Having described the kinds of capabilities EUCLID is intended to provide for reading arguments, we now consider argument generation. It is useful to distinguish a number of processes which must all be supported by EUCLID. These processes are closely related to processes that have been studied in the creation of ordinary text [Flow88, Haye80]. The processes are intermingled, and should not be viewed as serial phases.

- *Dump*: Generate terms and assertions the author feels to be central to the argument. EUCLID serves as electronic paper.
- *Reader-preprocess*: Indicate for various terms and assertions what they assume about the reader: background, interests, what other items have been previously read (*prerequisite* relations), etc. EUCLID stores this information for use in the linearization process (below).
- *Organize*: Insert definitions of terms, relations between assertions (eg., *supports*, *contradicts*). EUCLID serves as ARL structure editor and browser.
- *Fill-in*: Generate missing terms, claims, and arguments. EUCLID provides templates for common arguments types, checks for missing components of these argument templates, and satisfies useful database queries (eg. "find claims lacking *supports* links"). EUCLID's

EUCLID

Refute and Support Argument	
Main Claim: Only brains and machines with the same causal powers as brains can think.	
Refute Argument	
Main Claim: An AI program (running on a von Neumann machine) that can pass the Turing test lacks an important element of understanding.	
Claim to Refute: An AI program (running on a von Neumann machine) that can pass the Turing test lacks no important element of understanding.	
<i>Claims</i>	<i>Refutation</i>
The argument from information processing. • • The Chinese Room Argument and Replies.	
The argument from behavior. • • Just behaviorism.	
The argument from implementation independence. • • Just modern-day dualism.	
Support	
• argument from lactation	

Figure 7. An actual screen image of the high-level argument structure of the Chinese room debate, from a Symbolics™ implementation of EUCLID. The overall argument structure is that of a **refute-and-support-argument**: opponents' arguments are stated and refuted, then supporting arguments are given. Here the **refute** part has three sub-arguments and corresponding refutations; the **support** part has a single sub-argument.

EUCLID

Refute and Support Argument																					
Main Claim: Only brains and machines with the same causal powers as brains can think.																					
Refute Argument																					
Main Claim: An AI program (running on a von Neumann machine) that can pass the Turing test lacks an important element of understanding.																					
Claim to Refute: An AI program (running on a von Neumann machine) that can pass the Turing test lacks no important element of understanding.																					
<i>Claims</i>	<i>Refutation</i>																				
Basic Argument	Refute and Support Argument																				
Main Claim: The argument from information processing.	Main Claim: The Chinese Room Argument and Replies.																				
Analogy Argument	Refute Argument																				
<table border="0"> <tr> <td align="center"><i>Antecedent Domain</i></td> <td align="center"><i>Consequent Domain</i></td> </tr> <tr> <td align="center">Humans</td> <td align="center">Computers</td> </tr> <tr> <td>The human brain does something called "information processing."</td> <td>The computer does information processing.</td> </tr> <tr> <td>Clinch: Humans understand the information that they are processing.</td> <td>Claim: Computers understand the information that they are processing.</td> </tr> </table>	<i>Antecedent Domain</i>	<i>Consequent Domain</i>	Humans	Computers	The human brain does something called "information processing."	The computer does information processing.	Clinch: Humans understand the information that they are processing.	Claim: Computers understand the information that they are processing.	<table border="0"> <tr> <td align="center"><i>Claims</i></td> <td align="center"><i>Refutation</i></td> </tr> <tr> <td>The systems reply •</td> <td>The internal Chinese room.</td> </tr> <tr> <td>The robot reply •</td> <td>The internal Chinese room with peripherals</td> </tr> <tr> <td>The brain simulator reply •</td> <td>the argument from water pipes.</td> </tr> </table>	<i>Claims</i>	<i>Refutation</i>	The systems reply •	The internal Chinese room.	The robot reply •	The internal Chinese room with peripherals	The brain simulator reply •	the argument from water pipes.				
<i>Antecedent Domain</i>	<i>Consequent Domain</i>																				
Humans	Computers																				
The human brain does something called "information processing."	The computer does information processing.																				
Clinch: Humans understand the information that they are processing.	Claim: Computers understand the information that they are processing.																				
<i>Claims</i>	<i>Refutation</i>																				
The systems reply •	The internal Chinese room.																				
The robot reply •	The internal Chinese room with peripherals																				
The brain simulator reply •	the argument from water pipes.																				
	Support																				
	Analogy Argument																				
	<table border="0"> <tr> <td align="center"><i>Antecedent Domain</i></td> <td align="center"><i>Consequent Domain</i></td> </tr> <tr> <td align="center">The Chinese Room</td> <td align="center">AI program on von Neumann machine that can pass Turing test</td> </tr> <tr> <td>written Chinese symbols #1 •</td> <td>scripts</td> </tr> <tr> <td>written Chinese symbols #2 •</td> <td>story</td> </tr> <tr> <td>written Chinese symbols #3 •</td> <td>questions</td> </tr> <tr> <td>written Chinese symbols #4 •</td> <td>answers</td> </tr> <tr> <td></td> <td>Searle • von Neumann</td> </tr> <tr> <td>written English instructions •</td> <td>machine language compilation of AI program</td> </tr> <tr> <td>Searle performs formal symbol manipulations</td> <td>computer performs formal symbol manipulations</td> </tr> <tr> <td>Searle produces symbols that •</td> <td>computer produces symbols</td> </tr> </table>	<i>Antecedent Domain</i>	<i>Consequent Domain</i>	The Chinese Room	AI program on von Neumann machine that can pass Turing test	written Chinese symbols #1 •	scripts	written Chinese symbols #2 •	story	written Chinese symbols #3 •	questions	written Chinese symbols #4 •	answers		Searle • von Neumann	written English instructions •	machine language compilation of AI program	Searle performs formal symbol manipulations	computer performs formal symbol manipulations	Searle produces symbols that •	computer produces symbols
<i>Antecedent Domain</i>	<i>Consequent Domain</i>																				
The Chinese Room	AI program on von Neumann machine that can pass Turing test																				
written Chinese symbols #1 •	scripts																				
written Chinese symbols #2 •	story																				
written Chinese symbols #3 •	questions																				
written Chinese symbols #4 •	answers																				
	Searle • von Neumann																				
written English instructions •	machine language compilation of AI program																				
Searle performs formal symbol manipulations	computer performs formal symbol manipulations																				
Searle produces symbols that •	computer produces symbols																				

Figure 8. The screen from Figure 7 after the **refute** part of the argument has been expanded to show more detail.

EUCLID

Refute and Support Argument	
Main Claim: The Chinese Room Argument and Replies.	
<p align="center">Refute Argument</p> <p>Main Claim: Minor modifications of the Chinese Room Argument defeat all counter-arguments. Claim to Refute: The Chinese Room is wrong for a variety of reasons.</p>	<p align="center">Operations on Refute Argument.</p> <p align="center">view object at top <input type="button" value="hide"/> add refutation pair</p>
<i>Claims</i>	<i>Refutation</i>
<p align="center">Basic Argument</p> <p>Main Claim: The systems reply</p> <ul style="list-style-type: none"> • Concede that: the person who is locked in the room doesn't understand Chinese. • He is merely a part of a whole system. • The whole system understands. 	<p align="center">Basic Argument</p> <p>Main Claim: The internal Chinese room.</p> <ul style="list-style-type: none"> • Have the person in the Chinese Room memorize (internalize) the elements of the system. All calculations are done in the person's head. Call this the internal Chinese Room. • It is still possible (and probable) that the person can memorize all of this, and still doesn't understand.
The robot reply • The internal Chinese room with peripherals The brain simulator reply • the argument from water pipes.	
Support	
Analogy Argument	
<i>Antecedent Domain</i>	<i>Consequent Domain</i>
<i>The Chinese Room</i>	<i>AI program on von Neumann machine that can pass Turing test</i>
written Chinese symbols #1 • scripts written Chinese symbols #2 • story written Chinese symbols #3 • questions written Chinese symbols #4 • answers Searle • von Neumann written English instructions • machine language compilation of AI program Searle performs formal symbol manipulations • computer performs formal symbol manipulations Searle produces symbols that humans judge correct • computer produces symbols that humans judge correct	Claim: computer lacks an important element of understanding English.
Cliché: Searle lacks an important element of understanding Chinese	

Figure 9. The refutation of "The argument from information processing" is itself a **refute-and-support-argument**. Here it has been expanded to fill the display.

EUCLID

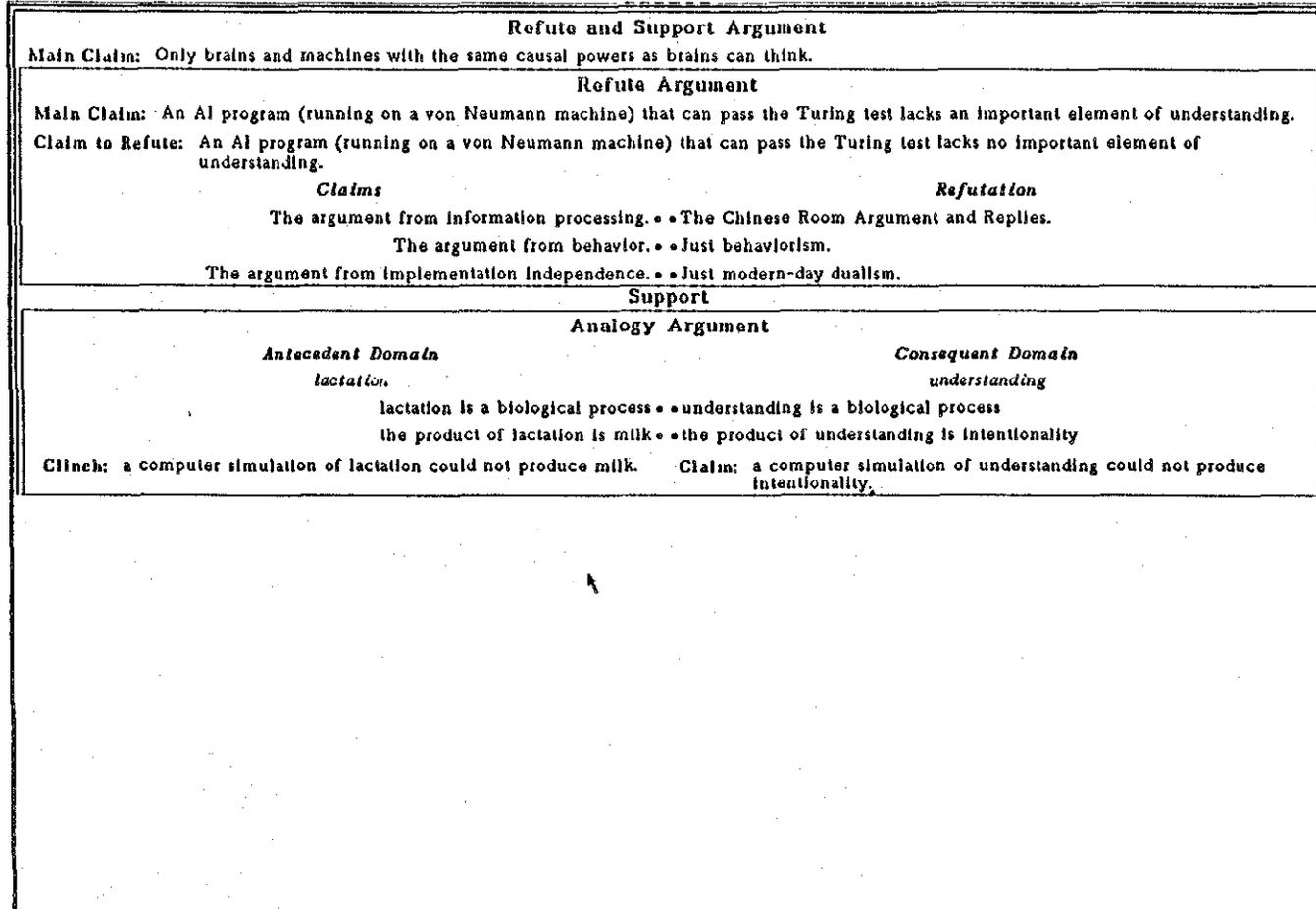


Figure 10. A return to the top-level of the whole argument, as shown in Figure 7, with the support sub-argument now expanded in place. It is an **analogy-argument**.

library of examples of different argument types lets the user browse for possible approaches.

- *Linearize*: Impose on parts of the argument graph a linear order, thereby generating a document. EUCLID partly automates this process, making use of **prerequisite** relations, and filtering the database on intended readers' background and interests. Schemas for document types help guide this process.
- *Edit prose*: Generate readable text. EUCLID functions as text editor integrated into ARL structure editor.

The support provided by EUCLID in these processes is substantial. While the activity of expressing an argument in ARL form is itself helpful in developing the argument, the support that EUCLID can supply on the basis of the formal ARL relations that it can process is a major part of the benefit of using EUCLID instead of pencil and paper for developing arguments. On the other hand, we feel that the process of expressing an argument in ARL form requires extensive human processing and that the prospects for automating the process are dim.

1.3. Effectiveness of EUCLID

Three techniques for overcoming the limitations of human reasoning capacity are exploited by EUCLID. In discussing them, we will illustrate the techniques with examples from EUCLID but also from a familiar and extremely powerful reasoning tool: algebra.

The first technique is the *elimination of irrelevant information*. In algebra, one important reason we can solve problems is that once the problem has been cast into a set of equations, we can forget what the variables denote, and not be distracted by that now-irrelevant information. The process of expressing an argument in EUCLID form has some of the same advantages as expressing a word problem in algebraic notation. Once an argument's *structure* has been expressed in ARL, a number of structural analyses can proceed without the distractions introduced by the content of the argument: for example, it is possible to find all claims that would become unsupported if a given claim in the argument were denied, or all the unsupported claims, or pivotal claims on which large parts of the argument hinge.

The second technique is *explicit chunking*. In algebra, a new variable can be introduced to denote a complex subexpression, and the simplification the new variable affords enables us to cope with expressions that would otherwise be too complex to manage. In EUCLID, a central tool for assessing arguments is to encapsulate large subarguments into a single conclusion, often, a conclusion that is only *implicit* in the original, non-EUCLID form of the argument. This encapsulation enables users to more effectively cope with complex arguments.

The third technique is *explicit decomposition*. In algebra, it is crucial to decompose a problem into a set of individual equations that can be analyzed separately, or to decompose a complex expression into individual terms that can be analyzed separately. Only by isolating independent subparts can we cope with complex algebraic problems. Similarly, in EUCLID, a central technique is the decomposition of complex arguments into subarguments which can be constructed, understood, or assessed independently.

In addition to these particular techniques for overcoming cognitive capacity limits and enhancing reasoning, there are very general reasons why EUCLID can promote the effectiveness of reasoned discourse.

Expressing an argument in ARL amounts to making notationally *explicit* the kind of logical structure that is often left *implicit* in informal argumentation. Explicit notational systems offer many advantages over implicit systems: standardized systems permit automatization of standard skills; they promote information retrieval; they enable explicit instruction, study and analysis of the notational system; and the public sharing of the system enables evolution of the system and provides a common basis for communication. Furthermore, there is good evidence that providing readers *explicitly* with the overall structure or *macrostructure* of documents significantly increases their comprehensibility and retention [vanD83].

1.4. Relation of EUCLID to existing systems

Because EUCLID is an AI-based system for argumentation and reasoning, the project is often identified with AI projects aimed at computer understanding of arguments in natural language form [Alva85, Birn82, Birn80, Flow82] and with attempts to formalize the principles of human reasoning (eg. circumscription and default logic—[Arti80]—fuzzy logic—[Zade79, Zade84]—logics with modal operators—[Hint69]—and so forth). In fact, the EUCLID project involves *neither* of these goals. The notation of the EUCLID system, ARL, is a *semi-formal* language, processed both by the computer and by users. Since only information about argument structure is formalized in EUCLID, there is no attempt to formalize domain information as is necessary for an AI system that understands arguments in natural language. Furthermore, the validity of arguments in EUCLID is assessed by the user, informally, so there is no attempt to formulate formal principles by which validity is assessed in complex, realistic arguments.

Nonetheless, ARL is a language for expressing argument structures whose development is being driven in part by the need to provide the power necessary to express the *argument molecules* of Birnbaum, Flowers and McGuire [Birn82, Birn80, Flow82], the *argument units* of Alvarado, Dyer, and Flowers [Alva85] and the argument strategies and fallacies studied in the informal logic literature (for example, [Acoc85, Enge80, Foge82, Govi85]).

There are number of existing general hypertext-like systems that can be related to EUCLID. NoteCards™ [Brow85, VanL85], NotePad© [Cyph86], IdeaSketch™ [Brow85], and, to a much more modest degree, ThinkTank™, are all systems for developing and interrelating ideas. EUCLID can be viewed as a specialization of these systems to the particular domain of argumentation. The system is tuned to the specific demands of argumentation, from the display of information, to the editing operations, to the retrieval of information. One particular manifestation of this tuning is that the screen is continually managed to maintain an orderly display of information and the logical relations between items. The system assumes such a large burden of screen management that EUCLID has forced the development of a new hypertext displaying system, discussed in Section 2 of this paper.

1.5. Discourse analysis

As stated at the beginning of this paper, we feel that hypertext interaction needs to be structured in a way appropriate to the discourse type of the particular material in question. Our construction of the EUCLID system is therefore being guided by research into the structure of reasoned discourse, in currently existing media such as text and conversation, and in the hypertext medium as it emerges. This research is being carried out as part of the EUCLID project, and a number of techniques from discourse analysis are being applied. As explained at the beginning of the paper, hypertext involves a mixture of discourse elements

from conversation and text, and thus provides fertile new ground for discourse analysis. A goal of our analysis of reasoned discourse is to maximize the effectiveness of EUCLID by providing users with the kind of support they most need in constructing, comprehending, and assessing arguments.

2. THE CONSTRAINT-BASED APPROACH TO HYPERTEXT

The EUCLID project requires a screen management system that allows the spatial layout of items on the screen to represent complex relations between dynamically selected items from the underlying database. Each time the user adds or deletes information from the screen, the screen needs to be redesigned so that the spatial layout of the new screen properly reflects the new information content. (Of course, it is important to minimize the screen reorganization so that users do not need to continually make major reorientation to the screen.)

EUCLID is an example of a hypertext application that is *fine grained*: the nodes and links being displayed on the screen at a given time number in the hundreds. Each node contains a small amount of content (on the order of 10 words); the hundreds of relations being displayed obviously cannot each be overtly displayed, eg. by an labelled arrow; most must be displayed implicitly by the spatial arrangement of the displayed nodes. This is just how an outline, for example, displays the tree structure of its nodes: that node x is a child of node y is displayed by placing x below y , indented by one quantum of indentation; that node z is the next sibling of node x is indicated by having node z displayed immediately below x with the same amount of indentation. This sort of implicit representation of relations is used heavily in the EUCLID displays of Figures 1 through 6. (Even where relations are explicitly displayed with arrows in these Figures, the nodes are spatially arranged so that these relations are *simultaneously* implicitly represented in the layout.)

We are developing a general hypertext facility for providing the high degree of screen control demanded by fine grained hypertext applications such as EUCLID. This facility is called *CBH*, for *Constraint-Based Hypertext*. EUCLID is a particular application built on CBH.

CBH is used to display portions of network databases of the sort usually assumed in hypertext. At any given moment, there is a subset of the database that is currently being displayed on the screen: the *active perspective*. (There may be other inactive perspectives stored: these are database subsets that are not currently being displayed). All the items in the active perspective are entitled to influence the current display. The displayed items in the active perspective include all kinds of database items: objects, properties, and relations. CBH is designed for fine grained hypertext systems where perspectives can simultaneously include hundreds of nodes and links.

The layout of the current display is computed as follows. Each item in the active perspective influences the screen layout by contributing *constraints* on the layout; all the constraints contributed by all the items in the active perspective are assembled together into the *active constraint set*. A constraint satisfier then computes a screen layout that satisfies the constraints in the active constraint set. (These constraints may be satisfied in an approximate fashion.)

Examples of the kind of constraints that appear in the active constraint set are as follows. Consider a data object with textual content: say, a EUCLID claim c . This object might contribute to the active constraint set the constraint that in some rectangular region $R(c)$ on the screen, the string giving the content of the

claim c must be printed. This constraint can be written:

```
contains-text( R(c), content(c), Display-attributes(c))
```

Alternatively, in a more compact representation, this object might contribute the constraint that in some rectangular region on the screen, a generic icon designating "claim" must be drawn:

```
contains-graphic( R(c), claim-icon-3, Display-attributes(c))
```

Next consider an item from the database that is not a data object but rather a property: say, the EUCLID attribute `unsupported`, applied to some particular claim c . This item might contribute to the active constraint set the constraint that the display of c must use a large bold font, or, alternatively, must appear in the color red:

```
size( Display-attributes(c) ) = large
style( Display-attributes(c) ) = bold
color( Display-attributes(c) ) = red
```

Finally, consider a database item that is a relation between two objects: say, the EUCLID relation `refutes`, holding between two claims c_1 and c_2 . This relation might contribute to the active constraint set the constraint that the rectangles in which c_1 and c_2 are displayed must be horizontally aligned:

```
horizontally-aligned( R(c1), R(c2))
```

Alternatively, this relation might contribute the constraint that there must be an arrow bearing the label `refutes` from the display of c_1 to the display of c_2 :

```
joined-by-arrow( R(c1), R(c2), Path, "refutes")
```

The constraints contributed by displayed items involve a number of *display variables*, such as locations of rectangles on the screen (R) wherein information is to be displayed, paths of lines and arrows ($Path$), display attributes (such as color, font size and style: `Display-attributes`), and so on. The problem of screen layout is then the problem of assigning values to all these display variables so that all the constraints in the active constraint set are met.

As indicated by the examples above, alternative methods for displaying a given item correspond to different constraints. Choosing to represent a claim by its contents or by an icon, choosing to represent a property by font size or by color, choosing to represent a relation by spatial alignment or by an arrow, all amount to choosing for database items different constraints to contribute to the active constraint set. Each package of constraints that a database item can contribute to the active constraint set is called a *constraint schema*. For example, a constraint schema that has already been mentioned is `joined-by-arrow`: for any binary relation r in the database, the relationship $r(x, y)$ can contribute the constraint

```
joined-by-arrow( R(x), R(y), Path(r), label(r))
```

Thus a particular screen is determined by two things: the active perspective—which determines *what* database items are displayed—together with a choice for each item of a particular display constraint schema to contribute to the active constraint set—which determines *how* each item is displayed. Since it will often be desirable to have all items of a particular type displayed in the same way, i.e., according to the same

constraint schema, it is important to be able to specify at the level of data *types* the choices of which constraint schemata are to be contributed by the various items in the active perspective. This can be arranged as follows.

Consider all the database items in a perspective. Each item has a data type, and we can tack each item onto the database type hierarchy by suspending it from the node for its type. This creates a tree, whose leaves include the database items in the perspective, and whose non-terminal nodes are datatypes. To any node *n* on this tree we can attach information specifying the constraint schema governing the constraints contributed by the given individual or type *n*. We call this tree a *display constraint schema hierarchy*, or for short, a *display hierarchy*. A valid display hierarchy has the property that every individual database item in the tree either has a constraint schema directly attached to it, or inherits a constraint schema from some ancestor in the tree.

Thus to specify a display in CBH, it is necessary to specify a perspective and a display hierarchy. The CBH displayer takes the active perspective and for each item in that perspective, it finds in the display hierarchy the appropriate constraint schema. It then uses this constraint schema to generate the set of particular constraints to be contributed by that item. It assembles the constraints contributed by all the items in the perspective to form the active constraint set. It then uses its constraint satisfier to assign values to all the display variables occurring in the constraints. Once these values are assigned, the display is composed; the displayer then uses the computed values of the display variables to paint the display on the screen.

Figures 11 through 15 use a small example to summarize this discussion of how CMH works. Figure 11 gives the overall architecture of the CBH system. Figure 12 shows a small EUCLID-like screen. Figure 13 shows the relevant portion of the database; the active perspective corresponding to Figure 12 consists of all the instances in the portion of the database shown in Figure 13. Figure 14 shows the display hierarchy used to create the screen of Figure 12; all specification of constraint schemata is done at the type level, so the instances suspended from the types are not shown. The relation linking nodes in the display hierarchy to their types is called **display-constraint-schema**. Figure 15 gives part of the definitions of the constraint schemata used, using a Prolog-like notation in which the names of variables begin with upper-case letters.²

The preceding discussion has treated display constraints as *hard* constraints: predicate-calculus-like propositions that would presumably have a truth value of either *true* or *false*. Actually, many of the important constraints in screen layout are *soft*: they are constraints that can be met more or less well, and the goal of constraint satisfaction is to meet the total set of soft constraints as well as possible. Examples of such soft constraints include: make the location of an item as close as possible to its position in the previous display (assuming it was present in the previous display); make the font size as close as possible to 10 points (where smaller font sizes may be necessary to fit items onto a crowded screen); line up horizontally as well as possible the following items; place this item as high as possible on the screen; place these items as close as possible to each other; and so forth. Our approach to display constraints incorporates soft constraints as well as hard ones. As usual with soft constraints, numerical strengths are

2. The conventions for formally specifying constraints used in the text were modified slightly relative to Figure 15, for expository convenience. The intent of this paper is to communicate the basic idea of constraint-based hypertext; details of constraint specification will be taken up elsewhere.

CBH architecture

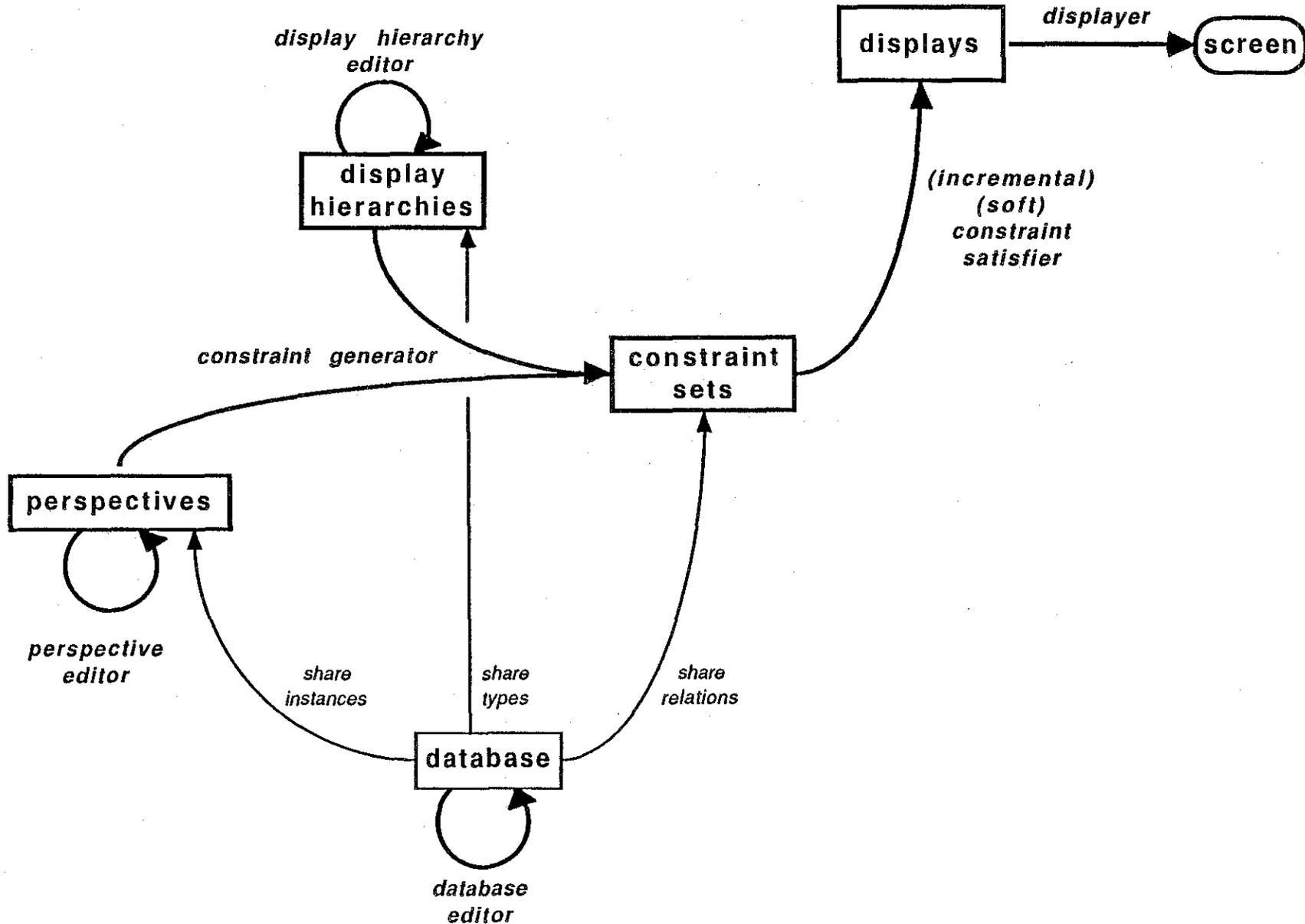


Figure 11. The overall architecture of the CBH (Constraint-Based Hypertext) system we are developing as a general-purpose hypertext system on which to build EUCLID.

screen

The Searle Debate

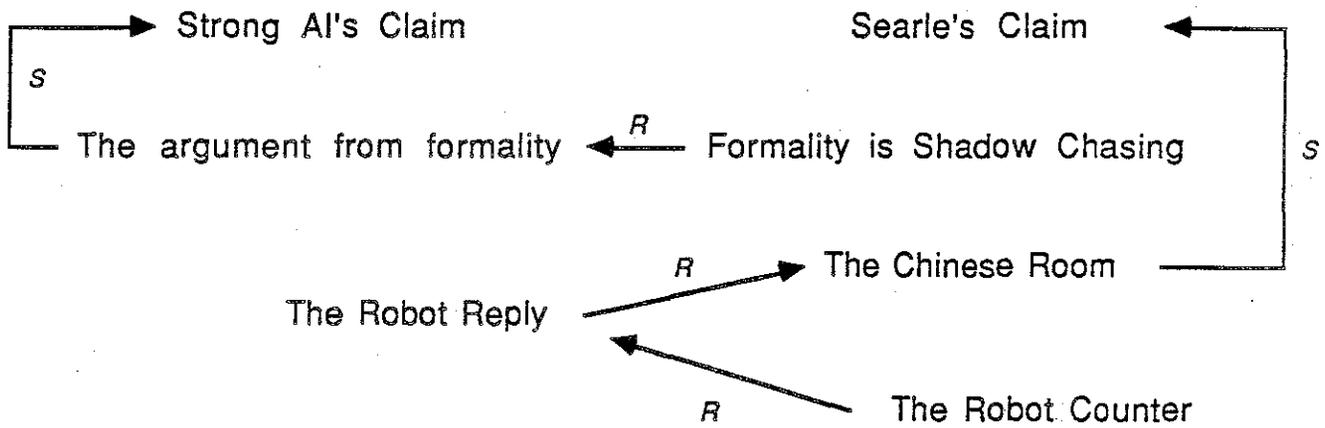
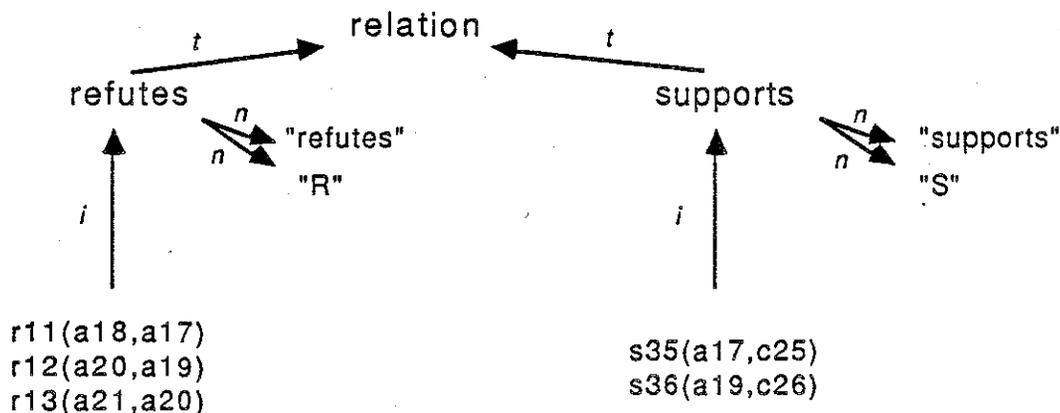
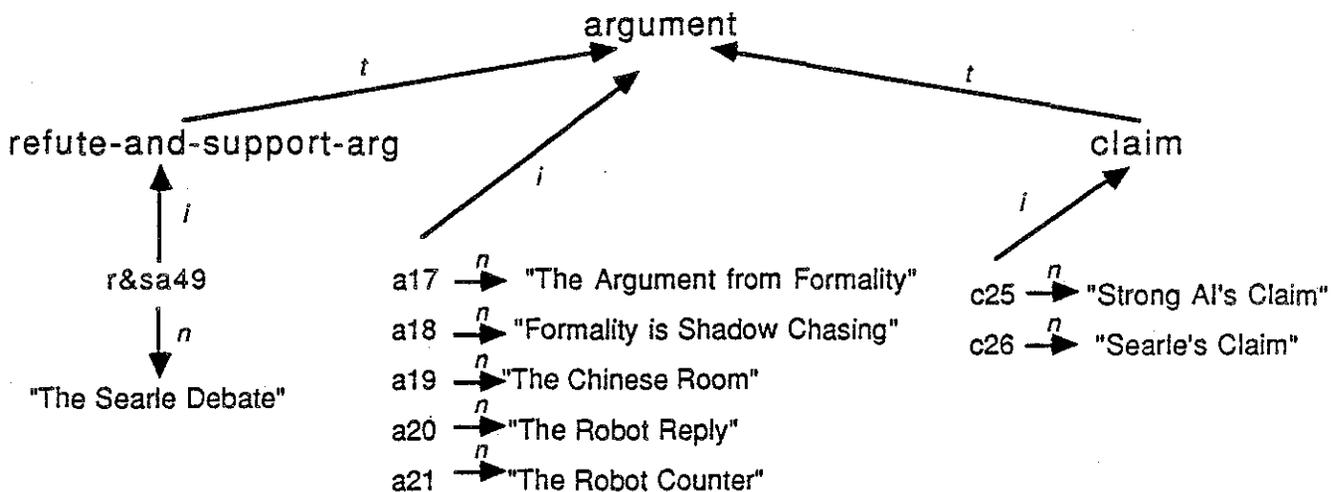


Figure 12. A very simplified sample EUCLID screen to illustrate the constraint-based approach to hypertext.

database



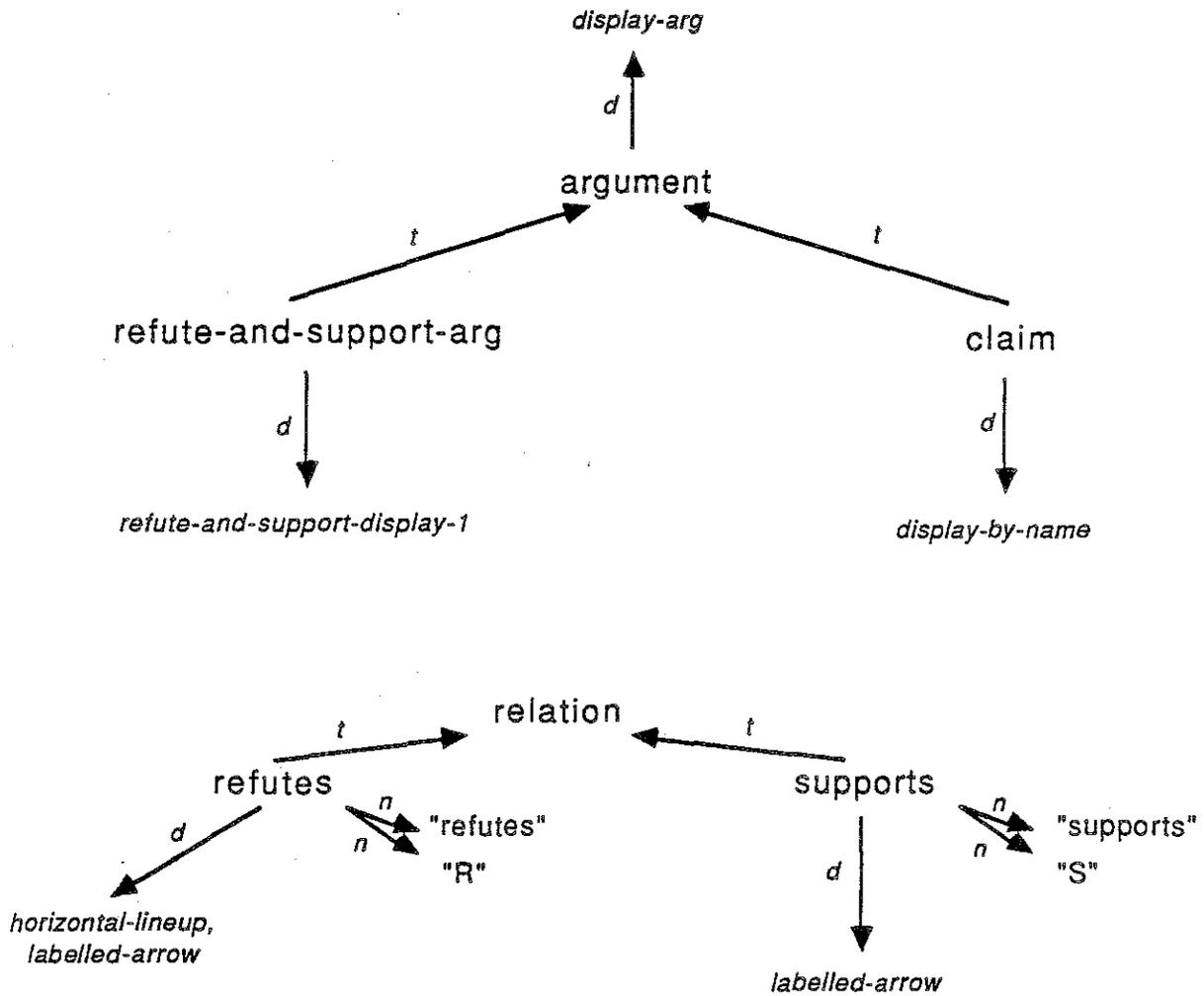
relations: t = type-of n = has-name i=instance-of

active perspective

r&sa49, a17 - a21, c25 - c26, s35 - s36, r11 - r13

Figure 13. The network database and the active perspective underlying Figure 12.

display hierarchy



relations:

d = display-constraint-schema

t = type-of

n = has-name

i=instance-of

Figure 14. The display hierarchy underlying Figure 12.

Display Constraint Schemata

```
display-by-name (X,Rect,Printstring) :=
  contains-entire-string(Rect, Printstring),
  name(X, content(Printstring)),
  font(Printstring) = default-font(type(X)),
  size(Printstring) approx=(3) default-size(type(X)) .

refute-and-support-arg-display-1 (A) :=
  vertical-stack(R5,[R1,R4]),
  horizontal-stack(R4,[R2,R3]),
  display-by-name(A,R1,Printstring),
  underlined(Printstring),
  display-arg(refute-side(A),R2),
  display-arg(support-side(A),R3) .

display-arg (A,R) :=
  vertical-stack(R0,[R1,R2,R3]),
  contains-entire-string(R1,Printstring1),
  name(A, contents(Printstring1)),
  display-claim(R2,conclusion(A))
  display(R3,arg-body(A)) .

vertical-stack (R,Rlist) :=
  rectangle(R),
  list-of(rectangle,Rlist),
  top(R) = top(first(Rlist)),
  bottom(R) = bottom(last(Rlist)),
  for-all(Ri,tail(Rlist),top(Ri)=bottom(predecessor(Ri,Rlist))) .
```

Figure 15. Examples of definitions of some of the display constraint schemata used in Figure 14.

necessary, and the specification of these strengths is part of the job of the constraint schemata (and these strengths are, of course, user-modifiable). Each soft constraint contributes a term to a sum S that defines the total degree to which the constraints are violated. The goal of constraint satisfaction, then, is to minimize S . Our initial approaches to this numerical optimization problem include both the stochastic technique of simulated annealing [Kirk83] and the deterministic technique of gradient descent.

It is important to note that in the hypertext setting, most of the constraint satisfaction problems that need to be solved are *incremental* problems. That is, the typical case is that a screen has already been laid out, and the user has just added an item to or deleted an item from the perspective; the set of constraints that now need to be satisfied differ very little from the set already satisfied, so the previous display provides a very good starting point from which to seek an optimum for the new S function.

Figure 11 includes a number of "editors" for modifying the various data structures in the CBH system. To illustrate how these work, imagine a user to have clicked on a displayed item for a menu of operations. Imagine the user selects *delete*. A second menu now appears, offering a number of different "deletions" available to the user: the selected item can be deleted from the perspective, so that it disappears from the screen but remains in the underlying database; or the underlying item can be deleted from the database; or the constraint schema for the selected item can be deleted from the current display hierarchy. These three forms of deletion call upon the functionality of the three editors: the perspective editor, the database editor, and the display hierarchy editor, respectively. (From the user's point of view, there is no need to distinguish the editors per se: there are simply a number of operations available through menus that alter the active perspective, the database, or the current display hierarchy.)

3. CONCLUSION

In this paper we have addressed the problems of controlling the interaction between users and hypertext, and of controlling the screen. We have suggested that user/document interaction be structured according to the particular discourse type characterizing the given material. We have described a hypertext system, EUCLID, for the support of reasoned argumentation. Interaction in EUCLID is being designed to reflect the structure of a particular discourse type, reasoned discourse. EUCLID rests on a semi-formal argument representation language, ARL, in which formal specification of structural information, for use by both user and machine, is combined with informal specification of content information, to be used only by the user. The EUCLID system assumes the burden of maintaining at all times a coherent, well-composed screen, in which complex structural relations among the underlying database items are represented by complex spatial relations among the displayed representations of those items. The great demands this places on control of the screen have led us to develop a general system for hypertext display in which displayed items contribute constraints on the display, and the display system composes a screen layout that satisfies the total set of constraints. The design of this constraint-based hypertext system CBH has been described; its implementation is now underway.

ACKNOWLEDGEMENTS

This research has been supported by NSF grant IST-8609599, a grant from Symbolics, Inc., and by the Department of Computer Science and Institute of Cognitive Science at the University of Colorado at Boulder.

REFERENCES

- [Aboc85] Acock, M. (1985). *Informal logic examples and exercises*. Belmont, CA: Wadsworth.
- [Alva85] Alvarado, S.J., Dyer, M.G., & Flowers, M. (1985). Memory representation and retrieval for editorial comprehension. *Proceedings of the International Joint Conference on Artificial Intelligence*.
- [Arti80] *Artificial Intelligence*. (1980). Special issue on non-monotonic logic. Volume 13, Numbers 1-2.
- [Bir82] Birnbaum, L. (1982). Argument molecules. *Proceedings of the American Association for Artificial Intelligence*.
- [Bir80] Birnbaum, L., Flowers, M., & McGuire, R. (1980). Towards an AI model of argumentation. *Proceedings of the American Association for Artificial Intelligence*.
- [Brow85] Brown, J.S., & Newman, S.E. (1985). Issues in cognitive and social ergonomics: From our house to Bauhaus. *Human-Computer Interaction, 1*, 359-391.
- [Cyph86] Cypher, A. (1986). The structure of users' activities. In D.A. Norman & S.W. Draper, Eds., *User centered system design*. Hillsdale, NJ: Erlbaum.
- [Enge80] Engel, S.M. (1980). *Analyzing informal fallacies*. Englewood Cliffs, NJ: Prentiss-Hall.
- [Flow80] Flower, L.S., & Hayes, J.R. (1980). The dynamics of composing: Making plans and juggling constraints. In L.W. Gregg & E.R. Steinberg (Eds.), *Cognitive processes in writing*. Hillsdale, NJ: Erlbaum.
- [Flow88] Flower, L.S., Hayes, J.R., Carey, L., Schriver, K., & Stratman, J. (to appear). Detection, diagnosis and the strategies of revision. *College Composition and Communication*.
- [Flow82] Flowers, M., McGuire, R., & Birnbaum, L. (1982). Adversary arguments and the logic of personal attacks. In W.G. Lehnert & M.G. Ringle (Eds.), *Strategies for natural language understanding*. Hillsdale, NJ: Erlbaum.
- [Foge82] Fogelin, R.J. (1982). *Understanding arguments: An introduction to informal logic*. New York: Harcourt, Brace, Javanovich.
- [Goul80] Gould, J.D. (1980). Experiments on composing letters: Some facts, some myths, and some observations. In L.W. Gregg & E.R. Steinberg (Eds.), *Cognitive processes in writing*. Hillsdale, NJ: Erlbaum.

- [Govi85] Govier, T. (1985). *A practical study of argument*. Belmont, CA: Wadsworth.
- [Greg80] Gregg, L.W. & Steinberg, E.R., Eds. (1980). *Cognitive processes in writing*. Hillsdale, NJ: Erlbaum.
- [Haye80] Hayes, J.R. & Flower, L.S. (1980). Identifying the organization of writing processes. In L.W. Gregg & E.R. Steinberg (Eds.), *Cognitive processes in writing*. Hillsdale, NJ: Erlbaum.
- [Hint69] Hintikka, K.J.J. (1969). *Models for modalities*. Dordrecht: Reidel.
- [Holt83] Holt, R.C. (1983). *Concurrent Euclid, the UNIX™ system, and Turis*. Reading, MA: Addison-Wesley.
- [Kell85a] Kellogg, R.T. (1985). Computer aids that writers need. *Behavior Research Methods, Instruments, & Computers*, 17, 253-258.
- [Kell85b] Kellogg, R.T. (1985). Why outlines benefit writers. Paper presented to the Psychonomic Society, Boston.
- [Kell88] Kellogg, R.T. (in press). Designing idea processors for document composition. *Behavior Research Methods, Instruments, & Computers*.
- [Kirk83] Kirkpatrick, S., Gelatt, C.D. Jr., and Vecchi, M.P. (1983). Optimization by simulated annealing. *Science*, 220, 671-80.
- [Lamp77] Lampson, B.W., Horning, J.J., London, R.L., Mitchell J.G., & Popek, G.J. (1977). Report on the programming language Euclid. *SIGPLAN Notices*, 12, Number 2.
- [Sear80] Searle, J. (1980) Minds, brains, and programs. *The Behavioral and Brain Sciences* 3, 417-457.
- [Smol88] Smolensky, P., Fox, B., King, R., & Lewis, C. (in press). Computer-aided reasoned discourse, or, How to argue with a computer. In R. Guindon (Ed.), *Cognitive Science and Its Applications For Human-Computer Interaction*. Hillsdale, NJ: Erlbaum. Also available as Technical Report CU-CS-358-87. Department of Computer Science, University of Colorado at Boulder. February, 1987.
- [vanD83] van Dijk, T., & Kintsch, W. (1983). *Strategies of discourse comprehension*. New York: Academic.
- [VanL85] VanLehn, K. (1985). Theory reformulation caused by an argumentation tool. Report. Xerox Palo Alto Research Center, Palo Alto, CA.
- [Zade79] Zadeh, L.A. (1979). A theory of approximate reasoning. In J.E. Hayes, D. Michie, & L.I. Mikulich, Eds., *Machine Intelligence 9*. New York: Wiley.

[Zade84] Zadeh, L.A. (1984). Syllogistic reasoning in fuzzy logic and its application to reasoning with dispositions. Technical Report 16. Cognitive Science Program, UCB, Berkeley, CA.

[Zuke85] Zukerman, I. & Pearl, J. (1985) Tutorial dialogs and meta-technical utterances. Manuscript. Computer Science Department, UCLA.

For proprietary reasons, the original, reviewed version of this paper has been withdrawn from publication. The authors, however, will present the original paper in the session called Argumentation. – Ed.

gIBIS: A Hypertext Tool for Team Design Deliberation

Jeff Conklin and Michael L. Begeman

MCC
Software Technology Program
3500 West Balcones Center Drive
Austin, Texas 78759-6509
ARPA: conklin@MCC.COM begeman@MCC.COM

ABSTRACT.

This paper introduces an application-specific hypertext system designed to facilitate the capture of early design deliberations, which implements a specific design method called Issue Based Information Systems (IBIS). The hypertext system described here, gIBIS (for graphical IBIS), makes use of color and a high speed relational database server to facilitate building and browsing typed IBIS networks. Further, gIBIS is designed to support the collaborative construction of these networks by any number of cooperating team members spread across a local area network. Early experiments suggest that the gIBIS tool, while still incomplete, forges a good match between graphical interface and design method even in this experimental version.

INTRODUCTION.

There is a growing recognition that hypertext is an ideal framework on which to base a support environment for the system design process. In the MCC Software Technology Program we have been working on a hypertext-based project called the Design Journal which is aimed at providing a team of system designers a medium in which all aspects of their work can be computer mediated and supported. This includes the traditional documents such as requirements and specifications, but it also includes designers' early notes and sketches and their design decisions and rationale. By design rationale we mean the design problems, alternative resolutions (including those which are later rejected), tradeoff analysis among these alternatives, and record of the tentative and firm commitments that were made as the problem was discussed and resolved. Our research has two thrusts: (i) to understand the structure within and between design decisions, and (ii) to address the interface problems inherent in capturing large amounts of informal design information and in providing effective methods for the indexing and retrieval of this information. As part of the latter thrust we have built a running prototype of the Design Journal called gIBIS, which is based on a simple model of design deliberation called Issue Based Information System, or IBIS.

THE IBIS METHOD.

The IBIS method was developed by Horst Rittel [RIT70], and is based on the principle that the design process for complex problems is fundamentally a conversation among *stakeholders* (e.g. designers, customers, implementors, etc.) in which they bring their respective expertise and viewpoints to the resolution of design issues. The IBIS model focuses on the articulation of the key Issues in the design problem. Each Issue can have many Positions, where a Position is a statement or assertion which resolves the Issue. Often Positions will be mutually exclusive of one another, but the method does not require this. Each of an Issue's Positions, in turn, may have one or more Arguments which either support that Position or object to it. Thus, each separate Issue is the root of a (possible empty) tree, with the children of the Issue being Positions and the children of the Positions being Arguments.

A typical IBIS discussion begins with someone posting an Issue node containing a question such as "How should we do X?" That person may also post a Position node proposing one way to do X, and may also post some Argument nodes which support that Position. Another user may post a competing Position responding to the Issue, and may support that with their own Arguments. Others may post other Positions, or Arguments which support or object to any of the Positions. In addition, new Issues which are raised by the discussion may be posted and linked into the nodes which most directly suggested them.

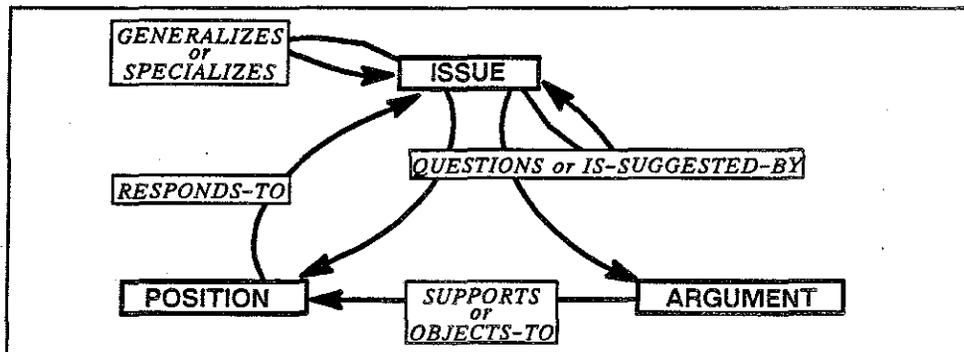


Figure 1: The set of legal rhetorical moves in IBIS.

THE gIBIS TOOL.

There were three technological themes guiding our design of gIBIS. The first was an interest in exploring the capture of design rationale [CON87b]. The second theme was an interest in supporting computer mediated teamwork, and particularly the various kinds of design conversations that might be carried on via networked computers, a la email or news [EVE86,HOR86]. Thirdly, we wanted an application in which we would have a sufficiently large information base to investigate issues regarding the navigation (i.e. search and browsing) of very large and loosely structured information spaces.

The gIBIS tool which emerged from these themes has the following features:

- **Integral Browser:** 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.
- **Context Sensitive Menus:** 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.
- **Multiple Access Paths:** 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.
- **Search and Query:** 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).
- **Multiuser Support:** 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.

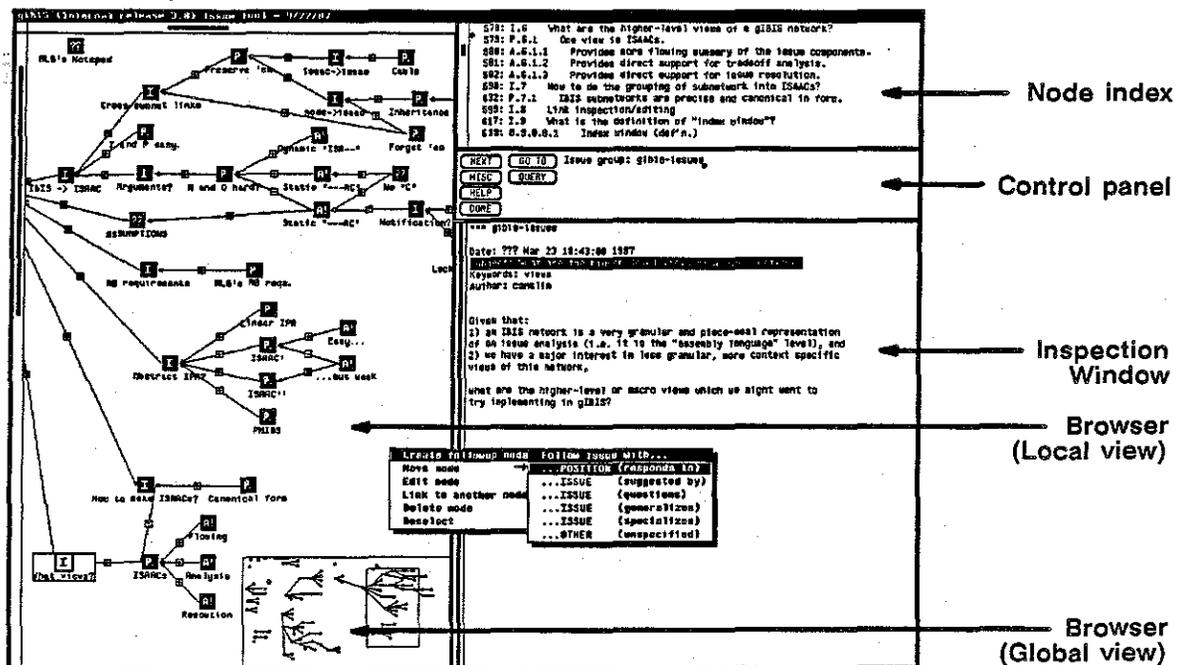


Figure 2: The gIBIS Interface.

OBSERVATIONS.

In the first seven months that gIBIS was available in the MCC Software Technology Program, 16 people created 21 issue groups containing a total of 1153 nodes and 1237 links. Based on this limited data we can make some preliminary observations, both positive and negative.

The Synergy of Tool and Method: The limited set of node and link types in IBIS and the use of color and hypertext in gIBIS seem to complement each other well. Users were generally enthusiastic about using the tool, and reported that it imposed a structure on discussions which exposed "axe grinding, hand waving, and clever rhetoric".

The Dangers of Premature Segmentation: One common but subtle difficulty in hypertext systems is that it is sometimes unnatural to break ones' thoughts into discrete units, particularly when the problem is not well understood and those thoughts are vague, confused, or shifting. With gIBIS this effect is pronounced, because the IBIS method imposes a rather austere selection of node and link types on the user. We are exploring composite node structures which will bring an Issue's deliberative text into a single node, smoothing the flow of ideas within Issue-based discussions.

A Problem with Context in Non-linear Documents: Traditional linear text provides a continuous, unwinding thread of context as ideas are proposed and discussed – a context which the writer is directly, if unconsciously, constructing to guide the reader to the salient points and away from the irrelevant and distracting ones. Indeed, a good writer anticipates the questions and confusions that the reader may encounter, and carefully crafts the text to prevent these problems. The hypertext (or at least gIBIS) author, however, is being encouraged to make his or her points discrete, and to separate them from their context. Even the careful author is in danger of not anticipating all the various routes by which a reader may reach a given node, and so may fail to sufficiently develop the context necessary to make the node's contents clear, if not compelling. We have as yet found no solution for this problem.

Annotative or "Meta" Discussions: In IBIS discussions there is sometimes a need for a meta-discussion when a participant in an issue group feels that someone has poorly or inaccurately used the IBIS structure to present their ideas. In fact, it has been noted that there are three levels of description for collaborative work: substantive (the content of the work), annotative (comments about substance), and procedural (comments about procedures and conventions for use of the medium) [TRI86]. In our framework, all three levels can be discretely represented by Issue-based conversations.

Macro-level Organization of the Browser Space: One of the "hot issues" in hypertext research is the problem of the effective use of a graphical browser to navigate in networks that have more than a few dozen nodes. This is linked to the more general problem of disorientation [CON87a], but bears particularly on the visual and spatial aspects of disorientation in a large data space. The gIBIS browser ran into these difficulties as well. The global view and mechanisms mentioned above have helped to reduce this problem significantly.

CONCLUSIONS.

We have briefly described the IBIS method, the gIBIS tool, and some preliminary observations about the use of the tool. Our experiments with gIBIS are informing our theory about the structure of design decisions and design rationale, and are providing us with important insights about the design of the Design Journal, a hypertext-based environment for system engineering which we will continue to design, prototype, test, and transfer into our shareholders' development environments.

REFERENCES.

- [CON87a] Conklin, J. "Hypertext: a Survey and Introduction". *I.E.E.E. Computer*, Vol. 20, No. 9, September, 1987.
- [CON87b] Conklin, J., "The Capture of Design Rationale". Paper in progress, to be MCC TR in fall of 1987.
- [EVE86] Eveland, J. and Bikson, T. "Evolving electronic communication networks: an empirical assessment." *Proc. CSCW'86: MCC/ACM conference on computer-supported cooperative work*. 1986.
- [HOR86] Horton, M. and Adams, R (Center for Seismic Studies, Arlington, Va.). "How to read the network news." Distributed by Mr. Adams quarterly over the USENET news network.
- [RIT70] Rittel, H. and Kunz, W. "Issues as elements of information systems." *Working paper #131. Institut fur Grundlagen der Planung I.A.* University of Stuttgart.
- [TRI86] Trigg, Randall, Lucy Suchman, and Frank Halasz, "Supporting Collaboration in NoteCard.," *Proceedings of CSCW '86: the Conference on Computer-Supported Cooperative Work*, MCC/STP, Austin, Texas, December, 1986.

Exploring Representation Problems Using Hypertext

Catherine C. Marshall

Xerox Special Information Systems
250 N. Halstead Street
Pasadena, CA 91109

ABSTRACT

Hypertext is a technology well-suited to exploring different kinds of representational problems. It can be used first as an informal mechanism to describe the attributes of objects and to capture relationships between the objects. Then hypertext structures can be constrained into a more formal representation of a domain, model, or analytic technique. A range of strategies for using hypertext can be employed to describe a problem and converge on an appropriate representation; competing representations can be informally evaluated to compare their relative expressive power.

This paper discusses several applications that have used NoteCards, a hypertext idea processing system, to tackle representation problems. Examples from each problem domain have been collected using the hypertext system as the initial acquisition vehicle. Subsequent analysis using hypertext structuring tools has revealed the semantics of each problem domain enabling the development of competing representations. Abstraction of the structure and form of these representations can be used to guide system extensions. These tailored extensions support the evaluation of a representation's relative merits; the representation that has been developed in response to a particular problem can be applied to analogous problems to determine the limits of its scope.

The first application described in this paper models a type of policy decision-making process; the second looks at approaches to representing the logical structure of an argument; and the third suggests some methods for capturing the structure of a political organization as an alternative to a conventional database design. The applications are discussed in terms of the issues they raise and the trade-offs they involve, how hypertext-based tools have been used to exploit the representations, and the solutions and techniques that have been developed in the process of creating each representation.

INTRODUCTION

Hypertext systems are generally aimed at the needs of authors, designers, on-line readers, and other users performing idea processing tasks, but they also meet the needs of many

representation tasks directed at producing a formalism. Hypertext systems provide tools to view and manipulate structure as well as content, thus supporting the move from textual descriptions to emergent forms and abstract structures. The ability to work with unstructured information in conjunction with formalized, systematically organized information is the chief advantage in using a hypertext system rather than a knowledge representation language or a database description language. Systematic structures and expressions of content can be introduced and manipulated without the constraints of a formalism. Examples can be collected and analyzed, and structure can be created and imposed as general patterns are understood.

The representation process parallels the principal activities comprising idea processing, acquisition, analysis, and exposition [Hala87]. The acquisition phase of the representation process involves collecting examples from the domain, and possibly organizing them in some rough conceptual framework. The analysis phase involves discovering which features describe an object, what relationships exist between objects, and how objects can be categorized. Thus node contents, network structure, and a classification system are represented. The analysis process produces an abstract form of the representation that may then be refined as flaws are identified and missing features are noted. The exposition phase uses the product of the analysis phase, the representation, in an application to demonstrate its effectiveness. The representation can be applied to analogous problems by instantiating the formalism with elements of the new problem. Figure 1 summarizes a hypertext approach to creating a representation:

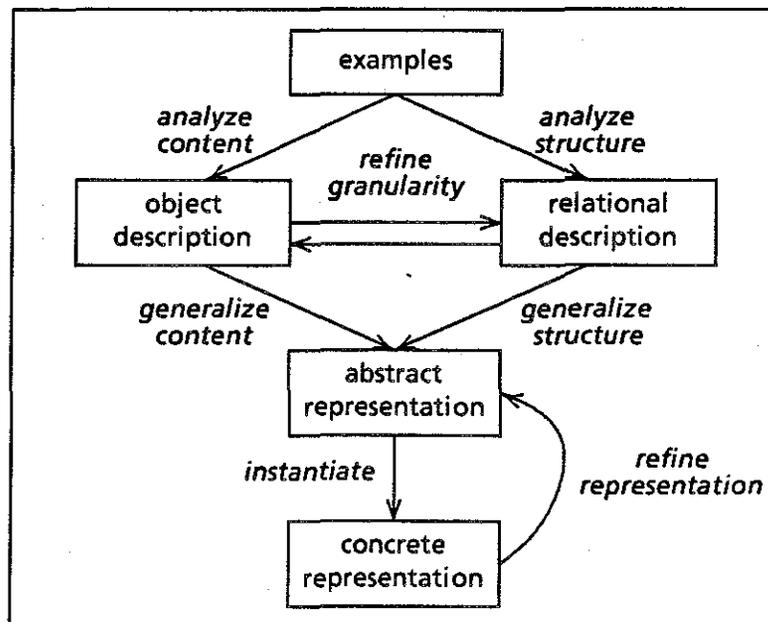


Figure 1. A model of the representation process.

NoteCards is a hypertext system that has been under development at Xerox's Intelligent Systems Laboratory for the past four years [Hala87]. Because it was designed to meet the needs of idea processing activities, it has served as an interesting basis for conducting

investigations involving representation problems. This paper describes the salient features of the NoteCards system, then discusses three different representation tasks that have used NoteCards (and, more generally, hypertext concepts) to support the processes entailed by the tasks. The applications bring up certain important issues and trade-offs, and suggest activities that hypertext can support. Specific features necessary for each task are also discussed.

SALIENT FEATURES OF THE NOTECARDS HYPERTEXT SYSTEM

The central construct in the NoteCards system is a semantic network that consists of electronic *notecards* as the nodes connected by *typed links*. Each notecard contains an arbitrary amount of information embodied in text, graphics, images, or some other editable substance. Each link designates a specific relationship between two notecards; the relationship may be either user or system defined. The system provides two specialized types of cards, *browsers* and *fileboxes* to help manage networks of cards and links. A browser contains a structural map of a network of cards; it may be system-generated or user-created. Browsers are both a presentation tool and editing mechanism. Fileboxes can be used to cluster and organize collections of notecards, yielding a hierarchical filing structure managed by the system. NoteCards also includes a set of protocols and functions for creating new types of cards and manipulating the information in the network. This applications-oriented extensibility plays an important role in preserving and formalizing a representation so it may be applied in other situations.

Figure 2 illustrates a sample NoteCards screen display. It includes a text notecard with a link to a second text notecard, a map with links inserted in it, and a browser of a relevant portion of the network. The map is drawn on a sketch card that is based on an object-oriented graphics editor. Sketch cards are used extensively in the applications discussed below.

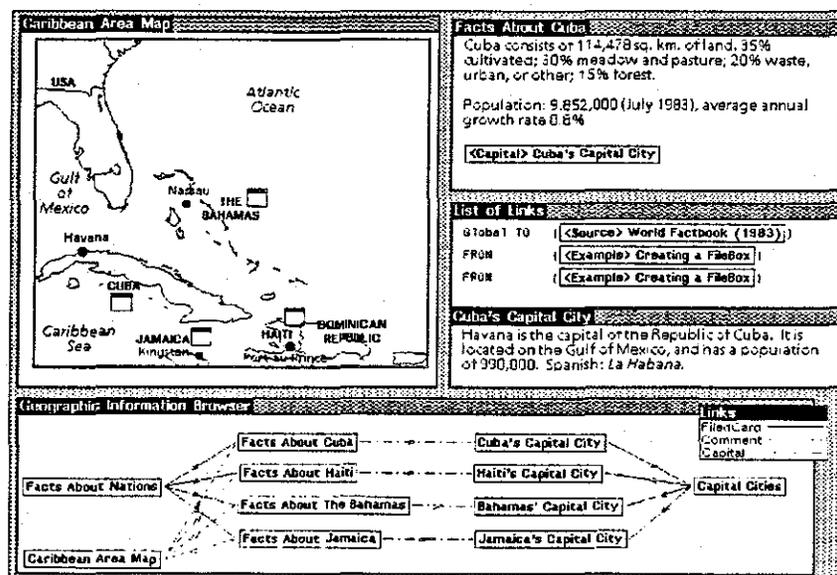


Figure 2. Sample NoteCards display.

USING NOTECARDS AS A VEHICLE FOR DEVELOPING REPRESENTATIONS

This section describes three application areas in which NoteCards has been used as the vehicle for capturing a representation in the context of a specific domain and problem. The first two applications, the Rational Actor Model for policy decision-making and the template for representing the logical structure of a legal argument, are both part of a more general investigation into the structures and processes of argumentation. The third application is more of a proof-of-concept problem; the representation created here was to demonstrate how hypertext could provide leverage in creating shared information resources. Each application is presented in terms of initial problem (what the representation is supposed to capture and a little bit of background), approaches used, and results. The abstract techniques embodied by each representation are intended to be sufficiently general that they can be applied to other problems.

Application 1: A Rational Actor Model for policy decision-making

Analyses of international events are often framed within a tacit Rational Actor Model to explain the logic of an action and justify the corresponding response or policy decision. This application defines a hypertext model to represent rational actor analysis; the template developed for analysis and decision-making is intended as a general framework for performing similar tasks. It is a vehicle for evaluating competing explanations and arguments about the consequences of any particular course of action. The Rational Actor Model assumes that the behavior of a nation can be viewed in terms of a monolithic, rational individual with a unified purpose and motivation and that the intricate processes within a government can be ignored. Interaction between rational actors occurs in a chess-like move/countermove framework. Thus, to perform a rational actor analysis in the NoteCards framework, the analyst adopts the role of the decision-making actor.

The examples for the NoteCards representation of the rational actor template are taken from Allison's multi-model analysis of the Cuban missile crisis [Alli71]. The initial part of the representation process involved simply moving Allison's outline of explanations and decision options into a hypertext form. In this case, text cards were used to collect verbal descriptions of the rational actor analysis. Cards were created for each of five hypotheses explaining possible Soviet goals in deploying the missiles in Cuba. Evidence for each hypothesis was also collected in individual cards and linked to the supported hypothesis. This structure soon proved to be too weak to represent the type of argument that was being employed in the source text.

In Allison's analysis, the arguments for and against each of the explanations are based on the other options the Soviets could have chosen to achieve that particular goal. The explanations of how certain options meet specific goals are framed not only in the context of the situation, but also in the constraints imposed by the Rational Actor Model. This form of explanation suggested that a representation be developed to use this notion of evaluation of goals with respect to options as constrained by the model. VanLehn's matrix summary cards [VanL85]

inspired the development of a similar type of matrix for use in this application. By using options and goals as rows and columns, a framework is provided for evaluating the explanations and the entire argument is given a structure. Figure 3 shows an abstract form of an Option/Goal matrix; each row is labeled with a different option, and each column is labeled with a goal. The cells of the matrix contain a semi-qualitative assessment of how well an option supports the corresponding goal:

	G1	G2	G3	G4	...
O1	+	-	0	+	
O2	0	-	+	0	
O3	0	+	-	0	
⋮					

Figure 3. An Option/Goal matrix.

In its NoteCards implementation, both the cells of the matrix and its row and column labels are links to supporting argument networks and description cards. To represent how an individual hypothetical goal is tested, a matrix like the one shown in Figure 4 is constructed. Testing an individual hypothetical goal involves weighing it against each of a set of plausible options. The options are obtained by introducing slight variations into the character of the situation. For example, in the case of the Cuban missile crisis, the Soviets chose to deploy MRBMs, IRBMs, and troops in Cuba. The character of the deployment should thus be varied to yield some other hypothetical options (for example, deploy a small number of MRBMs in Cuba or deploy Soviet troops in Cuba) and the location of the deployment can be varied (Berlin instead of Cuba) to explore that characteristic of the option chosen. Constraints introduced by the Rational Actor Model (such as "Avoid risk" or "Minimize cost") are also evaluated with respect to the options in the context of the hypothetical goal of the action. Figure 4 shows the Option/Goal matrix evaluating the hypothetical goal of Probe, an abbreviation of what Allison refers to as "Hypothesis Four: Cold War Politics."

Options/Consequences	Goal Set defining Possible Worlds				Σ
	Test Goal	Static Goals			
	HG: Probe	SG: Avoid risk	SG: Minimize cost	SG: Obey constraints	
O/C1: Defensive Missiles in Cuba	<->	<0>	<0>	<0>	<->
O/C2: MRBMs, IRBMs, Troops in Cuba	<+>	<0>	<->	<->	<+>
O/C3: Troops in Berlin	<+>	<->	<->	<->	<0>
O/C4: Small # of MRBMs in Cuba	<+>	<0>	<+>	<0>	<+>
O/C5: Troops in Cuba	<NA>	<NA>	<NA>	<NA>	<NA>

Figure 4. Option/Goal matrix evaluating a hypothetical goal.

In this representation, links are used to carry an assessment of value; they are semi-quantitative measures of argument strength. The sigma cell values combine the evaluation of the claim supporting a particular option with regard to the hypothesized goal (in the Test Goal column) with any negative values of the links in the row for that option, which

indicate that Rational Actor constraints (referred to in Figure 4 as Static Goals) have been violated. For example, in Figure 4, the option "O/C4: Small # of MRBMs in Cuba" is assessed as strongly supporting the hypothetical goal of probing American intentions in the context of Cold War politics; a value of <+ +> has been assigned to the appropriate cell. Since no constraining Rational Actor goals have been violated, the value in the sigma column for O/C4 is also <+ +>. But in the case of option "O/C2: MRBMs, IRBMs, Troops in Cuba," the very positive assessment of the hypothetical goal has been modified downward in the sigma column by the violation of cost minimization and ideological constraint satisfaction goals. The level of constraint violation was relatively small, so the sigma value for O/C2 has been lowered a qualitative step to <+>. In this case, assessment of the sigma values has been subjective; however, eventually the heuristics for combining values will be made explicit so the sigma column can be computed automatically.

The top level Option/Goal Matrix, the Possible Worlds Matrix shown in Figure 5, is constructed from the set of completed lower level Option/Goal matrices. Its rows match the option range of the lower level matrices and each of its columns represents a goalset combining a hypothetical situational goal with the set of constraining static goals. The values in its cells can thus be taken directly from the sigma column of the corresponding matrix for evaluation of a single hypothetical goal. For example, in the matrix shown in Figure 5, the column under "GS: Probe" is taken from the sigma column in the matrix shown in Figure 4.

Interpretation Hypothetical Options	Possible Worlds				
	GS: Berlin	GS: Coercion	GS: Defense	GS: Probe	GS: Power
O/C1: Defensive Missiles in Cuba	<0>	<+>	<+>	<->	<0>
O/C2: MRBMs, IRBMs, Troops in Cuba	<->	<->	<->	<+>	<+ +>
O/C3: Troops in Berlin	<NA>	<->	<NA>	<0>	<NA>
O/C4: Small # of MRBMs in Cuba	<+ +>	<+ +>	<0>	<+ +>	<+>
O/C5: Troops in Cuba	<->	<->	<+ +>	<NA>	<NA>

Figure 5. A Possible Worlds Option/Goal matrix.

To interpret the Possible Worlds matrix to isolate the goal associated with the most likely explanation, the row of cells corresponding to the opponent's action is used. In the case of Figure 5, the actual Soviet action is described by "O/C2: MRBMs, IRBMs, Troops in Cuba." In that row of cells, the most positive value corresponds to the hypothetical goal addressing Soviet strategic inferiority in the missile race, "GS: Power." Therefore, the winning argument is associated with that hypothetical goal, and it can be taken to represent the most likely possible world. An explanation and counterargument mechanism can be driven off of this matrix to produce explanations like, "If rational actor RA had wanted to satisfy hypothetical goal HG embedded in goalset GS, then it would have acted according to option O/C." For example, in the case of Figure 5, for the hypothetical goal embedded in "GS: Defense," an explanation could be built by finding the most positive cell value in its column, and instantiating in the form given above: "If the Soviets had wanted to satisfy the goal of Cuban

defense (GS: Defense), then they would have deployed Soviet troops in Cuba (O/C5: Troops in Cuba)."

The chief advantage of using this matrix layout to organize the Rational Actor analysis is that it is a very compact representation for evaluating competing arguments. The Possible Worlds matrix shows a range of scenarios that account for the Soviet action, as well as an assessment of their relative plausibility. Because the cells are links to a claim structure, a NoteCards browser can be used as the vehicle for examining competing arguments and evaluating the strength of the underlying structures.

The representation used for claim structures imposes the requirement that each leaf card of the claim network be linked to the network by either an *Evidence* link or an *Assumption* link. Figure 6 shows an example of a claim structure; it refers to a claim supporting the cell in Figure 4 corresponding to the option "O/C2: MRBMs, IRBMs, Troops in Cuba" and the hypothetical goal "Probe." The solid lines represent Evidence links; the dashed line represents a ++ link to the subclaim. It has been chosen as an example since it is relatively simple; many of the claim structures constructed in representing Allison's rational actor analysis of the Cuban missile crisis are far more complex. This claim structure representation also allows the network to be browsed to pinpoint evidence dependencies.

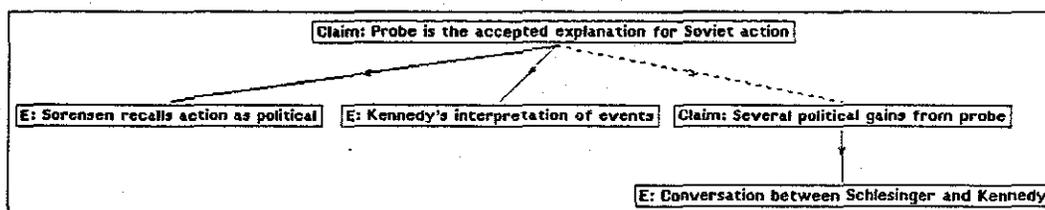


Figure 6. A simple example of a claim structure underlying a value link.

Several unexpected results arose from the assumption/evidence requirement on claim structures. (1) The need to identify assumptions helped to tease out some implicit characteristics of the Rational Actor Model. The ability to explicitly identify model dependencies may facilitate the process of applying this representation to the problem using a different analytic model, for example Allison's Governmental Politics Model. (2) Because there were so many examples of both evidence and assumptions, various categories of each became obvious. This observation led to the construction of a hierarchical framework of fileboxes to express a taxonomy of evidence and assumptions. Link types could also have expressed this taxonomy; for example, *Assumption* links could have become *Model Assumptions*, *Domain Assumptions*, and *Local Assumptions*. However, it was inconvenient to expand the existing set of link types since this application had already put much of the burden for value assessment on the link types.

Application 2: Logical argument structures

The second application investigates the logical or deep structure of an argument as part of a multi-level analysis of argumentation. This analysis of an argument's logical structure is

framed in the legal domain. Specifically, transcripts of the oral arguments from several Supreme Court cases are examined to arrive at a representation that captures individual micro-arguments and how they interact. The representation identifies legal concepts and organizes the separate concepts into an issue-level argument.

The representation of a micro-argument is based on Toulmin's argument layout [Toul58]. A Toulmin argument layout consists of six elements: a claim, which is an assertion about a specific entity or a central thesis; a qualifier, which may modify the claim; a datum, which is a fact in support of the claim; a warrant, which is a rule or principle governing the datum/claim relationship; a backing, which is a case, model, or other support for a warrant; and a rebuttal, which is a counterclaim that may function as a claim in another argument. In this application, the Toulmin structure has been simplified by omitting the qualifier, so the basic structure looks like this:

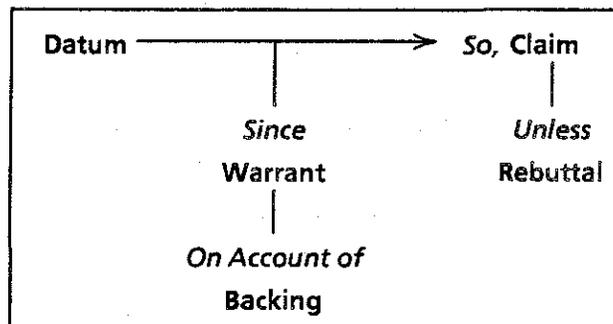


Figure 7. A Toulmin structure.

Arriving at a NoteCards representation of this structure took several iterations despite its apparent simplicity. The examples used to drive the development were taken from a California Supreme Court case, *People v. Carney*, which revolves around the issue of whether the warrantless search of the defendant's motor home was justified by the so-called "automobile exception" to the Fourth Amendment requirement and the right established by Article I, section 13 of the California Constitution. In particular, the primary argument that motor homes are included in the protection given by the Fourth Amendment was used as an example. This argument's datum is "Motor home X is being used as a home," and its claim is "Motor home X cannot be searched without a warrant."

From Figure 7, three initial requirements are placed on the representation. First, the spatial layout of the argument is important to the comprehension of the content. The spatial layout includes the linguistic cues, *so*, *since*, *on account of*, and *unless*, since they are key to understanding how the logical structure maps to a surface structure of connected sentences. Second, the relationships between the argument elements, the datum, claim, warrant, backing, and rebuttal, should be expressed in some way. Third, there needs to be a way to represent connections between elements in different micro-arguments. For example, two arguments might share a warrant, or a claim in one argument might be a datum in another

The first attempt at representing a basic Toulmin structure used the NoteCards browser card. The browser seemed to be an apt candidate since it shows a graph of the network, provides facilities for editing structure and creating new nodes, allows relationships between nodes to be specified and differentiated on the display, and can be saved in the database. Figure 8 shows the browser version of an argument card. Each node corresponds to a card; the card's title is shown as the label for the node. A link legend appears on the upper right hand corner of the browser. Labels have been added to show the linguistic cues.

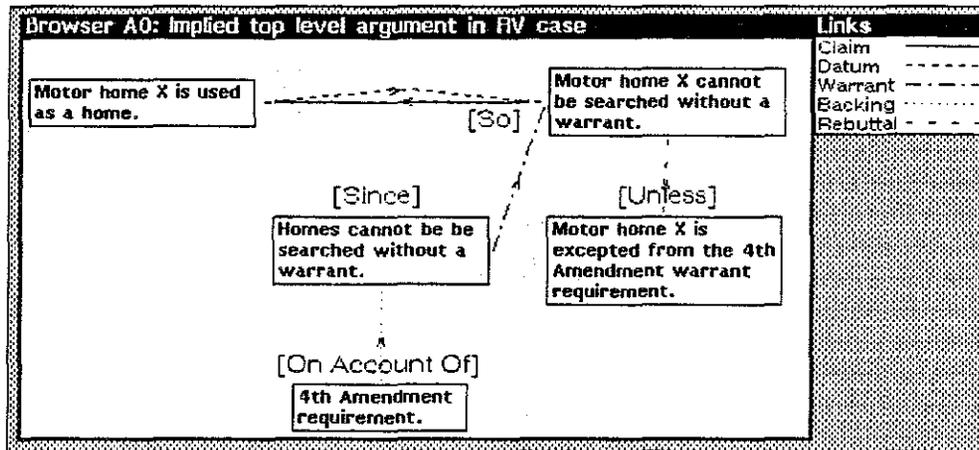


Figure 8. Argument card as browser.

Not only does the appearance deviate from the ideal layout shown in Figure 7, but also the semantics of the argument become muddled. There are directed links in both directions to indicate the relationship between datum and claim; this structure seems to be an adequate reflection of the Toulmin layout's semantics. The relationships between rebuttal and claim and backing and warrant are similarly straightforward, but the warrant relationship actually annotates the links between datum and claim, not the claim itself. There is no way to make a link from a card to another link in NoteCards. The other semantic problem with this representation concerns how the elements are related to the argument. The nodes in a browser in fact represent link icons whose destination is the card shown by title. Therefore, the link from the argument card to its constituent elements is of type *BrowserContents*, a system-supported link type that is essentially meaningless in the context of this application.

With these criticisms in mind, a sketch-based intermediate representation was tried and rejected, primarily because the editing functionality was inappropriate for the task. However, the link semantics seemed to match the problem domain and were carried over to the third representation of an argument card; this time the representation was based on a simple node-link graph, managed by the user, not the system. A graph-based argument card is shown in Figure 9.

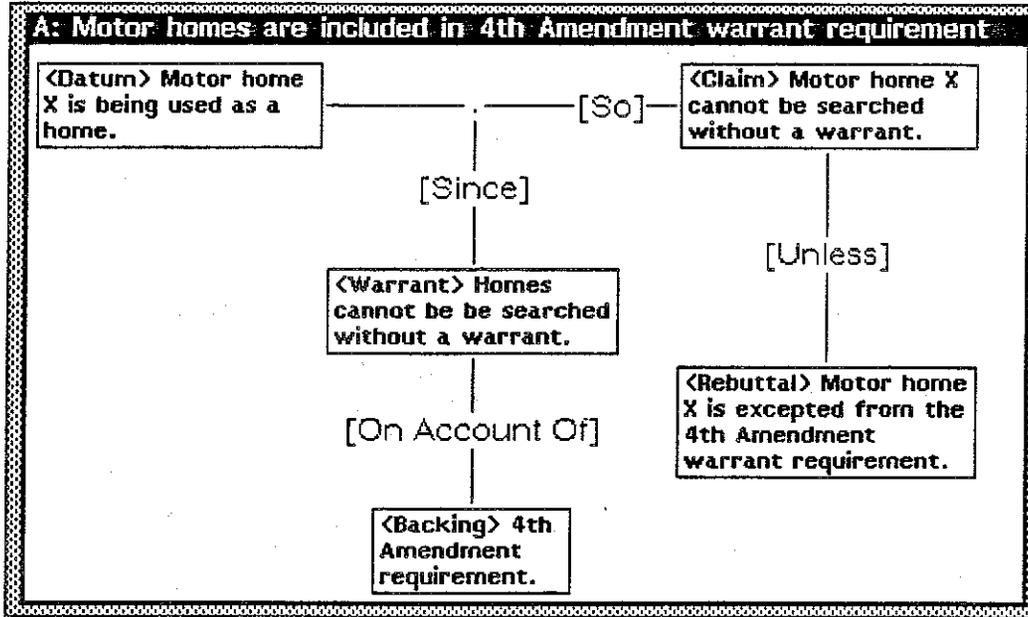


Figure 9. A graph-based argument card.

In this variation, the link that connects the argument with its constituents shows the role each constituent plays in the context of the individual micro-argument. Link types are shown as part of the link icon in this display - the icons are of the form *<link type>destination card title*. Hence, a particular constituent can play a different role in another argument (most often, a claim becomes a datum in another layer of logical argument or a warrant is tested with a new datum). Note that there are two kinds of nodes in this graph card: link icons to connect arguments with constituents (such as *Datum* or *Warrant*) and labels that annotate the graph with linguistic cues (such as *Since* or *Unless*). The two types of nodes fulfill the first two representational requirements concerning spatial layout and relationships between argument elements. In addition, the title of the argument card now reflects the purpose of the argument, showing why motor homes are included under the Fourth Amendment protection. This change made it easier to organize the sixty or so micro-arguments that were found in the analysis.

The final requirement on the representation for micro-argument structure is that interargument relationships can be meaningfully exploited. Figure 10 shows a NoteCards browser, following the path from argument card to constituents, to the next set of argument cards that share these constituents, to the set of constituents for the second set of argument cards. This browser has been constrained to show only four levels of structure and the five types of argument links in both directions. It gives some indication of the pattern of argument construction used in the transcript. In practice, the browser was expanded further and helped to identify the two main concepts (actual function and mobility) used in the course of the argumentation process.

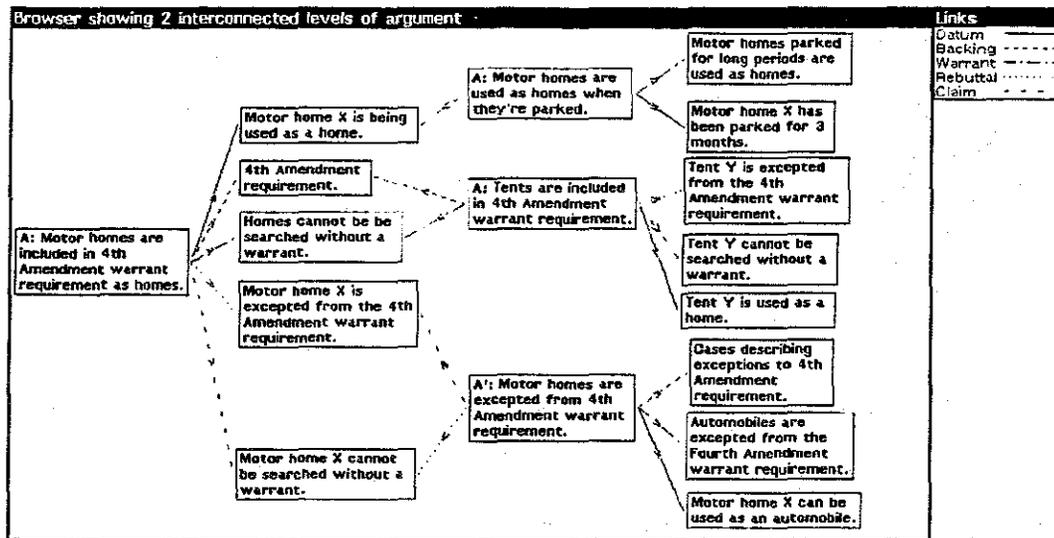


Figure 10. A browser showing two interconnected levels of argument.

Application 3: Socialist Republic of Vietnam Party Structure

At first glance, creating an information resource based on a description of the party structure of the Socialist Republic of Vietnam seems like a fairly conventional database problem rather than a hypertext application. But the problem proved to be amenable to this more "expensive" way of structuring information. First, a description of an organization combines highly structured information with loosely structured information. Second, the application realizes some benefit from using multiple representations of the same information. For example, an organization chart can supplement a hierarchical structure. Third, database extensions may involve information that does not conform to the original pattern. For example, the database may need to be extended by taking notes on news wires and in some way tying these unstructured notes to both individuals and organizations. Finally, the information is accessible from a number of different routes - some users may find it more convenient to work from an organization chart, while others may find that browsing the hierarchy conforms more closely to how they think about organizational structure. The complexity of having multiple representations is hidden from the user who opts for one information model over another.

This application is especially valuable in demonstrating the trade-offs between representing information as relational links or as hierarchical structures that cluster like entities. This trade-off first arises in the decision of whether to use the system-supported hierarchy of fileboxes and filed cards, or to design an organization chart scheme to provide context. In the latter case, the cards representing individual secretariat members are linked to the sketch-based organization chart with *Secretariat* links. Figure 11 illustrates the trade-off. In this application, both methods were used, since links are not a costly information structure in NoteCards.

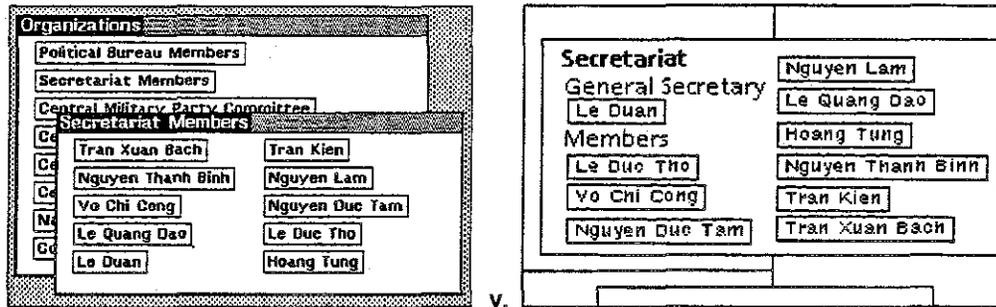


Figure 11. Hierarchical filing scheme v. spatial filing scheme.

The impact of these two representation schemes is reflected in Figures 12 and 13 showing the division of semantics in two browsers. The browsers have been designed to show the same information, namely which Political Bureau members belong to other Party organizations. In Figure 12, the information is conveyed largely by the way a card corresponding to an individual is filed in the hierarchy; information is carried only in the existence of links, not through their type. The original browser layout has been manipulated for the sake of compactness.

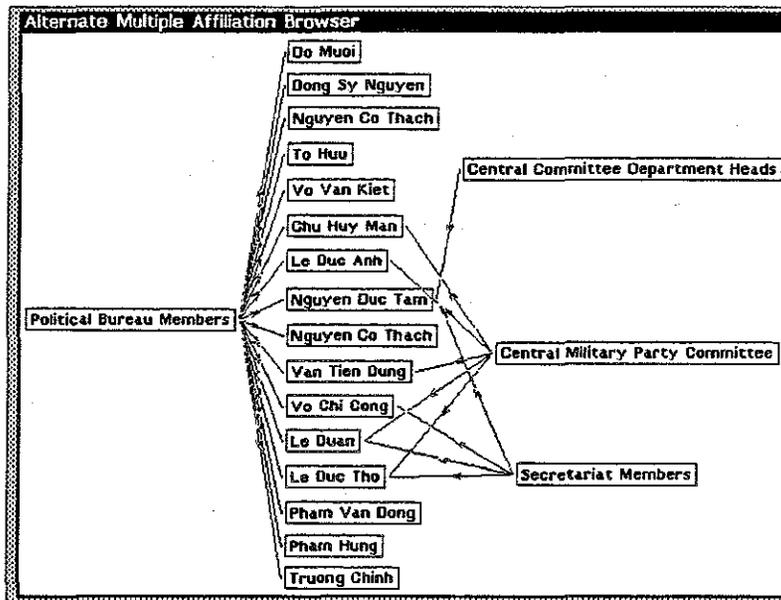


Figure 12. A portion of the browser generated from a filebox hierarchy.

In Figure 13, the semantic weight has been shifted almost entirely to the links. Note that the number of links connecting the Socialist Republic of Vietnam Party Structure with the various Political Bureau member indicates the extent to which they are multiply affiliated and the link dashing style shows the specific affiliations. Labels have been added to the browser to enhance its readability.

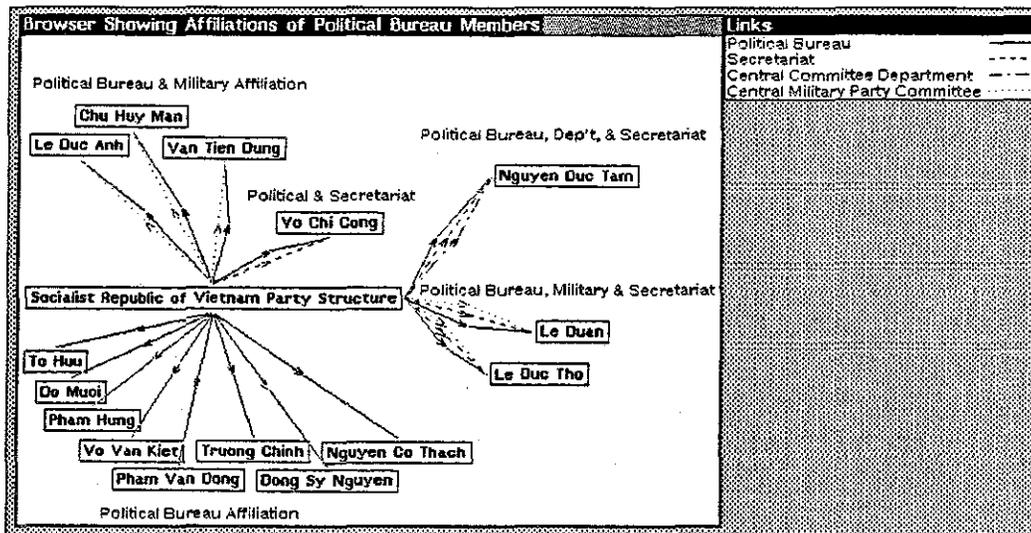


Figure 13. A browser generated from a network structure.

This application illustrates one of the most compelling reasons to use a hypertext system to address representation problems. Both of the organization methods discussed can be supported concurrently; the redundancy strengthens the representation rather than detracting from it.

ISSUES

Representation problems raise several important issues for the design of hypertext systems and for strategies for using hypertext. One recurrent issue in developing representations is choosing the most appropriate grain size for the individual objects. In NoteCards, this translates to deciding when to segment information in separate cards, and what information to use as the content of cards. For example, in the argumentation application discussed earlier, one crucial decision was whether a micro-argument defined a card that simply contained the statements corresponding to the argument elements (claim, datum, warrant, backing, and rebuttal), or whether the argument elements each needed to be on a separate card. This single-card scheme for representing a micro-argument could have easily been made to appear exactly as it does in Figure 9. In fact, in the initial phases of development, representing each element as a separate card seemed too fine a division of structure - after all, the only information each card held was its title. Yet this division turned out to be necessary to meaningfully represent the relationships between micro-arguments. Then too, a structure was eventually developed for the contents of the argument constituent cards, so the grain size proved to be appropriate.

In many representation problems, a key decision centers around the distribution of meaning - should links or cards carry the semantic burden. Sometimes the decision is forced by the node-link implementation in the hypertext system. For example, in NoteCards links can't annotate other links; this limitation helped to determine the form of the micro-argument representation discussed earlier. In the representation of the Vietnam Party structure, the

trade-offs were such that multiple distributions of semantic weight were developed. More specifically, the decision in NoteCards often centers around whether to use the system-maintained hierarchical filing structure, or whether it is necessary to use a network of user-defined links, thus placing the semantic burden more heavily on link types. Sometimes the distribution of meaning between link types and nodes is driven by the focus of the application. Performing a representation task or interpreting the resulting of an analysis becomes confusing if link types are used for too many semantically orthogonal purposes.

One of the most interesting representational issues centers around using spatial layouts in support of a structuring scheme. All three applications described in this paper use either sketch cards, graph cards, or browsers as the focal means of representing structure. The Rational Actor Model relies on sketch card-based matrices to show relationships between options and goals and to create a framework for expressing competing arguments. In the second application argument cards use a graph to provide spatial context for the constituent elements of a Toulmin structure and browsers to show interargument relationships. The third application uses a literal representation of an organization chart (based again on a sketch card) to provide a spatial filing system for the cards describing individual Party members. In these and other applications, a graphic representation of structure has been quite effective in organizing information and promoting comprehension.

The issue of distribution of meaning between links and nodes ties in with the representation of quantitative or semi-quantitative values. Value links as implemented in the Rational Actor Model show the strength of a relationship, but this assessment could have been captured using a destination card property as shown in Figure 14. The problem with this strategy is that the card is then limited to one role and one context - if the claim were applied in another situation, the strength of the support it offered might be far less. A better solution to the assignment of value to links is to make value a link property, but links properties are not yet supported in NoteCards.

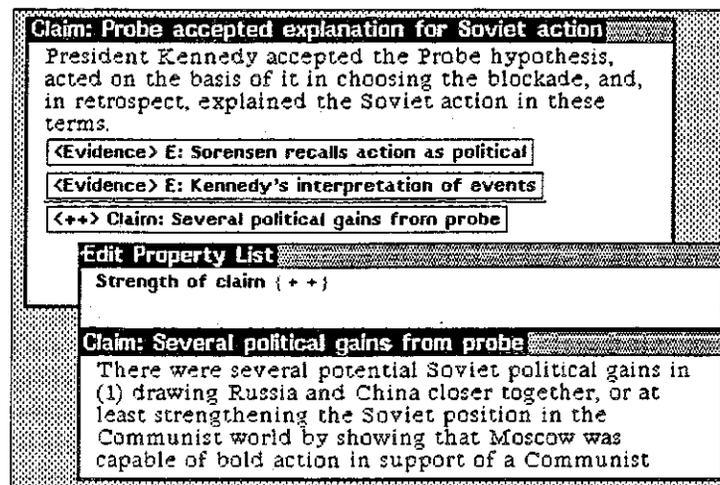


Figure 14. Moving the semantic burden from a value link to a destination card.

The final issue in representation development is capturing the exploration of a problem in a concrete structure so that it can be reapplied. In NoteCards, this may mean creating a new card type, or it may mean performing some other type of system tailoring through the programmer's interface [Trig87]. In many cases it means that constraints must be imposed on the kinds of links that can be anchored in a given card; it also means that the form (and perhaps content) of the substance of the card types used in the abstract representation must be defined. This process of "casting a representation in concrete" is facilitated somewhat by the protocols and functions provided in NoteCards, but could be given considerably greater assistance. The template cards developed for the Instructional Design Environment (IDE) [Russ87] are one instance of how abstractions can be described in NoteCards.

CONCLUSIONS

Hypertext meets many of the needs imposed by a representation task. First, hypertext systems are generally based on a semantic network of nodes and links, where the nodes can contain arbitrary content. Semantic networks are a convenient mode of representation for a variety of applications. Second, hypertext systems frequently provide tools for viewing and manipulating this network - browsing seems to be a ubiquitous requirement in establishing the viability of a particular representation. Furthermore, it is often most convenient to interact with a network as a structure; structure editing capabilities are ideally provided by a hypertext system. Third, hypertext systems can provide a way of abstracting and preserving a representation so it may be reapplied and possibly generalized or refined. Fourth, and most importantly, hypertext systems support the emergent properties of the representation process.

The flexibility of a hypertext system is exercised in performing a representation task. In the course of developing a representation, a user may need to switch the granularity of the representation to reflect a growing understanding of which information should be embedded in the content of a node, and which should be made explicit in the network. It is also necessary to balance the semantic weight between network nodes and links. Similarly, the user needs to be able to move between graphic and hierarchical organizing schemes. Often the system must be sufficiently flexible to support multiple representations simultaneously.

One important way in which hypertext systems can be extended is in their capacity to aid in the formalization process. First, patterns must be recognized so they can be abstracted; an extended hypertext system may support this activity. The formalization process includes a certain amount of system tailoring and imposing of constraints, but it also includes checking consistency and restructuring existing work in a new framework. Once the system has been tailored and the necessary support provided for the representation, subsequent work can adopt the new scheme, but additional system support is required to enforce it on earlier work.

A second way hypertext systems can be extended is by facilitating programmatic interaction with user-created structures and substances. For example, if a representation is used by an expert system or some other type of inference engine, it may be necessary for the program to interact with the content of a node or object to extract the knowledge inside it. Since hypertext

is a good vehicle for capturing and structuring knowledge, programmatic interpretation of a hypertext network appears to be a logical extension.

REFERENCES

- [Alli71] Allison, G., *Essence of Decision: Explaining the Cuban Missile Crisis*, Little, Brown and Company, Boston, Mass., 1971.
- [Hala87] Halasz, F.G., Moran, T.P., and Trigg, R.H., "NoteCards in a Nutshell," *CHI + GI '87 Conference*, Toronto, Canada, April, 1987.
- [Mars86] Marshall, C., "Representation of a rational actor model in NoteCards," Xerox Special Information Systems (internal report), Pasadena, California, July, 1986.
- [Russ87] Russell, D.M., Moran, T.P., and Jordan, D.S., "The Instructional Design Environment," to appear in *Intelligent Tutoring Systems: Lessons Learned*, J. Psozka, L.D. Massey, & S.A. Mutter (Eds), Lawrence Erlbaum Associates, Inc, Hillsdale, NJ, in press.
- [Toul58] Toulmin, S., *The Uses of Argument*, Cambridge University Press, Cambridge, 1958.
- [Trig87] Trigg, R.H., Moran, T.P., and Halasz, F.G., "Adaptability and Tailorability in NoteCards" *Proceedings of INTERACT-87*, Stuttgart, West Germany, September, 1987.
- [VanL85] VanLehn, K., "Theory reform caused by an argumentation tool," Xerox Palo Alto Research Center Technical Report, ISL-11, July, 1985.



Systems I I

Thoth-II: Hypertext with Explicit Semantics

George H. Collier

Bell Communications Research, MRE2Q373
435 South St.
Morristown, NJ 07960

ABSTRACT

This paper describes a hypertext system - Thoth-II. This system provides a rich means for modeling semantic interconnections among texts. It allows a user to browse texts, exploring their relations with other texts. These relations are modeled by a directed graph. The texts are embedded in the graph. Connections among specified phrases in the text and the graph structure are automatically formed. In the browsing mode the user is presented with an interactive graphic display of the directed graph. In the text mode the user can use multiple windows to display and interact with the stored text.

INTRODUCTION¹

Modern high speed digital computers have made possible the existence of many new tools for working with text. Some of these are: WYSIWYG text editors (such as the one I composed this paper on); idea processors such as MORE [MORE86]; desktop publishing tools such as ReadySetGo! [Read86]; key word retrieval systems such as S-Book [Gome87]; and recent *hypertext systems*, such as the one that I will describe.

Hypertext is not new; it was first proposed by Vannevar Bush in the 1940s [Bush45]. Computerized versions of Bush's Memex were proposed by Ted Nelson and Douglas Engelbart in the early 1960s. Ted Nelson coined the term "hypertext" to refer to the electronic documents supported by such a system. Such systems are in vogue; a recent survey [Conk86] counted over fifteen in various states, two being commercial products for Macintosh PCs.²

This paper will describe my version (and vision) of hypertext. It takes a unique approach in that it attempts to support the direct modeling of the semantics underlying the hypertext structures created by the document author.

Printed Texts: Ink on Paper

Printed text is an inherently linear structure. A sentence is made up of words in a certain order. In turn, a paragraph is made up of sentences in a certain order. Chapters follow chapters in a set sequence. Yet text is highly interconnected and multiply linked by the *meanings* of the terms it uses, its *logical structure* and the *rhetorical relations* among its components. These relations are intrinsically non-linear.

For example, consider the arrangement of a reference work such as an encyclopedia. The articles in the encyclopedia are ordered alphabetically by entry names. Yet the articles are highly interrelated and multiply interconnected by semantic, logical, and rhetorical relations. For instance, an article on modern art might refer to World War I and its influence on the modern conception of the world. These articles might describe the same object, for instance, France, in which case a form of *semantic co-reference* ties these two articles together. An author might refer to another article as holding a position supporting his. Here one part of a text is *logically* linked to another as it provides logical support for a premise in the other. As an example of a *rhetorical* relation, an article on a painter might be an expansion of some topic only touched on in passing in another article.

Sometimes these relations are marked explicitly in the printed text, but often they are not. Existing structures for marking relations among texts include footnotes (from marker to note), reference sections (from text to text), glossaries (from phrases in a text to a defining phrase), and indices (from a topic to a place). Yet these markers represent much less of the structure than one would desire. Consider a "see also" section that points to other articles. But such a section does not provide information about why it would be useful to see another article or the context for the reference. In a printed text one can only provide a weak pointer to an interrelated piece of text. With modern computers it is possible to provide much more powerful tools.

- Printed text has inherently linear and often arbitrary ordering forced on it by the print medium.
- Semantically and logically texts are tied together in conceptual "webs".
- Existing means for marking connections among texts support only *part* of the web of interconnections.

HYPER Texts

Awareness of the forgoing limitations in ink-on-paper is not new. In 1945 Vannevar Bush, the director of the Office of Scientific Research and Development, proposed in an article in *Atlantic* that :

Selection by association, rather than by indexing, may yet be mechanized. ... Consider a future device for individual use, which is a sort of mechanized private file and library. ... A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility.

All hypertext systems begin with this proposal. Bush did not know of the developments in micro-electronics leading to today's computers; so his proposal is limited by the technology of 40 years ago. The memex was to be based on microfilms for memory and a mechanical mechanism for retrieving the appropriate entry. None the less, the article is prescient and subsequent proposals differ mainly in the technological base.

Ted Nelson in a unusual³ book called *Computer Lib* and in a later book [Nels81] proposed the notion of hypertext. In Nelson's proposal (as implemented in the Xanadu system [Greg83]) documents are connected together by links. The links can be point-to-point, point-to-span and span-to-span. A point-to-point link goes from from one "point" - a particular location in a document (such as immediately following the dagger here: †) - to some other point in the same or another document. "Spans" are regions of text (such as this paragraph. Links connect regions of text (spans) to places in text or span to span.

Another pioneer was Douglas Engelbart [Enge84] whose system, NLS (later called AUGMENT), focused on the group work environment , providing support for collaboration. This system includes facilities for electronic mail, multiple authorship of documents and even a remote on-line "conferencing mode". Again in NLS any text entity may be addressed and linked to other pieces of text.

All of these, Bush's Memex, NLS/AUGMENT, and Xanadu links, are designed to break the "linear straightjacket" of ink on paper. There have been many systems which have followed on the heels of these original proposals.

Other Proposals

The central notion in all hypertext systems is that of "linking" text together. There are many different hypertext projects; the major ones are: NLS/Augment [Enge84]; Xanadu [Greg83], [Nels81]; TextNet [Trig83], [Trig86]; TIES [Shne86]; Symbolics Document Examiner [Symb85]; NoteCards [Hala86]; Intermedia [Yank85]; Neptune [Deli86]; PlaneText [Gull86]; and Guide [Guid87]. All of these systems rest on the idea of tying

pieces of text together. However, the treatment of the links varies widely. I do not plan to review these systems in this paper; the interested reader should see [Conk86].

What does characterize all these systems is the underlying notion of a link. These links are used to connect things together, usually text, although several of these systems allow connections to graphics, etc. It is possible to think of the links as connecting abstract nodes (be they whole documents, places in documents, pictures, etc.) and conceive of hypertext as a *directed graph*. It is this conception that underlies the design of Thoth-II.⁴

THOTH-II'S DESIGN.

Thoth-II, like all hypertext systems, needs to provide several things:

- A way of storing and manipulating the pieces of text which jointly make up the hypertext.
- A way of storing and manipulating links among the pieces of text.
- An interface which makes the system usable.

In the sections that follow I will discuss the way that these are provided in Thoth-II. The underlying design of Thoth-II is simple. Thoth-II *is based on thinking of hypertext as a directed graph*. Everything in the system is treated as a node, a link, part of a node or link or the result of traversing a link. Since Thoth-II treats hypertexts as objects embedded in a directed graph it is very similar to TextNet [Trig83], [Trig86] and Netbook [Shas85a].

Links and Nodes Make a DG

In graph theory a network is defined by a set of vertices and edges or nodes and links. All hypertext systems share the notion of text that has been "stitched" together by a set of links. In Thoth-II the links bind together *nodes*. Together the links, which represent *connections* between the nodes, and the nodes define a network.⁵ The hypertext reader interacts with the system by following interconnections between pieces of text and visiting them.

In Thoth-II, unlike other hypertext systems such as the Brown Intermedia project, nodes have an existence separate from the text. In the Intermedia system links connect regions of text to other regions of text. In Thoth-II the links connect nodes and *the text is in turn connected to these nodes by other types of links*. As strange as it might seem, it would be possible to create a network in Thoth-II that has no text at all embedded in it. This is because the network is intended to do more than "just" connect pieces of text; it is intended to represent some part of the designer's conception of the topic about which he is creating the document.

This abstraction of the nodes is one of the differences between Thoth-II and TextNet. While both Thoth-II and TextNet share the idea of embedding text in a semantic net, in TextNet the notion of a node or *chunk* is much more tightly coupled to a piece of text. One and only one piece of text is stored at a chunk. The same is true in NoteCards [Hala86], where each link points to a "notecard" on which text is stored. In Thoth-II it was my intention that the nodes be stand-ins for concepts (or real world objects) and thus they may have none, one, or many pieces of text tied to them. Nodes represent a conceptual anchor around which collect pieces of text and connections to other objects. These anchors can (to abuse a metaphor) put hooks into the text through a process which I call "partial semantic interpretation". This is described in more detail below, but essentially is a process of connecting the network structure to the text on the basis of target phrases. In this, Thoth-II creeps closer to natural language processing (although it would have a long way to go).

In a directed graph it is possible to *label* the links and the nodes. By the perspicuous use of labels it is possible to create directed graph structures that represent relations that hold among objects. We might say that such a graph has been given *meaning* or *semantics* and that it is a *semantic net*.

Hypertext with Semantics

Thoth-II is designed around giving the hypertext object, the text pieces and their interconnections, an explicit semantics. The hypertext object consists of a set of linked pieces of text (or other objects). In Thoth-II the structure created by this network is given an explicit meaning.

What is Semantics?

The term "semantics" has had a long history in philosophy, psychology, linguistics and artificial intelligence. It has been used to "mean" many things. In AI, for instance, there are several major theories of how representational schemes such as semantic nets gain meaning, i.e., how they are useful for representing knowledge. At least three positions may be distinguished: procedural semantics, extensional semantics, and intensional semantics [Coll87a].

A semantic net, which Thoth-II relies on, is a *declarative* style of AI knowledge representation. Declarative styles do not embody procedures which can be understood as their semantics.⁶ The sort of networks used in Thoth-II are static and they gain their semantics from a relation to some other thing: *the world* or some *conceptualization* of it.

Semantics in a Semantic Network

A semantic net is a labeled directed graph. In particular, a semantic net is a labeled directed graph in which the labels are chosen in an interesting way. For instance, suppose

that one wanted to represent the fact that Santa Claus lives at the North Pole. One might represent the North Pole as one node, Santa Claus as another node, and the relation "lives at" as a link between the two nodes. The fact that Rudolph shares Santa Claus's place of residence might be represented by adding a node for Rudolph and connecting it to the North Pole node with another instance of the "lives at" link.

In a semantic net nodes represent objects and links represent relations among these objects.⁷ One can think of the objects as concepts or meanings and the relations among them conceptual relations (an intensional account) or think of the objects as entities in the world and relations among them (an extensional account). Notice that while the relations allowed are only binary this does not restrict the *expressiveness* of the semantic net. Any n-ary relation can be decomposed into a binary relation. The commitment to binary relations does have some effect on the *fluency* of the representation as this decomposition may do some damage to original conception underlying the construction of the network. Another restriction on semantic network representations as opposed to the more powerful frame based representations is a difficulty in representing *kinds* of things and *natural classes* [Coll87b]. It is hard to represent what we mean by saying "Rudolph is a reindeer" in a pure semantic net since a notion of inheritance of properties in a type hierarchy is lacking. In most semantic net notations this is added but Thoth-II does not support it.

The advantage semantic networks have over other types of semantics for hypertext is at least two-fold. First the *node-link-node* structure supports the hypertext notion in a very straightforward way. The nodes represent places where the texts can "live" while the links are connections (relations) among the texts/nodes. The *node-link-node* structure also supports browsing in a very natural way. In Thoth-II browsing is traversing the network where the links represent paths among the nodes. Finally, semantic nets are "broad-brush"⁸ relative to other styles of knowledge representation. Predicate logic and frames support more sophisticated styles of representation but this sophistication forces constraints. The difference in style is like that of drawing with finger paints vs. with a drafting pencil.

Visiting Nodes and Traversing Links

In Thoth-II as in all hypertext systems it is possible to *browse* a document by following the interconnections among the pieces of text. In Thoth-II this is conceived of as visiting a node and traversing links. Visiting a node gives access to the information connected to that node; browsing (or traversing) a link either causes an action to take place (such as retrieval of a piece of text) or gives access to another node.

Node and Link Types

In Thoth-II there are three kinds of links and one kind of node. The basic semantic network is defined by the nodes and *value* links. Value links point from a node to another

node. *Text* links connect pieces of text to nodes. *Lexical* links connect regions of text to nodes. Thus a *Thoth-II* document is a construction of these four kinds of things. Figure 1 is a representation of the sort of structures making up *Thoth-II* documents.

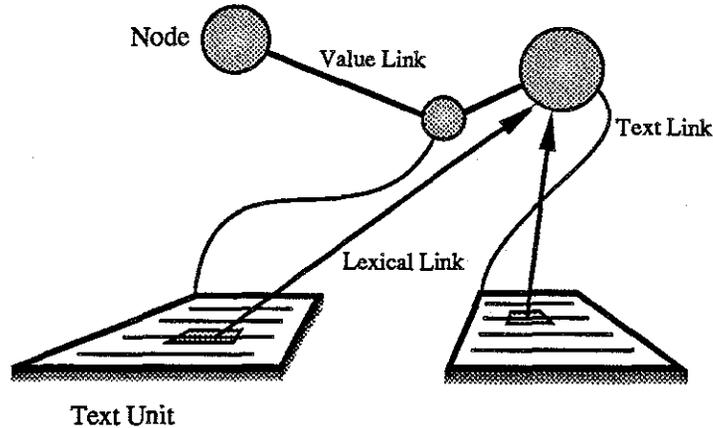


Figure 1. Diagram of the types of structures which make up *Thoth-II* documents. The spheres represent nodes in the network. Nodes in the network are tied together by *value links*. These nodes are tied to pieces of text (the polygons) by two kinds of links. *Text links* are direct connections between a node and a piece of text. *Lexical links* connected phrases in the text to nodes.

In another sense of "kind" there are (in principle) an infinite number of types of nodes and links. As mentioned above the nodes and links can be labeled. Each node in the network is treated as unique and its label is treated as its proper name. Each link label is treated as a kind (of relation) and information about the use of particular instances of each link type are collected in a central place.

Link Traversal Equals Action

A central metaphor in the design of the system is: link traversal causes actions. In the case of value links, which link nodes to other nodes, link traversal browses the network, moving the user from one "location" to another. On the other hand, traversing a text link causes the system to retrieve the text piece attached to a node by that link. Traversing a lexical link, which connects text regions to nodes, causes the system to move to that node. The design of the interface which implements this metaphor is described in more detail below.

Semantics In Thoth-II

One of the major design goals in *Thoth-II* was to embed semantics into the hypertext document. This is created by the labeled directed graph. Nodes in the network can be thought of as representing objects while the links between the nodes represent relations

among the objects. It is possible to represent in Thoth-II that a car has several major subsystems: its engine, drive train, suspension, etc. To each of these conceptual objects texts may be connected. These texts might provide descriptions of the drive train, for instance, and instructions for its maintenance. The links provide relations for modeling the connections among the conceptual objects. The engine might have many *causes of failure* and these could be represented in the system and connected to the node for the engine by a *cause_of_failure* relation.

By representing these concepts and representing the conceptual relations that hold among them the document designer is expressing a conceptualization of a domain *and* giving a reason for its existence. Hopefully this labeling of the structure can serve as a guide to a user as the reason for the connections among pieces of the structure are made at least slightly more specific. Presumably in any hypertext structure there is a reason for building a connection among one document (or place in a document) and another. Here the designer is able to make more explicit why that connection is there.

Partial Semantic Interpretation

There is semantics in the network as the network represents concepts and their interrelations. There is also semantics "in the text" as the system provides a mechanism for interpreting the text and tying it into the network. Just as a name, such as "Bob", or a definite description, "the present president of the United States" refers to some individual (perhaps the same); phrases in the stored text can be connected to nodes in the network. The system has a stored "lexicon" which it uses to make *exact string matches* against the text. If one tells it to search for the literal string "hypertext" and connect it to a node in the network then every time it displays text with that string embedded in it it will highlight the string (by italicizing it) and the user can mouse on that text and gain access to information stored there. This is similar to the Ties system [Shne86] but in that system these pieces of text point to other pieces of text and the connections among them.

Another difference between this process and the Ties system is that in Ties one must explicitly mark-off these text regions during document preparation. Thoth-II has a stored lexicon of phrases which it automatically and dynamically matches against the text. Each phrase is associated with a node in the network. The system can not deal with multiple nodes tied to a single phrase - the equivalent of polysemy in natural language; but it does allow multiple phrases to point to the same node - the equivalent of synonymy. Since the program does strict string matching and does not do any syntactic analysis (such as stemming) the matching is quite restricted. On the other hand, in technical texts there are many "frozen" technical noun phrases whose meaning does not vary with the context particularly. Here the technique has proved quite useful.

Semantics In Other Hypertext Systems

Other designers of hypertext systems have considered the issue of semantics. In fact, many begin with an explicit rejection of the notion underlying the design of Thoth-II. Hypertext systems are often seen as presenting an alternative to the type of "full-blown" knowledge representation used in natural language processing (frames; etc.). In Trigg's system the semantic network is used to model the logical and rhetorical relations among pieces of text and not the domain which the text is about. NoteCards [Hala86] began with frustration with trying to code a document into a full-blown knowledge representation scheme.⁹ In Brown's Intermedia project [Yank85] links connect "blocks" (what Nelson would call "span-to-span" links) and are not presently labeled.

In such approaches explicit representation of the target domain is either rejected or impossible. It is my belief that this is mistaken. The choice is not between representing the domain in "full-blown" formalisms or hypertexts. Rather both are points on a continuum. The structure of the links is at least implicitly a model of the domain. It is due to the *semantic content* of the text that one feels that one piece of text is related to another. I would claim that semantics is latently present (in a more or less organized fashion) in any hypertext structure. The advantage of the present approach is that it makes the semantics explicit.¹⁰

The motivation for ignoring semantics comes in part from a fallacious belief that a choice must be made between *real* knowledge representation and something else. *All* knowledge representation schemes, including the one in my head, are inadequate. *All* knowledge representation structures are incomplete. Some of the real dimensions of choice are (1) *amount*: how much information is represented; (2) *expressiveness*: what it is possible to represent; and (3) *fluency*: how easy it is to represent what one wants to represent. Along these dimensions variation is possible and, for instance, one can trade off the amount of information represented against ones investment in creation time. Underlying the design of Thoth-II is a commitment to such partial representation schemes.

Design Summary

The goal in the design of Thoth-II is to exploit the directed graph which is an intrinsic part of any hypertext structure; so that it explicitly represents information. The information represented in the directed graph relates to the texts embedded in it and provides a guide to their interrelations and dependencies. To this end texts are tied to particular nodes in the graph - conceptual anchors - and phrases in the text are recognized and pointers built to nodes in the graph.

- In Thoth-II pieces of text are embedded in a labeled directed graph - a semantic net.

- The semantic net is used to represent the domain with which the texts deal.
- Partial semantic interpretation of the text is done by string matching phrases and connecting them to nodes in the graph.

THE INTERFACE

The interface to Thoth-II is based on the Symbolics™ Lisp Machine user interface [Symb85] and makes extensive use of its mouse and menu facility. The interface has two modes - a browsing mode and a text display mode. In the browsing mode the user interacts with a graphic display of the directed graph, while in the text mode the user views units of text in a multi-window environment. The graph provides a model of the textual interconnections and the user is able to "visit" pieces of text that are connected to particular nodes.

Spiders: A DG Browser

Users browse the DG by interacting with a graphic display of the nodes and their connections. Figure 2 is a screen dump of a Thoth Spiders¹¹ display of a directed graph that represents part of a repair manual for a VW Rabbit.¹² This manual will be described in much greater detail below. The thin lines represent links; the dots represent nodes. Each link and node is labeled.

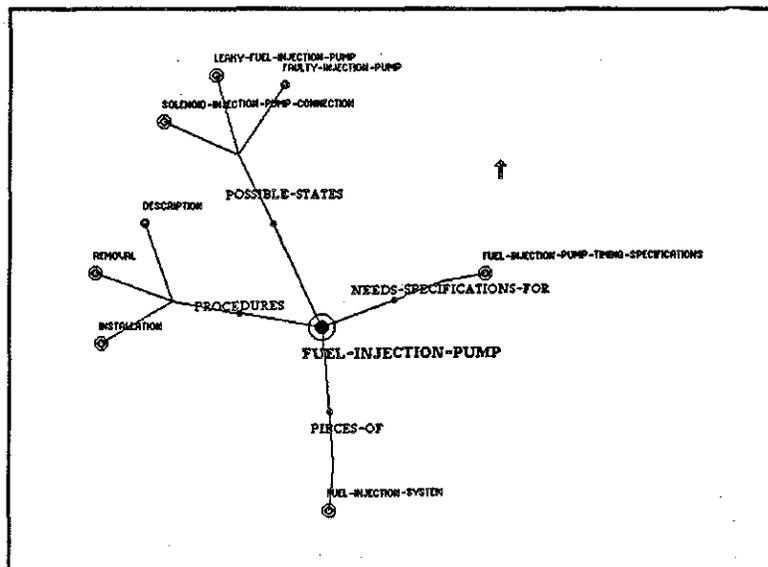


Figure 2. This figure is a reproduction of a screen dump of the Thoth Spiders display of a semantic net. The large black dots represent the nodes and the lines represent links between the nodes. All objects in the display are "mouse-able" and can be manipulated by the user.

This display is based on a different philosophy from other browsers for graph structures, such as SemNet.¹³ In some approaches the user is shown the whole structure at once. In

SemNet nodes occupy locations in a three dimensional space and only a single display object exists for any given node. The problem such systems face is that it is difficult to display all the myriad links and nodes without the display objects falling on top of one another and obscuring each other. This problem is partially solved in SemNet by locating the structure in three dimensional space and showing the user a two dimensional view of it. The user can "fly around" in the space varying the angle from which he views the structure. The interface model is: *the user is presented with different views of one static object that represents the complete structure.*

The design of Thoth Spiders is different. The space is a two dimensional *plane* rather than a three dimensional space. Location in the space is uninterpreted. Often in browsers such as SemNet, there is an attempt to map spatial location into some conceptual metric. For instance, the *y* axis might be used to represent abstractness. The most abstract objects would be plotted high up in the space while the more concrete sink to the bottom. In Spiders where a node is located on the display is purely a function of the users interaction with the system. Secondly, users interact with the structure that they view, creating the graphic objects as they browse. These properties flow from two basic decisions:

- The location of a graphic object has no particular meaning in terms of the abstract structure that is being displayed.
- There can be multiple graphic objects all representing the same abstract object.

This different use of the display space allows the drawing of the network to "flow out" over the 2D plane. In any system in which the nodes are multiply connected the placement of one node depends on the place of the nodes to which it is connected. By breaking these connections display is made inordinately easier. The display algorithm breaks *cycles* by treating every reference to a node as unique. This potentially creates multiple *graphic* representations of the *same* underlying node in the network. The interface model is: *the user views a two dimensional plane on which part of the larger graph structure is drawn.*

Users interactively create this structure by browsing the links connecting the nodes. The displayed nodes are in one of two states. If they are expanded, all of their links are drawn. In Figure 2 an expanded node (the heavy black dot) is at the center of the screen. A fan of links comes out of the node and, from the links, fans of attached nodes. The attached nodes are in an unexpanded state. Browsing the structure is performed by "expanding" a node. This results in the promotion of the node to the status of expanded, and all of its links are drawn. In Figure 3 the node for **Fuel Injection Pump Timing Specifications** hanging off of the link named **Needs-Specifications-For** has been expanded and we see more of the structure. Since the display is treated as a window onto a much larger plane, it is possible to slide the window around and view parts of the structure that have fallen outside the viewing area. The user moves his view around by clicking on menu entries. The bottom menu entries, **Up**, **Down**, **Right**, and **Left** slide the window around; the item **Center on Mouse**, centers the view on a point selected by the user with a mouse. Not only can a user

change the point of view, by moving the "window" about the graphic plane, he or she can manipulate the way the graph is drawn. The user can "pickup" and move any of the nodes and links. One of the major design goals is to give users as many tools for manipulating what they see as possible. Users should be allowed the view of the data they want. The display is the result of a *negotiation* between the user and the program.

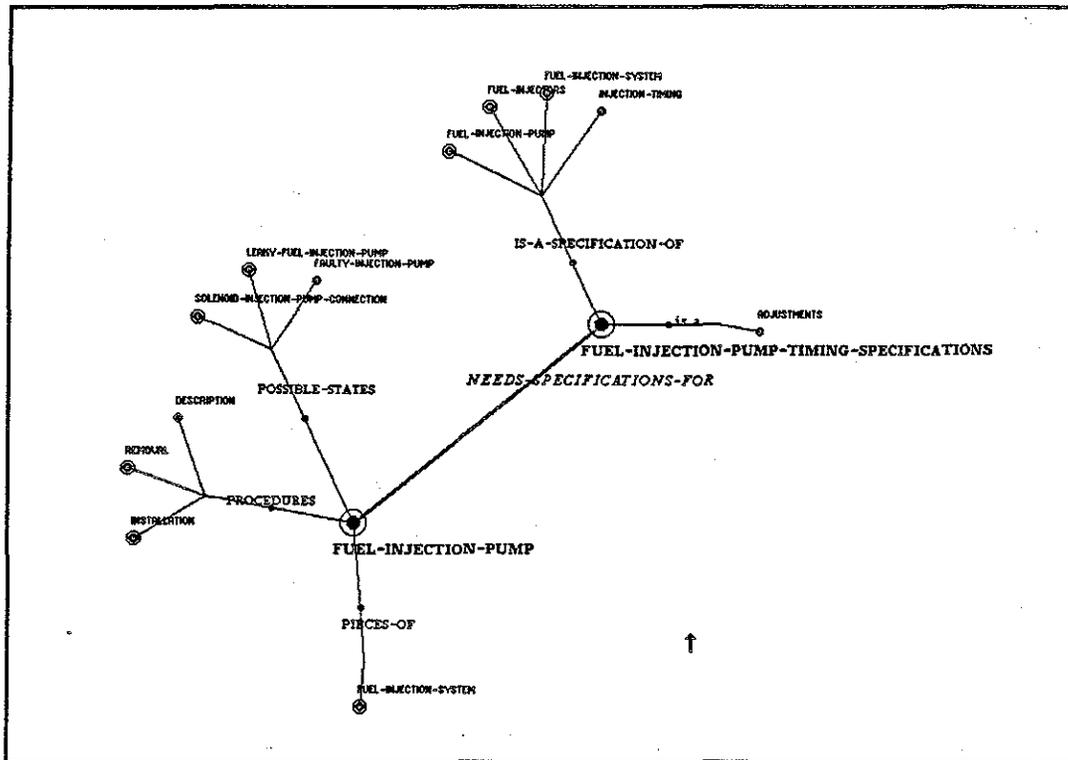


Figure 3. This figure is a screen dump of a Spiders display after a user has expanded a node and its links are now displayed. Spiders is designed to allow users to quickly browse a large semantic network by expanding nodes and following the links joining them.

Once users have decided that they would like to peruse the text linked to a particular node, they invoke text mode.

Text Mode

Figure 4 is dump of the screen, when the system is in text-viewing mode. The interface model is: *a window is a view of a node*. Here Thoth-II differs from other systems. In most systems - NoteCards for instance - nodes are pieces of text and vice versa. In Thoth-II there is another level of abstraction, the nodes to which text is connected. In a text-mode window the user can read any of the associated text at that node. These windows were created either by "querying" a node in the DG while in browsing mode or by "querying" a phrase in a piece of text while in text mode (see below). All text windows have the same structure. On the left side of the window is a menu bar. Displayed on the top is the name of the node to which the window points. In the menu are two types of commands. One set is window

operations such as resizing and movement. The other set consists of the **text** command and the **graph** command. The **graph** command puts the display back in the browsing mode. The **text** command pops up a menu of the text associated (linked) with that node.

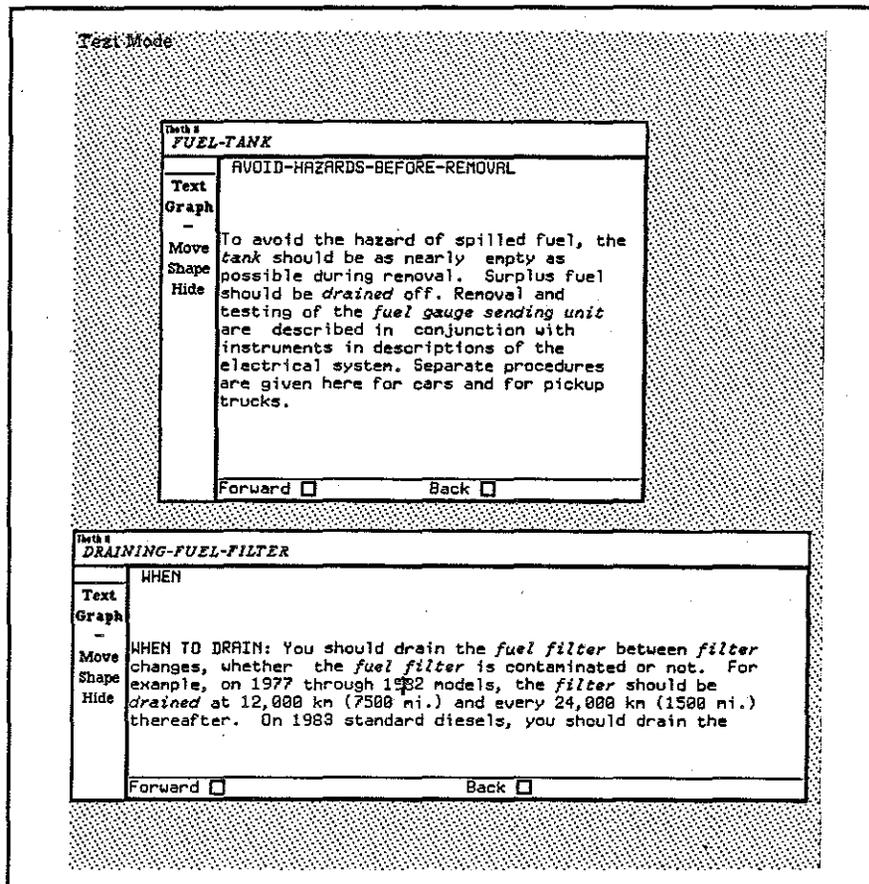


Figure 4. This is part of a screen dump of text mode. Here two windows are displayed. Each window is a 'window' on a node and the text attached to that node can be viewed in that window.

If any of the text displayed exactly matches one of the stored phrases, then that piece of text is italicized. In Figure 4 several words are italicized and the one with the mouse on has a box drawn around it. If the user clicks on a word when it is boxed, the system follows the pointer from the phrase to the associated node and makes the associated window. Thus if the user had clicked on the right mouse key while the system was in the state displayed in Figure 4, then a window would have been created that displayed text associated with the node corresponding to the phrase "fuel filter". This sort of interaction is similar to that found in the TIES system [Shne86]. However Thoth-II supports multiple windows and on-the-fly recognition of target phrases. In Thoth-II, as the text is being displayed, it is checked against the words in the stored phrasal lexicon.

IMPLEMENTATION

The current version of Thoth-II is implemented in Zeta-Lisp and runs on the Symbolics 3600 family of computers. It makes extensive use of flavors which is the object oriented programming facility on the Symbolics.

The use of object oriented programming allowed me to distribute the flow of control across the objects. For instance, each "spider" - which displays as a large central black dot with a set of lines radiating out of it - has a flavor object associated with it that "knows" how to draw itself, what other spiders it is connected to, what its current location on the screen is, whether it is off-screen, what node in the semantic net it is associated with, etc. The interface is based on message passing among objects and generic methods that are inherited and specialized by the particular instances.

Using Lisp and a Lisp Machine also allowed me to take advantage of the large virtual memory and built in memory management. The current network is held in virtual memory. It is not necessary to cache the links in a data-base as is done in the Intermedia project. Further, Lisp naturally supports the kind of connectivity exploited in Thoth-II and intrinsic to the notion of hypertext.

USING THOTH-II

Thoth-II is designed to help users browse a hypertext document. The definition of this document consists of a file in a certain format. The following sections define that format and describe a sample document.

The Input File

The syntax of input files to Thoth-II are simple. For value links the syntax is:

```
(node-name1 link node-name2)
```

or

```
(node-name1 link node-name2 node-name3 ... node-namen).
```

Here the links references have been underlined for clarity.

The syntax for text links is:

```
(node-name text-link "text for display")
```

or

```
(node-name text-link "file name").
```

Lexical links are similar:

```
(node-name lex "string to look for").
```

Two kinds of links do not directly create structure but are useful for organizing the hypertext. *Backlinks* automatically create links that go in an opposite direction to the links that they are a backlink for. Pretty-name links supply names for screen display.

```
(link1 backlink link2)
```

```
(node-name pretty-name "name to print")
```

```
(link pretty-name "name to print")
```

Let us consider a very small example. Below is a short list of input forms and Figure 5 is a graphical representation of the resulting hypertext structure.

```
(vw-rabbit haspart rabbit-engine)
```

```
(rabbit-engine problems engine-wont-start)
```

```
(engine-wont-start cause dead-battery)
```

```
(vw-rabbit description "The vw rabbit is a German car.")
```

```
(battery-dead description "The battery is dead if it has no electrical charge. This makes the car impossible to start.")
```

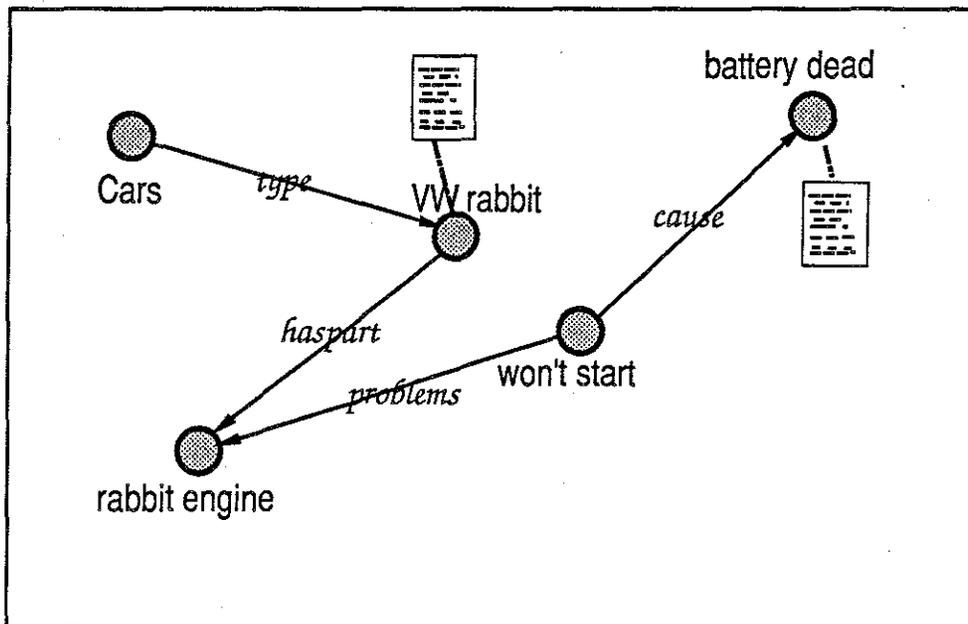


Figure 5. This is a diagram of the network that would be created by the sample input file described in the text. The circles represent nodes, while the arrows represent value links between the nodes. Two pieces of text have been attached to the nodes.

VW Hypermanual

This section describes the VW Hypermanual, a hypertext repair manual built in Thoth-II. This manual was designed and built by Cheryl L. Wilson of Bellcore.

Goal of the Hypermanual

The VW Hypermanual is designed to replace a paper repair manual for a VW diesel Rabbit. The paper manual is designed for the amateur and professional mechanic. The section modeled in Thoth-II is taken from the fuel injection portion of the fuel and exhaust system section. This corresponds to approximately 17 pages in the paper manual and took about 40 hours to construct. The network consists of 52 nodes and 96 relations.

Organization of the Hypermanual

There are several simple principles underlying the organization of the manual.

Conceptual 'planes of connection' are one of the most important features of the design of the VW Hypermanual. These planes partition the complete network and are a "view" of the complete VW system. For instance, one conceptual plane is *major systems*; these nodes and relations constitute a coherent whole which stands in its own right. Objects in this plane or view are much more highly interconnected with each other than with objects in other "planes". A second plane contains a representation of mechanical problems found in the rabbit fuel system. This is yet another view of the rabbit's fuel system and it is one that a user might use to guide his search for cures for his balky car.

Another organizing principle is the use of nodes in the network to represent categories of things. For instance, a node is used to stand for the class of problems encountered. Particular instances of problems are tied to the node representing problems in general by a relation of "instantiation". Some of the major categories are:

- Parts, e.g., fuel injection pump
- States of parts, e.g., leaky.
- Symptoms of the car, e.g., hard starting.
- Systems which make up the car, e.g., exhaust system.

Certain relations also play central roles. The car is made of parts, which are in turn made of subparts. So the relation *part of* plays a central role in organizing the network. Other important relations are *symptom* and *cause*.

Lexical entries, i.e., the recognition of word strings on the fly, provide an important mechanism for tying together the network. For instance, the description of the cause of some electrical problem might refer to the "battery". This word is tied automatically to the node which represents the car's battery. The user can then get more information about that part simply by clicking on the italicized text. The use of these phrases allows users to change their direction of attack on the fly, to get more detail about a topic when and if needed, to refresh memory of definitions, etc.

Paper versus the Electronic Manual

While constructing this hypertext document several major differences between it and the paper version became clear.

- A hierarchical organization has been avoided.
- The manual allows user to easily change topics by changing their "direction of travel" through the network.
- User can easily control the level of detail.
- The semantic network provides an organizing guide for the user that is not as restrictive as a strict hierarchy or an index.
- The user is not the "victim" of a single vision of the index as expressed by the index or table of contents.

Hypermanual Conclusion

While the manual described is quite small (it currently is being vigorously worked on), it is large enough to show, we believe, some of the potential of semantically based hypertext systems like Thoth-II for technical manuals. It seems to us that such manuals will be quite different from ordinary paper manuals and they may require completely new writing styles. On the other hand, their creation will be justified by the increases in flexibility and tailorability to the specific needs of users.

CONCLUSION

Thoth-II is a browser for hypertexts. It allows for powerful and flexible linking of texts together. The structure that links the documents together can also serve as a model of some aspect of the structure of the text, whether it be at the semantic, logical, or rhetorical levels. The interface uses the power of a high resolution graphic display to show complicated conceptual information in an immediate and vivid form. The system exploits multi-windows, allowing the user to view simultaneously many different pieces of text.

Advantages of the Approach

I believe that this approach differs from some others in significant ways. Further these differences give it some advantages.

- The latent structure embedded in any hypertext is made explicit.
- The semantics of the network provide a guide for use in browsing.
- The network provides an explicit conceptual model of the domain.

Disadvantages of the Approach

There are also substantial disadvantages to the approach.

- There is no hierarchical structure imposed. Tables of contents have their uses!
- It is easy to get lost.
- It is hard to do the representation work correctly.

Final Summary

Paper is hard to beat. It is cheap. You can put it in your pocket. It has very high resolution.¹⁴ The technology for using it is highly developed. Yet it is hard to model the structure of documents and their interconnections using ink on paper. Interesting computer systems have to do more than turn low resolution pages. Hypertext is one such possibility. In Thoth-II one can, it is hoped, capture something of the complex conceptual connections that tie pieces of text together. As a result, we can make an expert's view of a body of literature more accessible to others. Understanding often consists of knowing how things are connected together.

REFERENCES

- [Bush45] Bush, V. (1945). As we may think, *Atlantic Monthly*, 176, 101-108.
- [Coll87a] Collier, G. (1987). Describing States of Affairs in KEE 3.0. Bell Communications Research TM-ARH-009623.
- [Coll87b] Collier, G. (1987) On the Reification of Universals in Frame-Based Knowledge Representation. Bell Communications Research TM (in press).
- [Conk86] Conklin, J. (1986). A Survey of Hypertext, MCC STP-356-86.
- [Deli86] Deliesle, N. (1986). Neptune: A Hypertext System for CAD Applications, CR-85-50, Tektronix Computer Research Laboratory.
- [Enge84] Engelbart, D. (1984). Collaboration Support Provisions in Augment, *Afips Office Automation Conference*.
- [File84] Filevision (1984) Telos Software, 3420 Ocean Park Blvd., Santa Monica, CA 90405.
- [Fox79a] Fox, M. S. & Palay, A. J. (1979a). The BROWSE System: An Introduction, Carnegie-Mellon University.
- [Fox79b] Fox, M. S. & Palay, A. J. (1979b). The BROWSE System: Part II: Phase II and Future Research, Carnegie-Mellon University.
- [Gome87] Gomez, L., Landauer, T. & Remde, J. (1987). Superbook: An Automatic Tool for Information Exploration. Bell Communications Research TM (in press).
- [Greg83] Gregory, R. (1983). Xanadu: Hypertext from the future, *Dr. Dobb's Journal*, 75, 28-35.
- [Guid87] Guide (1987). Owl International Inc., 14218 NE 21st St. Bellvue, WA. 98007.
- [Gull86] Gullichsen, E., D'Souza, D. & Lincoln, P. (1986). The PlaneTextBook, STP-333-86(P), MCC.
- [Hala86] Halasz, F. (1986). NoteCards: An Experimental Environment for Authoring and Idea Processing, Xerox PARC.
- [Jaco84] Jacobs, M.A. & Murray, D.J. (1984). *Filevision*, Telos Software Products, Santa Monica, Ca.

- [Lena83] Lenat, D; Borning, A; McDonald, D.; Taylor, C.; & Weyer, S. (1983). KNOESPHERE: Building Expert Systems with Encyclopedic Knowledge, *IJCAI, 8th International Joint Conference on Artificial Intelligence*, 167-169
- [MORE86] MORE (1986). Living Videotext, 2432 Charlestown Rd., Mountain View CA 94043.
- [Nels81] Nelson, T. (1981). Literary Machines, available from author, Box 128, Swathmore, PA, 19081.
- [Read86] ReadySetGo! 3 (1986). Manhattan Graphics, 401 Columbus Avenue, Valhalla, NY 10013.
- [Robe81] Robertson, G; McCracken, D; & Newell, A. (1981). The ZOG approach to Man-Machine Communication, *Int. J. Man-Machine Studies*, 14, 461-488.
- [Scha85] Schatz, B (1985). Telesophy, Telesophy Project at Bell Communications Research.
- [Shas85a] Shasha, D. (1985a). Netbook - an exploratory environment for readers and writers, New York University.
- [Shas85b] Shasha, D. (1985b). When does Non-Linear Text Help?, New York University, TR-178.
- [Shne86] Shneiderman, B. & Morariu, J. (1986). The Interactive Encyclopedia System (TIES), University of Maryland.
- [Symb85] Symbolics, Inc. (1985). Programming the User Interface, Symbolics, Inc.
- [Trig83] Trigg, R. (1983). A Network-Based Approach to Text Handling for the Online Scientific Community, University of Maryland,.
- [Trig86] Trigg, R. & Weiser, M. (1986). TEXTNET: A Network-Based Approach to Text Handling, *ACM Transactions on Office Information Systems*, 1, 4.
- [Walk85] Walker, D. (1985). Knowledge Resource Tools for Accessing Large Text Files, Bell Communications Research, 85-21233-25.
- [Weye82] Weyer, S. (1982). Dynamic Book for Information Search, *International Journal of Man-Machine Studies*, 17, 87-107.
- [Weye85] Weyer, S. & Borning, A. (1985). A Prototype Electronic Encyclopedia, *ACM Transactions on Office Information Systems*, 1, 3, 63-88.
- [Yank85] Yankelovich, N. & Meyrowitz, N. (1985). Reading and Writing the Electronic Book, *Computer*, 10, 17, 15-29.

¹This research has benefited from the ideas and encouragement of Robert Amsler and Don Walker. See Walker (1985) for a broader discussion of some the issues raised here. I would also like to acknowledge the assistance of Margaret Nilson, an early user, and Cheryl L. Wilson who is mostly responsible for the VW Hypermanual.

²Guide (1987) and Filevision (1984).

³Unusual only now as it was truly a product of the 1960s. It resembled the Whole Earth Catalog.

⁴Thoth was the Egyptian god of writing.

⁵The objects that are connected are not necessarily texts. They can be pictures, video sequences, etc.

⁶This is, of course, an oversimplification.

⁷There are more complicated semantics but they are irrelevant to the current discussion.

⁸It is worth noting that major theories of human memory and of meaning have been expressed via semantic nets.

⁹Personal communication.

¹⁰Hence the title of this paper.

¹¹The display resembles a childhood drawing of a spider.

¹²This hypertext was created by Cheryl Wilson of Bellcore.

¹³Personal communication from Kim Fairchild at MCC.

¹⁴At 300 dots per inch - standard laser printing - a Symbolics high resolution display would be about three inches on a side.

The Architecture of Static Hypertexts

Tim Oren

Apple Computer, Inc.
10500 N. DeAnza Blvd. MS 27AJ
Cupertino, CA 95014

ABSTRACT

This paper's purpose is to describe how the hypertext technique can make CD-ROM (and other static storage media) a more comfortable environment for human use. I begin by considering implementation issues for hypertext on CD-ROM and surveying currently available products. I suggest desirable goals for the use of hypertext on the static CD medium, and propose that their achievement will follow from a correct choice of conventions of use and construction of the hypertext database. Such goals include augmenting text search algorithms, recovering lost benefits of the print medium, designing meaningful connections between documents to assist human communications, and allowing variable interactivity with the user.

WHAT IS HYPERTEXT?

Hypertext is defined as non-sequential reading and writing. Documents stored in a hypertext system have the property of active cross-references. Active means that you can instantly gain access to the cited document by picking the cross-reference from the computer display. You may then follow further references, or return to the original document. In hypertext parlance, this active reference is called a link. The documents are often called nodes. An assembly of nodes and connecting links form a hypertext network. A set of links are often referred to collectively as a "web".

Though the hypertext term is new, the concept of linked documents is very old. For instance, the scientific and engineering literature with its standard system of cross-references forms a paper hypertext spanning hundreds of thousands of documents and hundreds of years. Many works of classic literature, such as the Talmud and Homer, have spawned bodies of interlocking commentary all related by a standard scheme of verse or passage referencing.

The difference in true hypertext systems is the technological support which replaces the manual process of locating and fetching volumes and passages. This notion can be traced to Vannevar Bush's seminal Memex paper of 1945, which proposed cameras, microfilm, and a mechanical retrieval engine as a means for supporting scientific research [Bush45]. In the 1960's, Douglas Englebart's NLS/Augment project substituted magnetic storage for film and a computer for the mechanical engine, and brought an early version of Memex to life [Enge68]. It primarily supported hierarchical linking, and was designed as a dynamic medium for cooperative work or individual thinking and writing. NLS is a superset of what has come to be called an "idea processor" or outliner [Hers85][Fost85].

The term hypertext was popularized by Ted Nelson in his 1974 book "Dream Machines" [Nels74] which looked beyond hierarchical relations to a densely interwoven network of nodes which directly reflected the ideas within the human mind. In later works Nelson expanded his notion into "Xanadu", a network of interconnected hypertext engines which would be both an environment for cooperative thinking and a medium for electronic publishing of hypertext works [Nels80][Nels82][Greg83].

In the last few year, several implementations of hypertext have appeared as academic or commercial products on workstation computers. These include Brown University's Intermedia [Yank85][Garrett 1986][Meyr86], the TIES program at University of Maryland [Shne86], Xerox' Notecards system [Mont86a][Mont86b][Trig86][Hala87], Tektronix' Neptune [Deli86], and the experimental PlaneText program at MCC [Gull86]. A excellent technical report available from MCC surveys these systems and discusses issues in designing hypertext software [Conk86].

IMPLEMENTING HYPERTEXT ON CD-ROM

Why should hypertext be of interest to the CD-ROM industry? We can readily observe that people think using ideas, concept and facts, not arbitrary pages or screens. Ideas are connected in the mind, forming classes and narratives. Hypertext shares the property of connectivity, relating documents or information chunks which in turn contain facts and concepts. It is reasonable to hope that a medium with this property would help overcome the searching and browsing problems that arise in large collections of isolated documents converted from the static print medium.

(We should hasten to note that hypertext is NOT a form of artificial intelligence. A.I. in general strives for a representation of human knowledge and physical reality which may be used effectively by the machine. In constrast, hypertext uses the machine to augment human thinking by providing a dynamic medium not possible on paper [Stef86]. The evolved human uses of hypertext may prove enlightening to researchers of cognition, but there is no necessary intent to model human thought.)

CD-ROM is particularly suited as a storage medium for hypertext. Two properties of CD-ROM are helpful: its read-only static nature and large capacity. Dynamic, editable hypertext have a problem with updating. Links may originate in any document and, depending on the implementation, may point to an entire document or a passage or point within it. When the target document is edited, moved or deleted, any or all of these links may have to be changed. While this task is feasible with suitable data structures, it may be a considerable computational load for a microcomputer when dealing with tens or hundreds of thousands of nodes. On CD-ROM, the basic document cannot be removed or altered. Annotations or new nodes may be added, but an unchanging node structure will always be present, relieving the update problem.

Before the appearance of CD-ROM, massive hypertext databases could only be brought to a desktop computer user by remote access to massive hard disks attached to mainframes or distributed networks. Distribution of the hypertext across machines or update by multiple users compounds the link updating problem with issues of simultaneity. A single CD-ROM is large enough to store a hypertext equivalent to 100 large print volumes. The advent of Xanadu and other hypertext writing systems will eventually make larger hypertexts possible, but the economics of creating hypertexts probably makes this a reasonable size limit for the first generation.

The size and static nature of CD-ROM also allows storage of the hypertext links for optimal

retrieval. Rather than storing the pointers in a disjoint database, they may be stored within the document itself. Duplicate tables of incoming and outgoing links for each document may also be placed on the disc, eliminating the need for multiple accesses to the CD-ROM when the user requests these summaries.

Already, vendors of CD-ROM retrieval software have begun to adopt the hypertext technique as an additional or primary browsing technique. The "hotlinks" feature of KnowledgeSet's KRS and "sideways browse" in the TMS Research software augment basically hierarchical browsing structures [KRS86][Rese86]. Owl's Guide product goes further in adopting hypertext as the basic metaphor for browsing [Guid86]. General microcomputer software products such as Box Company's Window Book and Living Videotext's More also have hypertext-like features and would readily adapt to CD-ROM use [Wind86][More86].

WHAT IS THE DESIGN PROBLEM?

Fulfilling the promise of hypertext will require careful design of the hypertext database. Hypertext itself is simply a technique; it does not imply any particular structure of links or nodes. Unplanned, unstructured links are not the best use of hypertext; they can produce an "amorphous blob" database that may be fascinating to explore, but does little to ensure fast access to interrelated ideas. In such an unplanned hypertext, the access problem for links can become as bad as that for documents. The user does not know what links to expect in a node, and navigation from place to place becomes a hit or miss proposition. It is easy to get lost, and there is no consistent pattern of knowledge to be found. Experiments at KnowledgeSet with a hypertext generated by automatic recognition of cross-references within Grolier's Electronic Encyclopedia tend to bear out these statements [Oren86].

What we need are, in Alan Kay's term, "conventions that work" in the hypertext medium. In seeking these conventions of use, we should continue the hypertext paradigm of honoring the mental habits of the human. We should look at existing media for techniques that work in the new situation and will be familiar scenery to the user. We should also realize that hypertext on CD-ROM is inherently different from the dynamic Xanadu environment envisioned by Nelson. Where dynamic hypertext is a tool for cooperative writing and online publishing, hypertext on CD-ROM is a stored medium constrained to transmit human communication in the absence of feedback from reader to author. In this situation we cannot hope for desirable properties of hypertext to emerge through use, we must design them.

A conservative approach to the design task is to attempt to alleviate problems which are now apparent with CD-ROM databases. One of these problems is the loss of desirable properties of the print medium: location, closure, unity of type, and state preservation, for instance. Another is the known difficulties of full text search, in particular the Boolean algorithms most common in CD-ROM retrieval engines. These difficulties include tradeoffs between retrieval precision and recall, and inconsistent human or automatic choice of indexing terms. A larger design task is to improve the degree of communication which is achievable with a static medium. If the structure of hypertext combined with the storage capacity of CD-ROM could achieve this goal we would have reason to claim success for the medium. This should be possible if there is merit to the hypertext claim of matching some aspects of human cognitive behavior.

RECOVERING BENEFITS OF PRINT

In exchanging print for the electronic medium we lose some benefits which occur because the paper volume has physical being. Restoring them would be a reasonable goal for hypertext. I will consider the properties of state preservation, unity of type, serendipity, closure, and

location.

State preservation

By state preservation I mean that a printed volume stays the same when noone is looking at it. Bookmarks or fingers can be put between pages to temporarily save locations. Marginalia and underlines remain in place. Temporary notes can be paper clipped to the pages or stuck on with adhesive. A hypertext browser should have similar properties [Scra85].

Bookmarks and fingers are used by humans to replace the cognitive task of remembering page numbers and the physical task of looking them up again. The electronic equivalent of a finger in the book is a browser which remembers the previous position when following a link. One way of doing this is to generate a new view window at each jump, leaving the previous document, scroll point, and selection in the old window. Multiple windows allow side-by-side comparison of documents and specification of new links by direct manipulation [Shne83]. However, they can easily produce clutter on a small microcomputer display. A compromise is to jump the browser window to the target document, but preserve the state of the previous document and restore upon return. An user option for requesting a new window restores the side-by-side benefits.

Hypertext bookmarks are simply links to a document which are generated at the user's request and stored with some associated comment or picture. As the collection of bookmarks builds up, they present an access problem in themselves. For this reason, and the unity of type concerns presented below, each set of bookmarks should be collected as a full fledged hypertext document which may be searched and targeted by a link.

To simulate writing in the book, some form of document versioning is required. Since neither marginalia nor highlighting delete information or change the fundamental document structure, they can be stored as an increment file associated with the document. Making a full copy of the documents with changes can quickly fill up a magnetic disk if a large CD-ROM database is annotated frequently.

The equivalent of Post-its or paperclipped notes can be constructed by putting a link in the increment file, targeted to a new notepad document stored on magnetic disc. Again, notes and marginalia should be fully searchable. If one notepad document or bookmark set is allowed to link to another, the general hypertext update problem is reintroduced. Here the designer must choose between benefit and implementation difficulty.

An interesting issue is whether annotations and notes should be private to the author or public to all users of the database. Privacy and security considerations speak for local notes, but a public option would allow the CD-ROM workstation to be a focus of collaboration and serendipity just as Xanadu envisions, though limited to sequential use.

Unity of type

Simply because everything is printed on the same paper, books have a unity of interface seldom achieved in software. The lesson for hypertext is to avoid needless multiplication of data types. For instance, the storage medium for a document should be transparent to the user. Links, bookmarks, and document references generated by search should be of one type. Hypertext documents, user notes, bookmark sets, search result lists and tours (see below) should also form one type. In a hypertext world where anything can link to anything, complexity grows as the square of public data types. Disallowing some type combinations introduces modes in the interface, to the confusion of the user.

Of course, some dynamic elements of hypertext have no print equivalent: animated graphics and video and sound sequences. These complexities should be buried within the scope of a document. The onus should be on the hypertext system to determine the correct type of browser for a document or object within a document. The Smalltalk approach of hiding the display method within the data object has much to recommend it here [Gold83].

Serendipity

The physical nature of a printed book encourages serendipity. The book can fall open at an unintended place. Interesting pictures may draw the eye into articles which would otherwise be missed. A passage being read may adjoin another which catches the eye. Given the perennial shortage of screen space in the electronic medium, much attention is given to removing just this sort of "irrelevant" information. Returning serendipity without intruding on the main retrieval task is a design task visited on the database architect [Love84].

Links can easily be inserted in a hypertext to simulate physical proximity created in page layout. There is no requirement that these links should obey conventional topology; they can lead anywhere in the database. They might be an opportunity for the designer to build in weak connectivity with a model such as "I want a link here that might interest the kind of person that would read this article, even if it's not directly relevant." For this feature to be used, the cost to explore a link must be very low, a single mouse click or keystroke with almost no pause on the jump and return.

Because pictures are recognized as gestalts they are an effective way to draw attention to documents otherwise missed. A number of printed works, such as the Whole Earth Catalog, have deliberately used this property to promote discovery by readers. A hypertext equivalent might be to link all pictures in a list and provide a slide show feature to start flipping through them at any point in the list. The user could stop the show at any point of interest and examine the related document. Since pictures often contain items not of direct relevance to the associated document, it may also be useful to design a picture description language to be attached to the images and searched separately from the text documents.

Truly random jumps have limited appeal. A document randomly chosen might already be familiar. If it is novel, it is entirely out of context and its relation to the remainder of the database will be a complete mystery unless the user is willing to invest the time to explore the neighborhood. To make random jumps useful, very strong location cues and tools must be available. This aspect may be better handled by tours as described below.

Closure

Closure is simply telling when you are done, or how far you have to go. In a conventional book, the end is quite obvious. The amount and weight of paper in the left hand versus the right is a continuing cue. It is easy to find how much remains in a chapter by paging quickly ahead, scanning for breaks in the text rather than reading. Hypertext loses these abilities; there is no direct way to tell what fraction of the database has been viewed and it is difficult to draw an analogy to "pages left in chapter" when the reader can instantly branch from a given linear or hierarchical reading pattern.

These properties can be simulated when the hypertext system retains a memory of the user's actions, which documents have been viewed and which links followed, and in what order. This memory can be used to generate simple statistics such as fraction of database read, but more importantly they can be used to lead the reader to the portion which is undiscovered. It becomes meaningful to make requests such as: "Show me something new", "Show me a new path to something I've seen before", "Let's review the paths I have taken to this node", "Put me back

in the context of last Tuesday", or "Tour this section using link types I most frequently choose."

Notice that any of these requests can be filled without a deep understanding of the contents of documents. The idea does suggest that document and link objects should be given the property of recording requests make of them, and that a global record of features used will be more than a tool for the interface designer. Again the issue of private and public access to these records is interesting. What value is there in seeing the database through other's eyes? Would this be a breach of intellectual privacy so blatant as to stifle exploration and speculation?

Location

A well designed book uses print conventions to give the reader a sense of location. Running chapter titles, section headings, type styles and indentation situate the current passage in the hierarchical or narrative structure of the work. Positioning of images and tables reinforce the topic under discussion. Again, many of these are lost in the electronic medium because they are deliberately designed as peripheral cues which are only occasionally brought into focus by the reader. When the physical element of proximity is removed, and restricted display area constrains layout, it is easy for these cues to be abandoned [Mont86a]. Hypertext systems exacerbate the problem by encouraging a proliferation of small nodes encapsulating a single idea [Conk86]. Display systems which present only one document at a time are particularly poor for location, because they throw the entire burden of placekeeping on human memory [Robe81][McCr84].

Location is one of the most difficult problems in hypertext design. Some print conventions, such as running titles and headings, can be adopted as is or converted into hierarchical or linear links among documents. However, such adaptations cannot cope with thousands of nodes with a web more complex than simple linear or hierarchical relations.

The usual approach to this problem is to generate a view of the neighborhood around the current position. A straightforward approach is to show a "zoom out" or "road map" schematic view of the adjoining nodes. Such views can be automatically generated at the time of request, built manually during the editorial process, or defined by the user. Allowing the latter possibility is very interesting, because the construction of such a view by the user is a direct statement of perceived relevance among the documents. Such views might evolve naturally from collections of bookmarks. Most current techniques presume automatic generation, but have immediate application to precomputed or editorially created neighborhood views.

The neighborhood view can take many forms, all in the general class of contiguity maps. Print conventions include timelines and maps. We might borrow conventions from technology such as PC board layout with documents corresponding to packages and links to traces. A viewing method adapted from semantic net browsers is simulated 3-dimensional space [Fair86]. This deserves more investigation, but appears to have problems with screen clutter, complexity of controls to manipulate the view, and lack of obvious mapping to the documents themselves. "Spaghetti looks the same from any perspective" - Steve Weyer.

Any of these views can be overwhelmed by heavy branching at a node or attempts to view at several links distance from the current document. As the number of nodes in sight grows, the view either becomes cluttered or is forced to multiple screens, defeating the purpose of synopsis. Methods called filtering and focused or distorted views help overcome this problem.

Filtering is removal of documents from the view based on a simple criterion [Kay83]. If the documents are typed, a subset of these types may be displayed. If links are typed, the view may be generated by moving along a subset of link types. In some systems, such as Brown University's Intermedia, links may actually be loaded and unloaded in related sets. In

hypertexts with hierarchical structure, the filtering could create views up, down, and sideways in the hierarchical levels. If the current document contains narrative or tour links, they are direct editorial cues to location and other documents on the paths should be preferentially displayed. All filtered views have the characteristic of requiring the user to manipulate the filtering criteria, rather relying on automatic function of the hypertext system. Filtering is also be antagonistic to serendipity, as it removes the chance of accidental discovery of an unanticipated type of relation.

Distorted views use more complex criteria for rejection or inclusion of nodes in the view. The basis for choice typically extends beyond simple link or node types to include global or regional properties of the database. For instance, the fisheye viewing technique developed by George Furnas shows samples from the database in relation to their distance from the viewing point [Furn86]. In a fisheye view of a linear list, all items near to the current position would be shown, with more and more entries omitted farther from this focus point. By changing the degree of interest function associated with a fisheye view, the type of distortion produced can be varied.

Clusters are another type of distorted view. Each document is assigned to one or more clusters. The neighborhood view from a node shows adjoining clusters rather than single documents. One document may be chosen to stand for the cluster, or a separate designator may be generated. Attempts to automatically generate clusters may rest on similarity measures of the text within the documents, or on analysis of the pattern of connectivity amongst nodes. At KnowledgeSet, an attempt was made to cluster on the basis of minimizing connectivity between clusters, using untyped links derived from textual analysis of a print original [Oren86]. This project failed, probably due to a lack of editorial control of the links and very high connectivity among a large subset of the documents. Typing of the links, or prior design of the link and documents to fall into clusters may be necessary for success.

Like other views, clusters can be deliberately created as part of the editorial process, perhaps as an extension of the regular hierarchical outlining process. One can also allow the user to assign documents to self-defined clusters upon retrieval. The interface might consist of a small form with category check boxes, or a collection of "rubber stamp" icons to be applied to the document. This feature does not solve the general location problem, but does allow the user to build up a personal information space with invented clusters and names.

These neighborhood views share the property that they are deliberately invoked by the user. Other location methods attempt to be less intrusive, providing subliminal or periodic cues without explicit request. For example, transitions between documents might include a segue in which the browser zoomed back to a cluster view, panned to the cluster of the new document, and zoomed back in to the text level. Colors, background patterns or icons could be used to indicate the topic of a document or its level in a hierarchy. In a multimedia system, sound might be a peripheral cue to location.

AUGMENTING FULL TEXT SEARCH

Boolean full text search has become the de facto retrieval method for CD-ROM for a number of reasons. Users, implementers and data providers with experience in online retrieval are accustomed to Boolean search and have carried that experience into the new medium. It is easy to precompute the inversion tables required in Boolean search and optimize their layout on the CD-ROM medium. The technique also makes modest demands on the processor power available in desktop computers.

However, Boolean search is well known to be flawed. A severe tradeoff has been demonstrated between precision, the portion of retrieved items which are actually of interest, and recall, the fraction of actually relevant items which are found in the search [Blai85]. This failing may be attributed to imprecise or incomplete choice of search terms by users [Furn83], lack of semantic analysis of the documents, and the fact that relevance is simply not a binary decision. Whatever the reasons, unaugmented Boolean search is not a good technique to impose on a desktop computer user who has no experience with the tradeoffs involved, and who may be unwilling to formulate complex queries. Particularly with sensitive databases such as law or medicine, a false impression of completeness is dangerous, and a plethora of irrelevant documents may overwhelm the time available for research. Probabilistic and weighted Boolean methods of full text search [Salt83] offer substantial improvements, but have been slow to appear on CD-ROM due to the processing load they impose and greater difficulty of implementation. In the interim, The network structure of hypertext databases may allow improvement in the apparent performance of Boolean search [Bart85].

Hypertext links can allow text search to fail gracefully. If links joining related nodes have been built into the database, failure to find every relevant document is no longer so important. Assuming that the target of a search is in fact a set of documents with related semantics, it should suffice to retrieve a subset of these documents and reach the remainder by browsing via hypertext links. As link following replaces exhaustive scanning of a list of retrieved documents, the presence of false hits in this list becomes less obnoxious to the user.

The hypertext network can also be used as a direct component of search. The analogy of a hypertext to the interlinked body of scientific literature has already been noted. Bibliographic and cocitation analysis have been shown to dramatically increase search performance in this setting, presumably because the links directly trace the evolution of ideas [Bich80]. These techniques could be used to augment search of a CD-ROM hypertext where the links shared this property. The user would mark documents of known interest. Following and comparing incoming and/or outgoing links would generate a candidate document set which could be included as a term in Boolean search or used in a weighted search.

Even when the links are not explicitly bibliographic, the hypertext network can be used in relevance feedback. Spreading activation along links from user marked nodes can be used to generate document weights. Marking could be explicit, or the browser could simply record the nodes visited by the user since the last search, with an option to forget a document which was a false path.

The onus is on the hypertext designer to ensure that these techniques will work. The necessity is that links be true reflections of semantic relevance, if not exact bibliographic relation. If there are links in the database which do not indicate relevance, e.g., links between articles with the same date, the hypertext system should be able to associate relevance weights with links. In a hypertext system with explicitly typed links, the relevance weight could be associated with type.

IMPROVING HUMAN COMMUNICATION

For hypertext to succeed as a medium, it must offer advantages beyond print or isolated documents stored in an electronic form. Its particular potential on CD-ROM is to overcome the inherent limitation of a static medium to communication without feedback.

Both common sense and experiences suffered by AI practitioners indicate that an idea exists in the human mind as part of a dense web of entities and relations. Nonetheless, we manage to pass ideas from mind to mind using the medium of spoken language. Language itself is a

delicate balance between efficiency and breakdown. Efficient language is terse and evocative, resting on an assumption of similarity of concepts between the communicating parties. When this fails, breakdown occurs. When no continuity of ideas exists, the failure may be permanent. More often, an iterative process of exchanging concepts and examples eventually reestablishes communication.

The weakness of static media, whether print or electronically based, is the inability to support such iterative communication of ideas. Written language has adapted by being lengthier and more formal and precise. However, if breakdown does occur, there is no general mechanism for obtaining a meaningful elaboration, example, or definition of terms. At best, a conscientious author will provide sample problems with solutions, references to other works, and example applications of an idea. These may or may not fit the needs of a given reader, and often go unused because of the time and labor required.

Implementing hypertext on the static medium may ameliorate this difficulty in communication. Hypertext honors the connectedness of memory by making relations explicit, allowing them to be explored until the idea is clear. Hypertext jumps are less laborious than thumbing through a book or returning to the shelves, and should do much to encourage investigation when doubt occurs. Thus hypertext suits the static medium for communicating ideas, as opposed to transmitting disconnected bits of information.

Cognitive considerations

The capacity of the CD-ROM medium makes it possible to store a great number of possible paths of inquiry, hopefully anticipating the needs of most viewers. The design problem for the hypertext architect is anticipating and building the most useful paths, given that it is impractical to elaborate all paths due to production costs and time. This section proposes some rules for choosing useful paths, based on what we know and guess about human cognitive processing.

Beyond the economic limits on construction, there may be optimum and maximum numbers of links per document determined by the limits of human cognition. A small hypertext constructed in Xerox's Notecards system has an average of slightly greater than two links per card [Conk86]. A large hypertext automatically generated from Grolier's Encyclopedia also has an average of about two links per documents, with 80% of the documents having 10 or fewer links [Oren86]. Other experimenters with constructed hypertexts report ranges of four to eight, and two to ten links per screen [Kaeh87][Shne87].

It is suggestive that these connectivity figures mostly lie within the number of items considered optimal for selection from menus: "seven plus or minus two" [Mill56]. If this number represents a fundamental human limit on simultaneous consideration, the consequences of exceeding it in hypertext documents should be obvious. We would expect choosing a path from such documents to take longer, because of the need to reload links while reaching a decision. This phenomenon needs further investigation.

A related question is the optimum size of a document in a hypertext system. Because of the explicit representation of relations, it is not clear that print conventions are meaningful in this regard. In fact, limited experience again suggests that documents created in a hypertext system tend to express one idea, and are smaller than traditional print documents or text files [Conk86]. This fits well with the notion that links capture relations between ideas.

If we accept these suggestions, the burden on the author and editor of a CD-ROM hypertext

becomes clearer. As yet, we have no automatic tools to extract idea sized chunks from linear text. The requirement for parsimony of linkage suggests that just as in print, what is not said is just as important as what is represented. The burden of expression and selection falls squarely on the human content creator.

Contiguity and Similarity

In searching for the best types of links for inclusion in a hypertext, we should look at existing taxonomies of ideas and their associations, and methods used in present static media to express these relationships. Aristotle defined two associative principles, similarity and contiguity [Mand84]. Objects in a similarity relation share properties, but no further association; they are members of a set or class. On the other hand, contiguity puts objects and events into a spatial or temporal relation. The relation may be very simple, as in a time line, or very complex, as in an organized scene or script.

Contiguity relations are more readily learned and recalled than abstract concepts or hierarchies [Mand84]. This is not surprising, since contiguity is the rule in our everyday life. All of our experience occurs in contiguous time and space, and the human is adapted to survive by quickly storing this experience, as episodic memories, and later create generalizations with value in predicting the future by the process of abduction [Rosz86][Bate79].

Contiguity may be represented verbally, textually, or visually. In print media, contiguity relationships are shown with time lines, maps, or narrative. Maps can directly represent a complex physical or conceptual space, and may translate directly into the electronic medium when an appropriate metaphor can be found. Verbal narrative as a learning tool covers a range from aboriginal cultures such as the Australian Bushmen to the "war stories" that circulate in technical professions [Snyd83]. Print narrative is present not only in fiction, but in biography and history texts which transfer a lesson by telling a story.

Computer scientists have represented contiguity relations as frames, scripts, and schemata [Scha77]. Attempting to understand stories with these tools, we find that the information explicitly transferred is not enough for understanding, that text assumes underlying knowledge in the reader [Lena86]. In fact, we have effectively rediscovered the narrative convention that expected behavior is not mentioned [McLu64][Mand84]. We might say that the value of a narrative is precisely in its novelty, in deviation from the expected script or schema. Again, we can make the argument that we are evolved to learn by exception. If generalizations have value in predicting the future, an exception can be deadly. Our learning pattern is consequently skewed to quickly note the novel. Our media achieve parsimony by congruence with this phenomenon; by evoking pattern and articulating exception.

Here is precisely the assumed "common ground" of static media which we expected. The narrative convention causes communication breakdown when the reader does not have the appropriate abstraction in mind and is unable to extract it from the story. This type of failure means that the experience cannot be generalized and reapplied.

We use similarity relations to codify and retrieve knowledge. We build hierarchies of classification exhibiting the common elements of experiences, objects, and concepts. Similarity relations capture the generalizations deduced from narrative and experience. Similarity enables metaphorical and analogic reasoning by exhibiting common features among the related elements.

Similarity relations are shown in print with outlines, tables of contents, formal taxonomies, and encyclopediac classifications. Most textbooks are organized into a hierarchy of topics to be defined discursively. In the electronic medium, key term indices define similarity classes, and

structured documents and outline processors emulate print conventions for classification.

The difficulty of communicating similarity relations reflects their very generality. There must be some commonality of experience between author and reader in which to ground the argument, or the concepts abstracted become meaningless buzzwords. In this situation, the recipient is unable to learn the similarity relation because it cannot be restated in familiar terms. Textbooks provide specific examples to overcome limits of experience, but there is a practical limit in breadth and depth of coverage. Some level of prior experience must be assumed.

Supporting multiple relations in hypertext

We can see a complementarity of capability and limitation between contiguity and similarity, between narrative and classification. Hypertext fuses the two by breaking the one track limit of linear text. In a single hypertext we may embody multiple hierarchies of classification and many trails of narrative and experience. When a narrative is unclear due to a missed concept, the reader may pause to examine other documents in a similarity class. If an explanation of similarity becomes murky, a trail of examples can be followed. The user retrieving desired facts can move smoothly into the learning experience, and vice versa.

The ability of hypertext to represent multiple hierarchies breaks a limit of printed outlines. It allows the exposition of multiple points of classification with greater opportunity for discovery of analogy and metaphor. Different hierarchies can also embody competing views of the same experiences. Consider, for instance, how a Marxist and capitalist might classify and explain the same set of news stories. Limited experience shows that multiple hierarchies do occur in constructed hypertexts [Mont86b].

The multiple narratives in a hypertext correspond to Bush's memex trails [Bush45] and are also called tours or paths in the hypertext literature [Yank85]. Hypertext is a multivoice medium. Tours could embody differing pedagogical approaches to the material, or different points of view by the reader. For example, the same generalized material on ecology might be illustrated with different examples depending on the bioregion of the viewer. A tour can simply determine the next document presented by default, or it could be a dynamic presentation with realtime sound, video, and animation components.

DESIGN FOR A RANGE OF INTERACTIVITY

Note that we have moved from a user actively searching for documents to a learner acquiring concepts and experiences through both passive and active communication with the database. This ability to vary the nature of human/computer interaction within a hypertext system deserves exploration. I believe there is a spectrum of interactivity. At one end of the range is the active human, passive computer combination. This is the world of the user, the goal directed seeker of information with the computer performing search and retrieval on demand. This is the situation addressed by our cognitive models of problem solving behavior, and by user interface designs for productivity software. The user mode of interaction is also apparent in learning by exploration.

At the other end of the spectrum is the passive human and active computer combination. The human role as observer or viewer is found in many educational situations such as lectures and films. The lecture model gives the student a learnable narrative tour through material, explicitly demonstrating connections and relieving the cognitive load of navigation during learning [Conk86]. We may also note that every successful entertainment medium to date presumes the passive viewer model.

We are looking for a position in the middle of the spectrum where the human and computer both move between the active and passive roles. We might call the human in this situation a participant. A participant system is characterized by graceful yielding and recovery of control. There is a fluid transition between roles, the human may interrupt or redirect a default presentation at any point. The best educational simulations and adventure and arcade games fit this model. However, most such games embody very simple models of the computer as antagonist or riddlemaster, so the experiences generated without human input are unlikely to be satisfying.

A hypertext system fits the participant model when it includes elements such as saved tours which can be activated and viewed, and cancelled or paused at any time. During a pause the user can become an active explorer or experimenter and then resume the paused tour or switch to another tour. Memory in hypertext systems as described earlier also fit this model, for the user can turn control over to the computer, asking for review of previous material or access to novel documents.

What is the role of the computer in a participant system? In many cases it will simply act as a playback device, delivering a tour or experience created by the database author. In other systems the computer will be active, adapting to actions by the user and to properties of the documents and links being accessed. This role for the computer is very close to the software "agent" model proposed by Alan Kay [Kay84].

I suggest that an agent in a hypertext database need not have a deep understanding of the documents themselves, so long as it has a model of the conventions of the hypertext links, and the database follows these conventions uniformly. A parallel would be the human reference librarian who does not comprehend the material in articles being sought, but does understand the conventions of card catalogs, abstract collections, citation indexes and bibliographic references. Because these relations can be made explicit in a hypertext they can be utilized without, for instance, having any deep comprehension of the meaning of any article title. Such a hypertext agent would always be delving ahead of the user, calling up documents suggested by link patterns, rating them for relevance to the current topic and readying them for display. The user could either accept these suggestions or continue with a query strategy.

In a hypertext which embodies the patterns of narrative and classification discussed earlier, the agent might act as a teacher in a limited fashion, choosing to switch between elaboration by example or explanation by appeal to similarity. Such an agent could be explicit request of the student, or could present questions on material scanned and choose a strategy based on the response. A hypertext with planned structure of this sort also lends itself to a "quiz by demonstration", asking the student to traverse links to documents forming an example of a principle under discussion. A reasonable assignment from a human teacher might be to find a path between two documents, and explain the relations between every node on the path. Such a testing strategy encourages exploration, and puts emphasis on relations between ideas rather than memorization of isolated facts [Weye86][Rosz86].

This all suggests that hypertext may lead us to a reexamination of our models of human-computer interaction. Analysis and design to date has focussed on planning and problem solving behavior, on goal directed productivity applications. If we wish to design successful CD-ROM products for education and entertainment, the same level of analysis must be applied to the differing demands of these roles.

CONCLUSIONS

I have tried to show that CD-ROM hypertexts form a new medium, which not only opens new

possibilities but lets us augment material originated in other media. We are just beginning to understand the uses of hypertext and the issues it raises for interface and database design. Like any new technology, our design efforts will be accompanied by much trial and error. The task for technologists and publishers alike is to be conscious of what we are doing, meanwhile minimizing the costs of experimentation.

One of my themes has been the need for structure deliberately built into the hypertext database. The creation of carefully crafted electronic content is exactly analogous to the production value now added to print works by editors, designers, and publishers. Successful products in the new medium will require just as much creative effort, though it may take different form. Some of the forms will be unique, but many will be borrowed from print, video, and audio. Publishers need to be searching for these forms, and preparing to continue their role in the new medium.

The task for technologists is to keep the cost of this experimentation and creation low. The tools for building hypertexts and their component documents must be easy to use, lowering the barriers to content experts and creative artists. If a capability of the new medium is ill supported, it might as well not exist, because few creators will interrupt their work to struggle with poor or non-existent tools. There must be fast feedback between creation and testing, so that experiments can be evaluated. A once-through data preparation to final disc process is unacceptable. The authoring tools must be fully capable of demonstrating the behavior of the product in its delivery environment. Finally, the quality of tools directly determines the feasibility of database projects. Every authoring system has a "gain" that relates the production values achieved to the time and cost expended. If this performance is poor, few products will be economically possible.

CD-ROM has already won a position as an archival system for existing data. Existing standards and retrieval systems reflect this role. To go further, opening new markets and competing with other media, databases and retrieval systems must be specifically crafted for CD and the new audiences it can reach. This will happen if technologists and publishers ally in a conscious search for new forms. The hypertext technique is a good basis for beginning this search.

BIBLIOGRAPHY

- [Bart85] Bartschi, M., "An Overview of Information Retrieval Subjects", *IEEE Computer*, May, 1985, pp. 67-84.
- [Bate79] Bateson, G., *Mind and Nature*, Bantam, New York, 1979.
- [Bich80] Bichteler, J., Eaton, E. A. III, The Combined Use of Bibliographic Coupling and Cocitation for Document Retrieval, *J. Am. Soc. Inf. Sci.*, July 1980, pp. 278-282.
- [Blai85] Blair, D.C., Maron, M. E., "An Evaluation of Retrieval Effectiveness for a Full-text Document-Retrieval System", *CACM*, 28(3):289-299, March 1985.
- [Bush45] Bush, V., "As We May Think", *Atlantic Monthly*, July 1945, pp. 101-108.
- [Conk86] Conklin, J., *A Survey of Hypertext*, MCC Technical Report STP-356-86, Microelectronics and Computer Technology Corporation, October, 1986.
- [Deli86] Delisle, N., *Neptune: A Hypertext System for CAD Applications*, CR-85-50,

Tektronix Computer Research Laboratory, Beaverton, Oregon, January 1986.

- [Enge68] Engelbart, D.C., English, W.K., "A Research Center for Augmenting Human Intellect", *Proceedings Fall Joint Computer Conference*, 1968, pp. 395-410.
- [Fair86] Fairchild, K.M., Poltrock S., "Semnet", Videotape Program, ACM-SIGCHI, Boston, MA, 1986.
- [Fost85] Foster, E., "Outliners: A New Way of Thinking", *Personal Computing*, May 1985, p. 74.
- [Furn83] Furnas, G.W., Landauer, T. K., Gomez, L. M., Dumais, S. T., "Statistical Semantics: Analysis of the Potential Performance of Key-Word Information Systems", *Bell System Technical Journal*, 62(6):1752-1805, July, 1983.
- [Furn86] Furnas, G.W., "Generalized Fisheye Views", *Proceedings CHI'86*, Boston, Massachusetts, April 13-17, 1986, pp. 16-23.
- [Garr86] Garrett N.L., Smith, K.E., Meyrowitz, N., "Intermedia: Issues, Strategies, and Tactics in the Design of a Hypermedia Document System", in *Proceedings of the Conference on Computer-Supported Cooperative Work*, MCC Software Technology Program, Austin, Texas, 1986, pp. 163-174.
- [Greg83] Gregory, R., "Xanadu: Hypertext from the Future", *Dr. Dobb's Journal*, 75:28-35, January 1983.
- [Guid86] Guide Software for Macintosh, contact: Owl International, Inc., 14218 NE 21st Street, Bellevue, WA 98007, (206) 747-3203.
- [Gull86] Gullichsen, E., D'Souza, D., Lincoln, P., Casey, T., *The PlaneTextBook*, MCC TR N. STP-333-86(P), 1986.
- [Gold83] Goldberg A., Robson, D., *Smalltalk-80: The Language and its Implementation*, Addison-Wesley Publishing Co., Menlo Park, CA, 1983.
- [Hala87] Halasz, F., Moran, T., Trigg, R., "NoteCards in a Nutshell", *Proceedings CHI+GI '87*, Toronto, Canada, April 5-9, 1987.
- [Hers85] Hershey, W., "Idea Processors", *BYTE*, June 1985, p. 337.
- [Kaeh87] Kaehler, T., personal communicaton.
- [Kay83] Kay, A.C. "New Directions for Novice Programming in the 1980s", in *Programming Technology*, P. J. L. Wallis, ed., Pergamon Infotech, Elmsford, NY, 1983, pp. 209-247.
- [Kay84] Kay, A.C., "Computer Software", *Scientific American*, 251(3):53-59, September 1984.
- [KRS86] KRS (Knowledge Retrieval System) Software for IBM PC, contact: KnowledgeSet Corporation, 2511C Garden Road, Monterey, CA 93940, (408) 375-2638.
- [Lena86] Lenat, D., Prakash, M., Shepherd, M., "CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks", *AI Magazine*,

6(4):65-85.

- [Love84] Lovell, B., *Science Digest*, June 1984, p. 91.
- [Mand84] Mandler, J.M., *Stories, Scripts, and Scenes: Aspects of Schema Theory*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1984.
- [McCr84] McCracken D. L., Akscyn, R. M., "Experience with the ZOG Human-Computer Interface System", *Int. J. Man-Machine Studies*, 21(4):293-310, October 1984.
- [McLu64] McLuhan, M., *Understanding Media: The Extension of Man*, 1964.
- [Meyr86] Meyrowitz, N., "INTERMEDIA: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework", *Proc. OOPSLA 86*, ACM, Portland, Oregon, September, 1986.
- [Mil156] Miller, G.A., "The magical number seven, plus or minus two; some limits on our capacity for processing information", *Psychological Review*, 63:81-97, 1956.
- [Mont86a] Monty, M.L., "Temporal Context and Memory for Notes Stored in the Computer", *SIGCHI Bulletin*, 18(2):50-51.
- [Mont86b] Monty, M.L., Moran, T.P., "A Longitudinal Study of Authoring using Notecards", *SIGCHI Bulletin*, 18(2):59-60.
- [More86] More, Software for Macintosh, contact: Living Videotext, 2432 Charleston Road, Mountain View, CA 94043, (415) 964-6300.
- [Nels74] Nelson, T.H., *Dream Machines*, Hugo's Book Source, Chicago, IL, 1974.
- [Nels80] Nelson, T. H., "Replacing the Printed Word: A Complete Literary System", in *Information Processing 80*, S.H. Lavington (ed.), North-Holland Publishing Company, IFIP, 1980, pp. 1013-1023.
- [Nels82] Nelson, T.H., "A New Home for the Mind", *Datamation*, March 1982, pp. 169-180.
- [Oren86] Oren, T., Kildall, G., Rolander, T., "Experiences with Hypertext on CD-ROM", unpublished paper.
- [Rese86] Research Software for IBM PC, contact: TMS Inc., 110 W. 3rd Street, P. O. Box 1358, Stillwater, OK 74076, (405) 377-0880.
- [Robe81] Robertson, G. , McCracken, D. , Newell, A., "The ZOG Approach to Man-Machine Communication", *Int. J. Man-Machine Studies*, 14:461-488, 1981.
- [Rosz86] Roszak, T., *The Cult of Information*, Pantheon, New York, 1986.
- [Salt83] Salton G., McGill, M.J., *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, 1983.
- [Scha77] Schank, R., Abelson, R., *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1977.
- [Scra85] Scragg, G.W., "Some Thoughts on Paper Notes and Electronic Messages", *SIGCHI*

Bulletin, 16(3):41-44, January 1985.

- [Shne83] Shneiderman, B., "Direct Manipulation: A Step Beyond Programming Languages", *IEEE Computer*, 16(8):57-68, August 1983.
- [Shne86] Schneiderman, B., "User Interface Design and Evaluation for an Electronic Encyclopedia", *Proc. 2nd Int'l. Conf. on Human-Computer Interaction*, August 10-15, 1987, North-Holland (to appear).
- [Shne87] Shneiderman, B., personal communication.
- [Snyd83] Snyder, G., "Good, Wild, Sacred", *CoEvolution Quarterly*, Fall 1983.
- [Stef86] Stefik, M., "The Next Knowledge Medium", *AI Magazine*, 7(1):34-46, Spring 1986.
- [Trig86] Trigg, R., Suchman, L., Halasz, F., "Supporting Collaboration in NoteCards", in *Proceedings of the Conference of the Conference on Computer-Supported Cooperative Work*, MCC Software Technology Program, Austin, Texas, 1986.
- [Weye86] Weyer, S., "As We May Learn", in *Multimedia in Education: Interfaces to Knowledge*, Apple EAC (in press).
- [Wind86] Window Book Software for IBM PC, contact: Box Company, 56 Howard Street, Cambridge, MA 02139, (617) 576-0892.
- [Yank85] Yankelovich, N., Meyrowitz, N., van Dam, A., "Reading and Writing the Electronic Book", *IEEE Computer*, October 1985, pp. 15-30.

Note: a version of this paper was presented at the Strategies for Information Access Workshop of the 2nd International Conference on CD-ROM, Seattle, WA, March 3, 1987.

Document Examiner: Delivery Interface for Hypertext Documents

Janet H. Walker

Symbolics Inc.
11 Cambridge Center
Cambridge, MA 02142

ABSTRACT

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

INTRODUCTION

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

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

This paper describes the user interface strategy used in Document Examiner, an end-user interface for commercial hypertext documents⁵. Document Examiner is part of Symbolics Genera⁶, the software development environment (operating system) for Symbolics computers.

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

In order to set the context, I would like to outline some of the factors for categorizing hypertext systems and contrast our system with two typical, widely known hypertext systems, Xanadu⁷ and NoteCards⁴:

- What does it hold? We deliver documentation that is part of a larger software product. Xanadu was designed to manage a library of prepublished material, Notecards to organize a set of working notes.
- How is it organized? Our documentation is highly structured. The material in library-like systems has few inherent interrelationships. The material in Notecards is highly interrelated but typically not highly structured.
- Who writes it? Our document is prepared by a small group of cooperating writers. The documents in Xanadu were to be submitted by authors and the annotations by readers; in Notecards, each user serves as both writer and reader (although recent work⁸ has explored using Notecards for collaborative work.
- How much does it change? Our document is under constant maintenance, with revised versions published with every software release. It has more changes than a library but is more static or controlled than the information in Notecards.
- How big is it? Our documentation corresponds to 8000 printed pages with about 10,000 nodes and 23,000 links. This is small by "global library" standards but large in comparison to personal notes.
- Where does it fit in? This work has commercial rather than academic roots and production rather than research goals.

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

DOCUMENT STRUCTURE

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

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

Modularity

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

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

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

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

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

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

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

Relationships

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

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

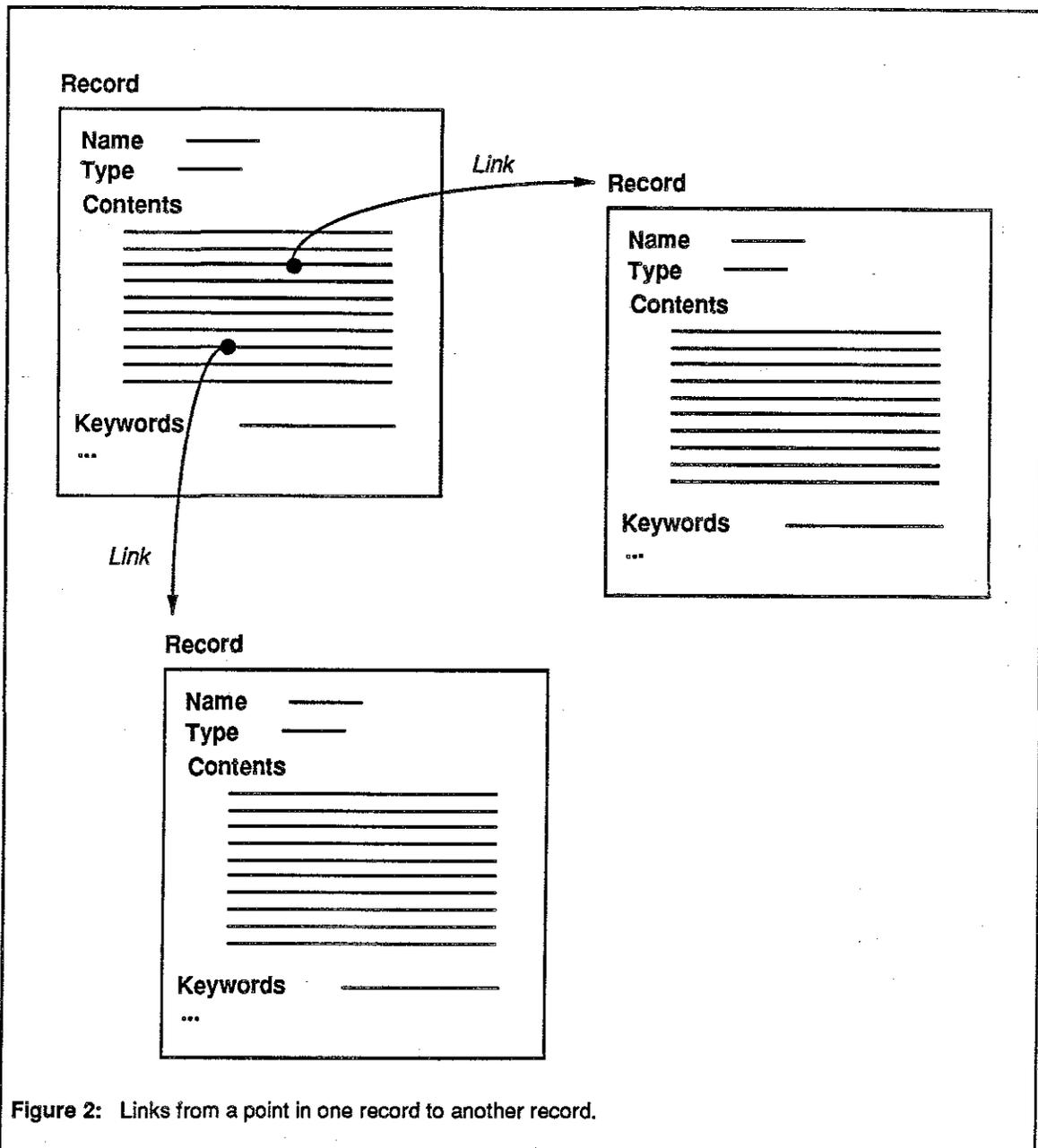


Figure 2: Links from a point in one record to another record.

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

A taxonomy of different kinds of links has been proposed by Trigg¹². At this time, we have only one kind of link and we support several different kinds of views for it:

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

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

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

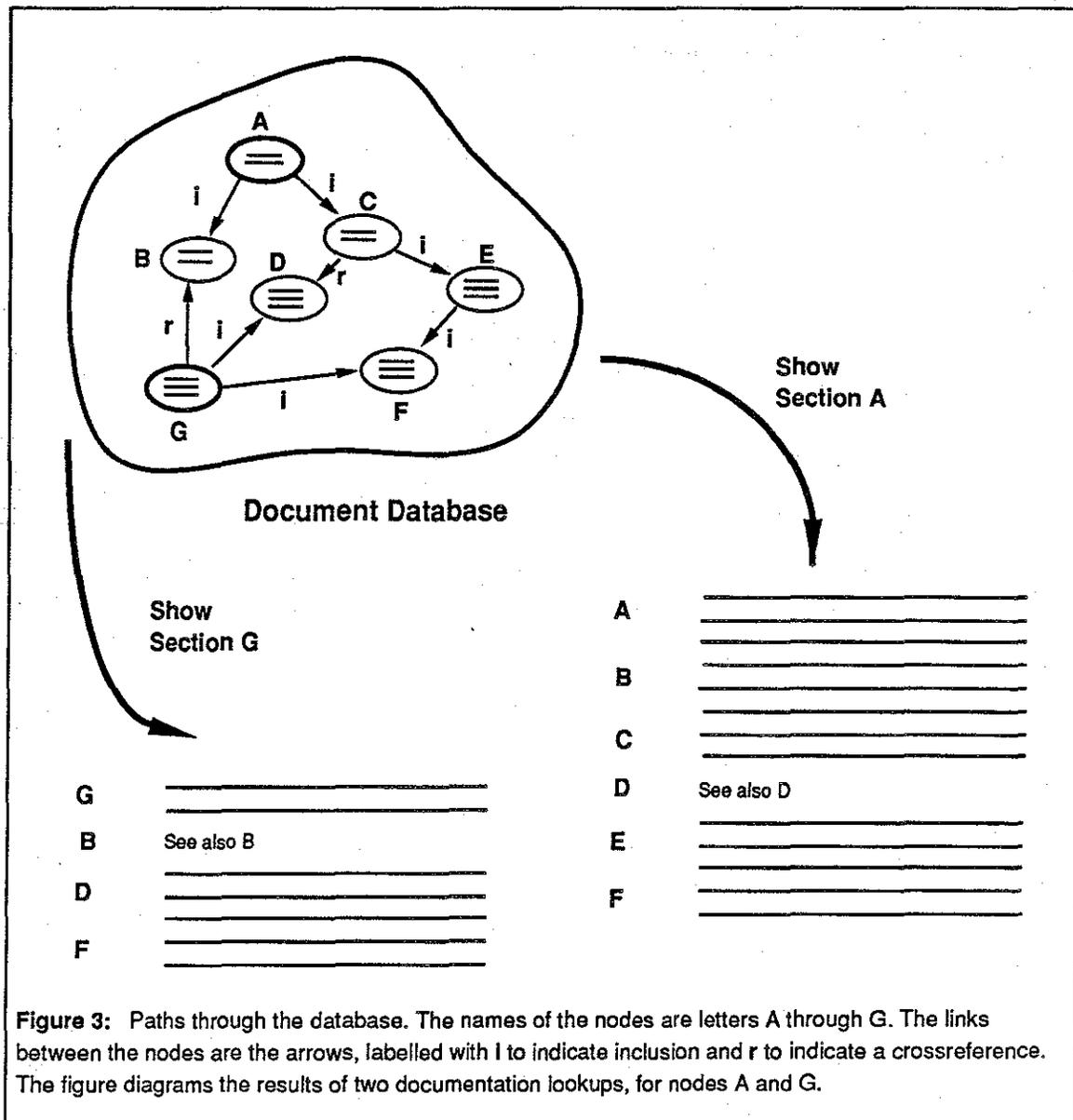


Figure 3: Paths through the database. The names of the nodes are letters A through G. The links between the nodes are the arrows, labelled with i to indicate inclusion and r to indicate a crossreference. The figure diagrams the results of two documentation lookups, for nodes A and G.

Versions

Some hypertext systems incorporate concepts that are usually addressed under the topic of version control or configuration control in software development systems¹³. This is primarily a document structure or management issue as opposed to an interface issue. For configuring the document, our document database uses the general system configuration tools¹⁴ available with Symbolics Genera. In addition, writers can view either the published version of a record or the version(s) they are currently working on.

INTERFACE REQUIREMENTS FOR HYPERTEXT DOCUMENT DELIVERY

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

In the near term, the problem facing designers of hypertext delivery interfaces is exactly that posed by Jeff Conklin in his description of using hypertext¹:

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

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

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

What does a paper manual provide?

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

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

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

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

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

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

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

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

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

What can an electronic manual provide?

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

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

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

DOCUMENT EXAMINER INTERFACE DESIGN

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

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

The screenshot shows the Document Examiner window with the following components:

- Title Bar:** Document Examiner
- Main Text Area:**

Predefined Presentation Types

Presentation types form the basis of the typing system for user input and program output. A large number of predefined presentation types exist; relatively few are used for program I/O. This is because every structure, flavor, and Common Lisp data type is also a presentation type. Most, however, are of little use in end-user-oriented application programs. Consider, for example, the Common Lisp types `hash-table` and `compiled-function`; you would not generally encounter these in end-user-visible places.

In this section, we list what we regard as the types most likely to be used by application programmers. Some, like `integer`, `string`, and `boolean`, are encountered frequently in all kinds of programs. Many others, like `sys:code-fragment` and `net:network`, are more specialized in their uses.

In any case, all of the types included here are also documented as individual entries in the Dictionary of Predefined Presentation Types. Also, many of them are defined in the file `sys:dynamic-windows;standard-presentation-types.lisp`, where you can look for models when defining your own types. The dictionary entry for each type notes whether it is one of those included in this file.

The documented types are divided into three groups:

 1. Common Lisp Presentation Types
 2. Symbolics Common Lisp Presentation Types
 3. Other Presentation Types

Of course, the Common Lisp types form a subset of the Symbolics Common Lisp types, but for the purposes of the present discussion, we separated them out. The Other Presentation Types include the potentially useful types exported from packages other than Symbolics Common Lisp; most of them are in the specialized-use category.

The following table lists the useful Common Lisp presentation types:

Common Lisp Presentation Types and character

Viewer: Standard (Reader)
- Current Candidates Pane:**
 - Presentation Substrate Facilities
 - Basic Presentation System Concepts
 - Predefined Presentation Types
 - Meta-Presentation Arguments to Presentation
 - Inheritance of Presentation Arguments
 - Presentation-Type Definition Facilities
 - Presentation Input Context Facilities
 - Presentation Input Blip Facilities
 - Other Presentation Facilities
 - Writing a Presentation Type Parser
 - User-Defined Data Types as Presentation Types
 - Exploring Presentation Types and Presentation
 - Show Presentation Type Command
 - Show Handlers For Types Command
 - Presentation Inspector
 - Using the Presentation Inspector
 - Invoking the Presentation Inspector
 - The Presentation Inspector's Frame
 - Strategy for Using the Presentation Ins
 - Presentation Inspector Commands
 - Summary of Presentation Inspector Comma
 - Help Presentation Inspector Command
- Bookmarks Pane:**
 - Predefined Presentation Types Section
- Commands Pane:**
 - Show Candidates
 - Show Documentation
 - Show Overview
 - Show Table of Contents
 - Help
 - Select Viewer
 - Reselect Candidates
 - Private Document

Mouse-R: Menu.
To see other commands, press Shift, Control, Meta-Shift, or Super.

[Thu 8 Oct 12:15:19] Keyboard CL-USER: User Input

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

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

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

General description and terminology

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

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

Topic Lookup

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

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

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

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

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

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

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

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

- Show Documentation
- Show Overview
- Show Table of Contents

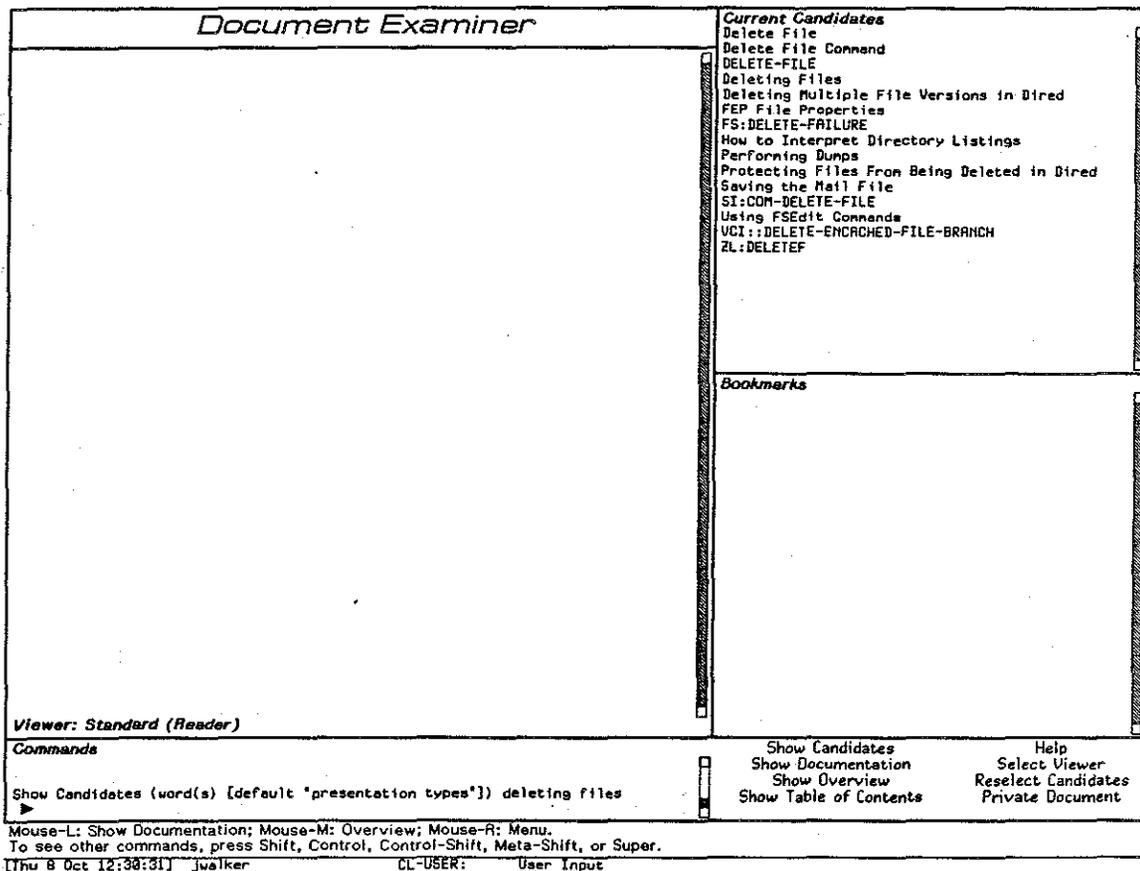


Figure 6: The candidates pane contains the list of candidates resulting from the search for "deleting files".

Examining context and structure

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

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

Document Examiner

<p>Overview</p> <p>Section: "Using Converse"</p> <p>It is included in topics: "Converse", "Talking to other Users"</p> <p>It appears in documents: <i>Communicating With Other Users, User's Guide to Symbolics Computers</i></p> <p>Keywords: Sending Interactive Messages</p> <p>See also: "Customizing Converse"</p> <pre> graph LR TU[Talking to other Users] --- IC1[Introduction to Converse] TU --- UC1[Using Converse] TU --- IC2[Introduction to Converse] TU --- UC2[Using Converse] TU --- CC[Customizing Converse] IC1 --- SR[Sending and Replying to Messages with Converse] IC1 --- DB[Default Behavior of Converse] IC1 --- CC1[Converse Commands] IC2 --- LSP[Lisp Listener Commands for Converse] UC1 --- SR UC1 --- DB UC1 --- CC UC2 --- LSP </pre>	<p>Current Candidates</p> <p>Append Conversation (n-K) Converse Command append conversation by references (n-K) Zmail C Delete Conversation (n-K) Converse Command delete conversation by references (n-K) Zmail C Introduction to Converse Replying to Zmail Messages select all conversations by references (n-K) Zn select conversation by references (n-K) Zmail C Write Conversation (n-K) Converse Command</p> <hr/> <p>Bookmarks</p>
<p>Viewer: Standard (Reader)</p> <p>Commands</p> <ul style="list-style-type: none"> ▶ Show Candidates (word(s) [default "deleting files"]) conversations ▶ Show Overview Introduction to Converse ▶ Show Overview Using Converse 	<p>Show Candidates</p> <p>Show Documentation</p> <p>Show Overview</p> <p>Show Table of Contents</p> <p>Help</p> <p>Select Viewer</p> <p>Reselect Candidates</p> <p>Private Document</p>

Mouse-R: Menu.
 To see other commands, press Shift, Control, Meta-Shift, or Super.

[Thu 8 Oct 12:57:56] jwalker CL-USER: User Input

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

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

This kind of display has significant advantages over either a conventional table of contents or a full display of the graph. It constrains the amount of information that the user has to process while still giving enough relevant information with which to make decisions. (In this sense, it is similar to the powerful "fisheye view" concept¹⁶.)

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

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

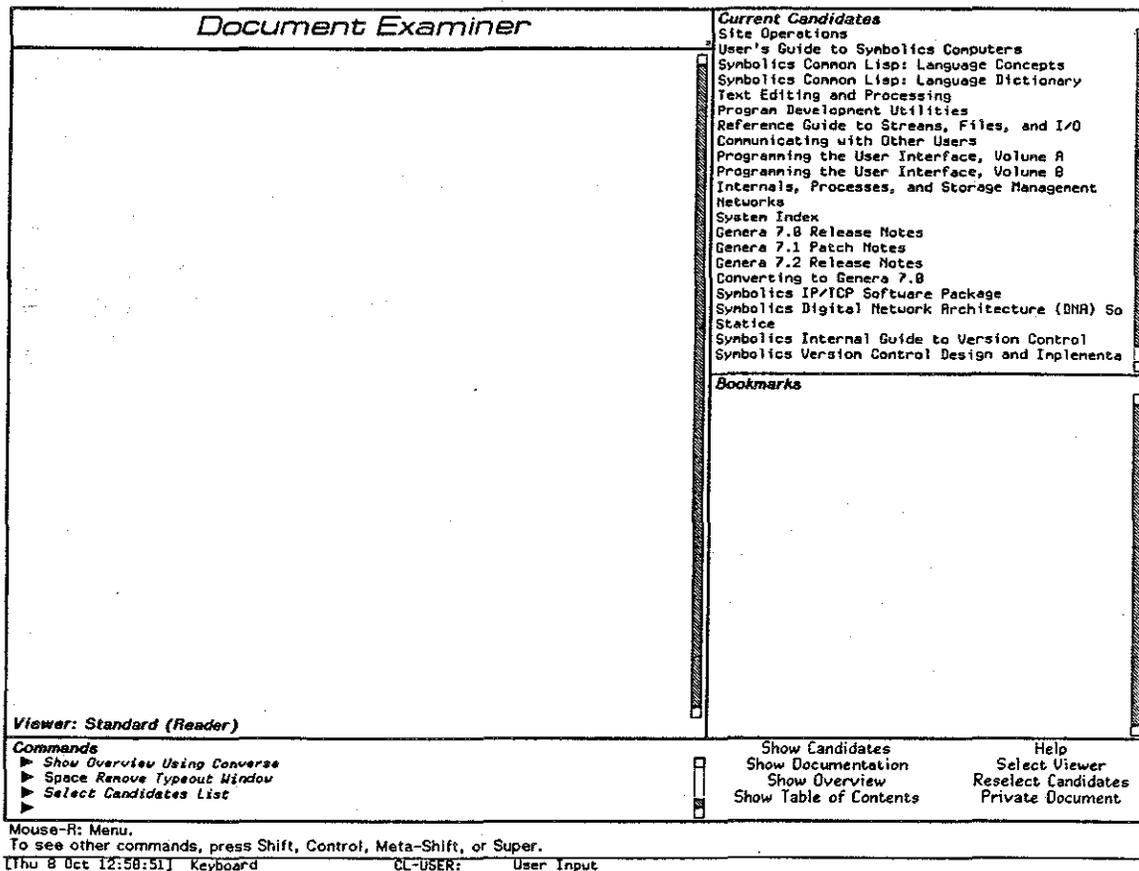


Figure 8: Initial screen configuration of Document Examiner, showing the "top level" directory of documents.

Saving the results of an investigation

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

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

- *Input history.* Using a standard system feature in Symbolics Genera, the user can recapture and edit any command entered earlier in a session.
- *Query result history.* In addition to reactivating earlier commands, a user can select the results of earlier commands. For example, the results of earlier queries can be reinstated in the candidates pane, saving some time but more importantly, eliminating the need for users to remember their exact queries.
- *Lookup history.* When a user asks to see a record, Document Examiner creates a bookmark for it. The set of bookmarks in the bookmarks pane constitutes a chronological record of a user's interactive session. The bookmarks are active, of course, so that a user can reselect a topic by clicking the mouse over the topic's bookmark.
- *Reading context.* Users follow crossreference links freely, suspending reading one topic in order to look at another. Document Examiner saves the user's reading position within a topic so that when they reselect that topic, it is positioned as they left it.
- *Preserving lookup history.* When substantial effort has gone into finding a set of relevant topics, it is useful to be able to save the results of this effort for a future session. The user can save a set of bookmarks in a file called a "private document". The set of topics represented by the bookmarks can then be read in automatically in a subsequent session. This is our approach to the need addressed by Bush's "associative trails" in Memex¹⁷.

EVALUATION

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

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

Document Examiner was first shipped as part of Symbolics software product in April of 1985. In our experience, it is both usable and used. In a year-long usage survey, we found users at all levels of experience used Document Examiner about equally often. Both groups of users looked up large "conceptual" topics as well as short reference ones; Show Overview and Show Candidates commands were used heavily for locating material to read¹⁸.

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

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

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

ISSUES FOR FURTHER WORK

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

- *Locators.* As the documentation being delivered by this kind of interface becomes larger, the index searching capabilities become correspondingly more important. Some of the work now underway in information science in automatic indexing is relevant to hypertext document delivery (for example¹⁹).
- *Annotation.* As in other hypertext implementations, users do need the capability to make notes "on" our documentation. We have approached this problem cautiously, however, since the design issues include helping users maintain their notes across different releases of the system documentation.
- *Context.* Readers often have some need to constrain the set of topics under consideration in searching tasks. Several kinds of constraints:
 - *Structural.* Consider only one particular document (that is, the records in a particular subtree).
 - *Content.* Consider only records that have anything to do with some general topic area (for example, only records related to I/O).

This issue has been addressed by Intermedia with the concept of webs²⁰.

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

CONCLUSION

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

ACKNOWLEDGMENTS

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

REFERENCES

1. Conklin, J., "Hypertext: An Introduction and Survey", *IEEE Computer*, Vol. 20, No. 9, September 1987, pp. 17-41.
2. Engelbart, D. E., "Authorship Provisions in AUGMENT", *Intellectual Leverage: The Driving Technologies*, IEEE Spring Compton84, 1984, pp. 465-472.
3. Robertson, G., McCracken, D. & Newell, A., "The ZOG Approach to Man-Machine Communication", *International Journal of Man-Machine Studies*, Vol. 14, 1981, pp. 461-488.
4. Halasz, F. G., Moran, T. P., & Trigg, R. H., "NoteCards in a Nutshell", *Proc. CHI+GI '87 Human Factors in Computing Systems and Graphics Interface*, SIGCHI Bulletin, April 1987, pp. 45-52.
5. Walker, J. H., "Symbolics Document Examiner", SIGGRAPH Video Review, Vol. 19.
6. Walker, J. H., Moon, D. A., Weinreb, D. L., & McMahon, M., "Symbolics Genera Programming Environment", *IEEE Computer*, Vol. 20, 1987, In press
7. Nelson, T. H., "Literary machines", Published privately by the author, 1981.
8. Trigg, R. H., Suchman, L. A., & Halasz, F. G., "Supporting collaboration in NoteCards", *Proceedings of the Conference on Computer-Supported Cooperative Work*, 1986, pp. 153-162.
9. Walker, J. H., "Symbolics Sage: A Documentation Support System", *Intellectual Leverage: The Driving Technologies*, IEEE Spring Compton84, 1984, pp. 478-483.
10. Walker, J. H., "Supporting Document Development with Concordia", *IEEE Computer*, In press.
11. Walker, J. H., & Bryan, R. L., "An editor for structured technical documents", Paper accepted for Protext IV conference.
12. Trigg, R. H. & Weiser, M., "TEXTNET: A Network-Based Approach to Text

- Handling", *ACM Transactions on Office Information Systems*, Vol. 4, No. 1, 1986, pp. 1-23.
13. Delisle, N. & Schwartz, M., "Contexts--A partitioning concept for hypertext", *Proceedings of the Conference on Computer-Supported Cooperative Work*, 1986, pp. 147-152.
 14. Symbolics Inc., *Volume 4. Program Development Utilities*, Release 7.0 ed., 11 Cambridge Center, Cambridge, MA 02142, 1986.
 15. Salton, G., *The SMART retrieval system--Experiments in automatic document processing*, McGraw-Hill, New York, 1968.
 16. Furnas, G. W., "Generalized Fisheye Views", *Proc. CHI '86 Human Factors in Computing Systems*, SIGCHI Bulletin, April 1986, pp. 16-23.
 17. Bush, V., "As we may think", *Atlantic Monthly*, Vol. July, No. 176, 1945, pp. 101-108.
 18. Young, E., & Walker, J. H., "A case study of using a manual online", Paper in preparation for CHI '88.
 19. Fagan, J. L., "Automatic phrase indexing for document retrieval: An examination of syntactic and non-syntactic methods", *Proceedings of the Tenth Annual International ACM SIGIR Conference on Research & Development in Information Retrieval*, ACM SIGIR, 1987, pp. 91-101.
 20. Yankelovich, N., Meyrowitz, N., & van Dam, A., "Reading and Writing the Electronic Book", *IEEE Computer*, Vol. 18, No. 10, October, 1985, pp. 15-30.

Issues

The Hype in Hypertext: A Critique

Jef Raskin

Information Appliance
1014 Hamilton Court
Menlo Park, CA 94025

ABSTRACT

Hypertext has received a lot of mostly uncritical attention. The author sees it as one part inspiration and nine parts hyperbole. A number of user interface and technical problems are discussed.

A CRITIQUE

The literature on Hypertext is generally effusive and non-critical. Even Conklin's survey article in *Computer* (*IEEE Computer*, September 1987, pg 17 ff) ends up admitting that the author hopes that "the reader come away from this article excited, eager to try using hypertext for himself, and aware that he is 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." This article looks at what some have seen as rough edges but which may be cracks that extend deep into the heart of hypertext.

Conklin's advocacy is tame stuff compared with Hypertext's prime mover, Ted Nelson, who writes with the messianic verve characteristic of visionaries. Many followers and supporters of the Hypertext concept take much the same tone; in fact, there have not been many who have gainsaid the concept, for on the surface it seems a good one that is well within our technical reach. On the other hand, it has not been implemented except for a few more or less experimental projects: the grand vision languishes unfinished, though often started. If the details are kept fuzzy enough, Hypertext seems like a wonderful, universally applicable, powerful, natural, human-oriented model for organizing and accessing knowledge. Having felt that draw, and also having implemented some real-world projects that were considered visionary when I started but became well-accepted (and commercially successful) when I was finished, I took a closer look at Hypertext and found some deep and fundamental difficulties that have not been much discussed.

A good place to begin is with some other ideas that, like Hypertext, sound very good but are simpler, and have failed to deliver on their promise.

There is a certain frustration in playing adventure games. The games ostensibly avoid requiring that you learn special computer commands by allowing you to frame your commands in English. This sounds very inviting, but whenever your response varies from the stylized vocabulary and grammar that the game recognizes you run into a wall of incomprehension. The game spins off a subgame of guessing which words the developer thought useful or significant. Synonyms in

English may or may not be synonyms to the game; you often find that though what you said is correct, it is not accepted. All in all these games tend to be a frustrating experience unless you come to accept that figuring just how to emasculate your vocabulary and grammar is part of the game. Not everybody thinks this is fun, but if you enjoy it, you are a prime example of a puzzle-lover or "hacker" in the old, non-pejorative sense of a person who likes to play with a system — for hours and days on end if necessary — in order to learn how it works.

What is really happening in these games is that the players are learning a set of computer commands — by trial and error. In an adventure-style sword battle your knowledge of fencing is of no use, but your knowledge of the system developers' frame of mind is. The notes that most players end up making for their own use form the manual that the game's inventors (deliberately) didn't provide. In other words, there is not much difference between an adventure game and a poorly documented computer program. Both present the same kind of challenge, and appeal to much the same set of users.

The problem with adventure games was that something that sounds like the right, natural, inevitable and easiest thing to do failed (in some sense) because it delivered only the aroma of what was promised. When you are hungry, this can be worse than nothing at all. The full meal may, in fact, be undeliverable. Natural language may be an inherently unsuitable medium for programming (in the present meaning of the word) or even controlling an imaginary universe.

This insight can be extended to Hypertext. Hypertext, in a nutshell, is text (in the sense of what one finds in books) where there are links between different texts and portions of texts for some very large universe of knowledge. You might be reading, on the display of your information appliance, about butterflies. Say that the text mentions a gathering of monarch butterflies that occurs in Monterey, California. So you point to (by some means) the word monarch, and you press the "delve deeper" button (or some such action) and a picture of the butterfly comes onto your screen. You touch one of the legs of the butterfly and up comes, say, details of the leg of the butterfly or perhaps an article on butterfly legs. You then press the "back to where I was" button, and continue reading. Upon pointing to the word "California" the delve deeper button might give you a map of California with Monterey called out, and so forth.

As technology permits, the concept of "text" in Hypertext expands to include color images, moving images (movies), sounds, real-time links to the author or other experts on the subject you are "reading" about, reference librarians who can point to further sources of information, and literally anything else that might conceivably be pumped through wires or the aether to your system.

Lastly, the Hypertext vision includes a growing body of on-line information, becoming richer as it is used because users provide an ever-increasing number of sources of data and links between items. It assumes wide acceptance by a large number of ordinary, non-technophilic people.

It sounds wonderful.

There are, of course, some technological questions that can be asked about Hypertext: for example, how will we get the very high bandwidths between sources of information and the individual's machine required to make pictures and text available quickly enough to avoid frustration? Another technological problem lies in the massive storage capacity, network,

switching, and software required. There are social, legal, and economic problems as well. Examples of these problems include questions such as: will the financially privileged have easier access, and thus widen the gap between haves and have nots? Nelson has addressed (though without sufficient depth) the questions of who owns the material, who owns the all-important linkages, how copyrights are preserved, who pays for the central system, how to insure compatibility between systems and how an author gets paid. But this essay is concerned with basic conceptual problems, assuming that the technological problems will be solved in the natural course of events, and leaving the other questions for another time. I will also ignore the very real but probably not as serious "chicken and egg" problem of how Hypertext gets started, since at first it is necessarily rather weak and thin. While this last problem will have to be faced, other services, such as telephones faced the same dilemma and overcame it. Whether it gets off the ground depends in part on the quality of the initial system: where will we get enough linked text to make it useful at first?

The problem with "natural language" games, which also applies to many "natural language" interfaces for more serious (this does not imply more worthy) applications, has a parallel in the proposed Hypertext systems. As an example, take the passage where the text mentioned a gathering of monarch butterflies. First of all, not every word will have the same kind or depth of information behind it. Say that a previous user had looked up the meaning of the word "monarch" and established a link to "king." Then, when you point to "monarch" meaning a kind of butterfly, you might find yourself in the midst of a discussion of the divine rights of hereditary rulers. You go "up" and try "butterfly" and you find a general description of the lepidoptera, which does not mention "monarch" since it only gives the Latin name, which you do not recognize. You can grope around for a while, trying this and that sub-heading in what you find, and maybe what you want is there, and maybe it isn't. One key question is: How Do You Know If It's There? This isn't the advertised smooth, rapid access "feel" of hypertext, this is a fishing expedition.

Let's say that, by some means, you can point for specificity to the whole phrase "monarch butterfly" and you get, rather than some particular linked item, a list or menu of such items and other menus. Aside from having to learn how to operate the menus and make selections from them, it is well known that menus are a slow way of getting from place to place, especially if they are many levels deep. Avoiding, also, the question of how (or by whom) the menus are generated (Nelson suggests that people will spontaneously create them and charge for their use), let's say that one menu item leads you to a picture of the butterfly. As before, you point to a leg. Now, it is not clear if you are pointing to, say, the tarsus, the whole leg, to legs in general, to butterfly legs, or to the whole insect. Since butterflies are beautifully symmetrical, are you pointing to ask about symmetry in general?

Not every item or detail in a picture will have a reference attached to it. How will you tell which do and which don't? None of the references about Hypertext seem to carefully address this problem. I tried a medical "Hypertext" styled system a few months ago, based on video disks with immense storage capacity. You could point to a structure in a dissection and get a close up view, or views from other angles. It seemed fine when being demonstrated by someone who knew which body parts had further data behind them, but when I tried to use it and pointed to structures that I was curious about, most of the time the system, being finite, could give me no information.

There is a vague assumption that as people use it, the system will monitor and gather up the links they generate, and thus grow in depth and value. Does a link happen whenever you move from one frame or screenful to another? It will not, according to Nelson. You will have to explicitly tell the system when you are making a link (just how you do this is not specified), and I think it is important to ask if you are likely to bother making links at all. Remember: people are inherently lazy, and when you are hot on the trail of some information how likely are you to stop and tell the system which links seem valuable to you? If as a result of this observation you make the system record all links, good and bad, will there be human editors who will prune them (and where would we get these editors, how would we pay them, and is it in principle even possible to tell a good link from a bad link?) If the links are not pruned, won't most searches turn into wild goose chases? One man's links are another man's sausages.

In other words, it is not clear that it can work as neatly as the hand-waving of its visionaries make it seem. I have been told that there are no bad links, and in a sense this is correct: any connection, even if it is a misreading of, say, "sheer" for "shear," is meaningful and potentially useful (say to a person studying common reading mistakes, or to a poet). But the clutter a user faces will be enormous.

The central lacuna is the omission of any specification of a human interface. The Xanadu project, trying to implement a real-world version of Hypertext, is just building a central processing and storage facility. When I have asked the implementors what it will look like to the user — a central question if it is to be widely accepted, and it must be used by a large population to fulfill the plans of its inventors — they say that the "front end" is not their concern. I claim that the "front end," namely what devices and how it will look to users is as important as the central nexus. It may be more important, since if the front end puts people off, they will never get any further. Yet there is no user interface specification for what Hypertext will look like to the individual user. It is important that the user interface be reasonably uniform from implementation to implementation. This point has been well demonstrated by the Apple Macintosh computer. One of my goals when designing the Mac was to make it easier for a software designer to use a provided interface model than to create a new one. This was a positive use of people's laziness. Thus, unlike any previous system, you can move from application to application with relative ease, and buying a new program is not as traumatic an experience as it used to be. With earlier computers, each new application program gave you a nearly totally new experience.

A Hypertext system will have good "feel" only if it is fast enough. This is another area solved primarily by looking the other way. In ten years the word "work" may easily generate 200 pages of menus of referents (since the word "work," like so many other words, has everything from negative implications (dirty work) to positive ones (work ethic), to technical ones (force times distance), to political ones (ownership of one's labor)). How long will it take the central system to find the menus, how long will it take to transmit them, how long will it take you to go through them, how long will it take you to get to the next submenu? Most people don't use dictionaries and encyclopedias because it takes from half a minute to a few minutes to look something up. If a computer system is as slow or slower, it will be avoided as thoroughly as people avoid other reference systems.

The lack of a carefully thought out "front end" is thus a major flaw in the design of Hypertext. A person should be able to use it from any available port, whether at home, at school, in the office, at the library, or when walking in the park using Alan Kay's mythical Dynabook. (There's another fine-sounding idea that has never been specified to the point where you can evaluate if it's workable. It, too, lacks an interface specification and apparently does what the user intends by a combination of wishful thinking and magic.)

In short, Hypertext only sounds like a good idea. It tends to evaporate when looked at closely. There are three basic human interface problems: (i) the linkages are often either cumbersome, wrong for your needs, or trivial, (ii) the problem of what aspect of a word, phrase, or picture you intend has not been addressed, (iii) a uniform and excellent human interface specification is both necessary and absent.

I am sure enough of human ability to believe that the problems being addressed by Hypertext-like systems are not impossible to solve. What I am questioning is whether this particular proposed part of a solution, namely linked text (with a very broad definition of "text") is the right one, or even a feasible one. As with playing "adventure" games, you will be trying to second guess the link builder's frame of mind rather than staying within the subject you are working with. Knowledge of the system will often be more valuable than understanding your field. As the students of a poor teacher learn to give not the right answers but those the teacher wants, Hypertext users will spend much of their time pleasing the system instead of themselves.

It is perhaps worthwhile asking why Nelson and the Xanadu people have looked at the aspects they have and ignored the ones mentioned here. I hope I may be excused for making an *ad hominem* observation: these people consider themselves hackers (in the best sense). They feel at home at the annual Hackers' Conference. They love technology, and like Joyce's Leopold Bloom, thrive on innards. They can discuss the minutiae of a particular system to all hours and, I fear, sometimes confuse their satisfaction in technical ingenuity and accomplishment with usability. The power of a linked data structure is formidable and amazing. My own thesis in computer science exploited such a structure. From speaking with Xanadu's implementors, and some people who hope to tap into its data, I have seen how they have been seduced. The potential is enormous, the attention given to implementation of searches into vast quantities of human knowledge brilliant, and a host of fascinating problems in computer science have been attacked, and some seem solved. One gets an honest feeling of accomplishment. But they have not addressed the question: when a person sits down at her system what will have to be done first? What will she see? What is the next step the user takes, and the next...? How will it look? How many keystrokes or mouse pushes or finger jabs will it take to find what is wanted? How long will it take? No, they speak to me of megabytes and optical disks instead of Susan and Jim and Marjorie and Bill. They speak of the great benefits that will accrue to individuals — that's one end; and the vast linked data base — that's the other end. But when asked how the ends are made to meet, just how the natural impedence between them is overcome, the designers become inspecific and waffle. How humans interact with systems is not their field and is not really close to their hearts. It is not the computer science they studied. It is a soft subject and making good interfaces (which they fervently believe should be done) will not win them the sincerest plaudits of their peers.

None of this denies their interest in doing good things for the world and for individual people. The "how" just hasn't been thought through. This may stem from Nelson's belief that "The starting point in designing a computer system must be the creation of the conceptual and psychological environment — the seeming of the system [a nice phrase, but not very informative] — what I and my associates call the 'virtuality.'"

And what everybody else calls the "concept." I agree that one must begin by deciding, as Nelson says, "how it ought to be." Unfortunately, Nelson's "how it ought to be" does not include specifics of how it works in detail. Whether the hypertext concept is useful or not is most decidedly a function of the user interface, just as computers are more or less useful depending on how they facilitate access to their power. Consider the relative difficulty of using a double-edged razor blade without a handle. It is just as sharp, but much harder to use. Nelson's avoidance of what he calls "front end" issues is a major failing in the quality of his vision, his priorities, and his understanding of what makes things work well for people.

Since creating the Macintosh project at Apple I've led the design of a number of products including the Canon Cat and SwyftWare. In the process, my associates and I obtained an excellent interface between people and text by using exactly what Nelson decried: a straightforward linear structure. It was acted upon not by links and list processing but by an extremely fast search (which can be considered a real-time volatile link). One major difference is that we were driven to this internal structure by starting with a particular and finely tuned human interface, and only then searching for technology to implement it. I do not agree with Nelson's statement that people naturally think in hierarchical structures of many layers. If that assumption falls, so does most of the Hypertext concept. Studies show that people prefer flat structures over deep ones, and if we do not accede to the way humans work then we are not likely to design workable products.

The Hypertext vision is worth a try. The demonstrations so far have been tantalizing. It is my guess that the reality will remain tantalizing, and will never fulfill the dreams of Hypertext's advocates — nor the dreams that they instilled in me, for that matter.

While Hypertext itself may or may not be a good idea, the vision of giving everybody access to vast reaches of human knowledge is a praiseworthy one. There are probably better and more realizable ways. Hypertext is certainly incomplete at present, both in terms of implementation (as its designers would agree) and in terms of concept (they might not agree). My intent has been to present a countervailing opinion to the vast majority of what has been written and to balance the present uncritical discussion (that is, adulation) of Hypertext.

Thanks to David Alzofon, Sam Bernstein, and Scott Kim for their constructive comments.

REFERENCES

Conklin, Jeff, "Hypertext: An Introduction and Survey", IEEE Computer, September 1987, 17ff.

Relationally Encoded Links and the Rhetoric of Hypertext

George P. Landow

Department of English and the Institute for Research in Information and Scholarship (IRIS)
Brown University, Providence, Rhode Island 02912

I. INTRODUCTION

More than two years' work on designing, writing, editing, and linking documents in *Context32* [Land86], the first course employing Intermedia developed at Brown University's IRIS (Institute for Research in Information and Scholarship), has provided valuable experience of hypertext and hypermedia systems. *Context32*, which contains more than a thousand text and graphic files joined by approximately 1300 links, appears the most ambitious implementation thus far of a full hypertext and hypermedia system intended for multiple users. Members of the development team at IRIS have previously described various aspects of Intermedia's object-oriented programming [Meyr86], general design [Meyr85, Yank87a, Yank87b], and educational goals [Land87]. This paper presents conclusions about what works best at each end of a hypertext path or linkway and proposes that, like other forms of discourse, hypertext requires systems of conjunctive and other relational devices.

II. THE NEED FOR A RHETORIC OF LINKING IN HYPERMEDIA

Designers of hypertext and hypermedia materials confront two related problems, the first of which is how to indicate the destination of links and the second, how to welcome the user on arrival at that destination. Drawing upon the analogy of travel, we can say that the first problem concerns *exit* or *departure* information and the second *arrival* or *entrance* information. In both cases the designer must decide what users need to know at each end of a hypertext link in order to make use of what they find there. The general issue here is one of interpretation—namely, how much interpretation must the designer-author attach (1) to link pathways and (2) to files at the end of links to permit them to function efficiently in a multi-user system?

At this relatively early stage in the history of hypertext systems, those involved, as one might expect, devote most attention to the simple fact of linking and to the effects upon discourse of such electronically linked text. Although hypertext clearly redefines some of the basic characteristics of page-bound printed discourse, such as the rigidly hierarchical distinction between a main text and its annotation in scholarly works, it still depends upon many of the same organizing principles that make page-bound discourse coherent and even pleasurable to read.

The experience of courseware developers and of student users with *Context32* clearly demonstrates that simply linking one text to another in some cases fails to achieve the

expected benefits of a hypertext system and even alienates the user. Drawing upon our experience with developing *Context32* at Brown, I would like to examine several examples of insufficiently encoded hypermedia materials and then, in later sections, to discuss various approaches to encoding or attaching necessary information. Since graphic images in hypermedia systems so clearly demonstrate the problems created by insufficient encoding, we shall examine them first and then later look at text files.

At an early stage in the development of *Context32*, some of my graduate-student assistants assumed that placing certain kinds of images that they deemed interesting at the ends of links would suffice to inform students in some undefined way. Portraits of writers and similar documentary or informational use of earlier graphic work, such as a nineteenth-century photograph of homeless boys (Figure 1), exemplify this kind of linked file. Ultimately, I removed most such files or modified them in ways that are described below, but some were left unchanged because I wished to learn how students would react to them (and some because we have not yet had the opportunity to make necessary modifications to the files). As it turned out, students found no educational value either in such links or in the files at their ends, and they resented the time required to call them up, inspect them, and put them away.

Why, then, were such kinds of linking educationally and informationally ineffective? In the first place, these graphic files confused the users, who could not quickly determine why such material had been included. Once confused, they resented the presence of the link. Before discussing several solutions to the problem of efficient handling of informative links in hypertext, let us note the basic assumptions underlying these reactions, the most important of which is that links represent useful, interesting, educationally significant relationships. Such assumptions, we must realize, do not derive from overestimations of hypertext but are intrinsic to such systems. In fact, because links and link-relations play such primary roles in hypertext (and hypermedia), they influence the content they convey and thus exemplify the McLuhanesque principle that the medium is the message [McLu62, McLu64]—or at least that every medium of communication itself communicates an identifiable bias or message. Sometimes the message takes the form of a negative bias created by the limitations of a specific technology. For example, using currently available hardware for the first implementation of *Context32*, we discovered that both mid-nineteenth-century woodblock illustrations and those that Beardsley created at the end of the century (Figure 2) appear clearly. Photographs and paintings with a wide range of tonality and those depending heavily upon color do not work as well. In fact they reproduce so poorly that we have had to omit most Romantic art—with the result that users of the system might conclude not that our technology has particular limitations that constitute a bias but that there is *no* Romantic art or that it has little relevance.

Hypertext as a medium also conveys its own positive bias or message, for hypertext's system of linked files conveys the strong impression that its links signify coherent, purposeful, and, above all, *useful* relationships. From which follows:

Rule 1: Hypertext links condition the user to expect purposeful, important relationships between linked materials. Such was the capacity of hypertext systems that I originally planned to draw upon when I began to work with software developers at IRIS, and from this capacity one can

deduce one of the principles embodied in *Context32* that is also a principle of hypertext:

Rule 2: The emphasis upon linking materials in hypertext stimulates and encourages habits of relational thinking in the user. Such intrinsic hypertext emphasis upon interconnectedness (or connectivity) provides a powerful means of teaching sophisticated critical thinking, particularly that which builds upon multi-causal analyses and relation of different kinds of data [Land87]. But one must note a third, cautionary principle:

Rule 3: Since hypertext systems predispose users to expect such significant relationships among files, those files that disappoint such expectations appear particularly incoherent and nonsignificant. When users follow links and encounter materials that do not appear to possess a significant relation to the file from which the link pathway originated, they feel confused and resentful.

As the examples of the author's portrait and documentary photograph suggest, appending brief texts in the form of titles to the images does not always provide enough information for the user, because titles do not sufficiently establish a relationship between the two linked files. Such inadequate relational encoding or markup does not appear only in hypertext and hypermedia systems, of course, but such systems accentuate the user's negative reaction. In fact, inadequately presented visual information characterizes many illustrated textbooks, particularly literary anthologies, that include portraits of authors, works of art, and other supposedly relevant visual materials. As studies have shown, students generally ignore such materials. Books permit the student user to avoid apparently nonsignificant and insignificant materials—one simply glances at them and turns the page or, in many cases, simply never glances at them at all—but hypermedia systems, whose linkages suggest that the user will encounter significant relationships between materials, make ignoring such materials more difficult. They force the user to confront relationality—or its absence.

III. SOLUTIONS: THE RHETORIC OF ARRIVAL

As the examples of a writer's portrait and documentary information about the historical background have shown, conventional titles do not adequately direct the user how to relate a graphic image to other materials. In contrast, "Victorian Design: Medieval Revival" [Figure 3], an illustration of an item exhibited at the 1851 Crystal Palace exhibition, the first world's fair, exemplifies the kind of encoding or rhetoric required to enable the user to make sense of graphic files—that is, to discern one or more conceptual relations between them and files to which they are linked. The appended text reads: "This curiosity from the Crystal Palace exhibition of 1851 is not a suit of armor but a stove built in the shape of one. What do such bizarre glances back at the past tell us about the Victorian age, which invented the idea of Progress as we know it? Can you find in the poems you have read any examples of thus clothing present purposes in ancient forms?" Students who highlight the link marker and issue a command to follow receive a menu of various Victorian literary works, including Tennyson's "Morte d'Arthur," a poem set in medieval England, and

"Tithonus," one set in the Greece of ancient myth. Experience with these materials has taught that to be educationally effective they must follow these principles:

Rule 4: Linked graphic materials must appear with appended texts that enable the user to establish a relation between file of departure and that of arrival. The solution we have adopted appears in Figure 3, whose text (a) provides factual information, (b) encourages users to relate that information to a problem on which they are working, and (3) contains links that allow them to pursue various investigations. From this follows two principles:

Rule 5: The entire text accompanying visual material serves as an introduction and not just the opening sentence or so. And:

Rule 6: The accompanying text does not have to specify all relevant information the designer wishes the user to have; rather, emphasizing that a relationship exists at all may be enough. From which follows:

Rule 7: Texts serve not only to provide information but also to reassure the user that the link embodies a significant relationship and to provide some hint, however incomplete, of how that relationship can be formulated by the user.

The visual information, which provides interesting data of an unexpected sort, also enforces the principle that relevance is in the mind of the beholder and that the investigator's function in whatever field is to inquire what connections might exist among various kinds of data and and their relative value might be evaluated.

The principles of making effective links to files containing largely visual information also apply when linking text to text. Here, too, one must employ devices that enforce hypertext capacity to establish intellectual relations. A user who activates the link marker in "Victorian Design: Medieval Revival" receives a choice of links to files about the poetry of Tennyson and Browning, both authors who, like the designer of the stove, use old forms to solve contemporary problems. The first of these files, "Tennyson's 'Morte d'Arthur'" (Figure 4), shows how one can emphasize the relationality of a text file by a combination of information, questions, and links. This text file begins by offering the student information about the poem's publication history and about its indebtedness to Sir Thomas Malory's Arthurian work, after which it continues with a series of alternating factual statements and questions that ask the student to apply the new information contained in them to draw broad conclusions about medievalism, the relation of the poem to Tennyson's own life, and its connection to Carlyle, a passage from whose work is included. The file also contains link markers indicating the existence of paths to files on the poet's biography and Carlyle. Planned additions include an essay on medievalism and links to it. The point here is that one must employ the same structure of information, questions, and link markers in both graphic and text files.

IV. SOLUTIONS: THE RHETORIC OF DEPARTURE

We have examined methods of encoding points of arrival before examining the encoding of link markers, which are points of departure, because the basic need for such encoding appears so clearly when users first confront files new to them. Experience with *Context32* has shown that links and link markers also require similar kinds of encoding to be used effectively, and this implementation of hypermedia employs at least six different forms, three of which are internal to the file containing the link marker and three external.

1. Internal:

- (a) Link marker apparently independent of accompanying text.
- (b) Link marker whose spatial proximity to text indicates probable nature of link destination.
- (c) Specific directions accompanying link marker.

2. External:

- (a) Link descriptions
- (b) Menu called up by link marker at site of multiple links.
- (c) Local map automatically generated by Intermedia.

An instance of 1.(a), a link marker that appears independently of accompanying text, occasionally appears in overview files, such as that for Tennyson (Figure 5). Overview files, which are graphic directories, play an important role in *Context32* since they simultaneously inform the user in a general way about the kinds of information available in relation to a particular subject and also enforce the point, a central theme of the course, that individual phenomena, such as an author (e.g., Tennyson, Pope), aesthetic category or periodization (e.g., Victorianism [Figure 6], Neoclassicism), or other topic (e.g., religion in England, Darwinism), relate to a range of subjects and approaches. An example of such naked or unaccompanied link markers 1.(a), which are relatively rare in *Context32*, appears in the center of the Tennyson OV (overview) file near the poet's name. Although this marker lies adjacent to a text specifying the main subject of this graphic directory file, that text does not suggest any link destination since the user is already within the Tennyson directory file. This marker simply represents the destination of another link (since linking creates link markers within each linked file) to an index file, and as such it represents something of an anomaly and should be replaced by form 1.(b). Each of the separate boxes or texts specifies to varying degrees what the user can expect to find on arrival at the link destination. The markers situated near the titles of individual works of Tennyson and the boxes labeled "Biography" and "[biographical] Timeline" clearly indicate their destinations. That labeled "Literary Relations" leads to another graphic directory file [Figure 7], which takes a standard form, and after using *Context32* to study a few authors or works, the student can expect to encounter this kind of graphic representation of literary relationships. Since more variety exists in relation to other materials organized by the overview file, the user does not know in advance what to expect from the label. Thus "Cultural Context," which here links to the Victorianism OV (Figure 6), also links to an essay on

"Tennyson and Victorianism," but authors less important in the course often do not have such additional linked documents. Similarly, there is an even wider variation in the number and nature of documents linked to "Religion" and "Science and Technology."

One approach to informing the user more about the nature of a link destination appears in those files, chiefly literary relations directories, that exemplify 1.(b) because the text their link markers accompany is so specific that it indicates the likely nature of the link destination. Such specifying texts also appear in primary text documents. For example, when a file on a modern author contains a link marker near "Freudianism," "World War I," or an aspect of literary technique, such as "Theme," or "Imagery," the user can expect to encounter an essay (or diagrammatic presentation of one) on these subjects at the end of a link. These observations on the way that *Context32* indicates link destinations in various kinds of graphic and text files lead to a crucial principle of hypermedia:

Rule 8. Any file in a hypermedia (or hypertext) system is a directory file. Although I have been writing as if *Context32* only employed those graphic files specifically entitled "OV" (for overview) as directories, users can and do rely on whatever file they find themselves in to organize linked information. From this characteristic of *Context32* follows a point at which hypermedia capacities converge with an important emphasis of contemporary literary theory:

Rule 9. Regardless of what kind of directories the authors and designers include in a hypermedia system, users can organize it according to their individual interests. Modern literary theorists argue that literary investigations should be organized not in the traditional manner according to authors or periods but according to individual texts [Scho85]. Hypermedia, which has strong individualistic and democratic potential, permits users to make any file (and any interest) the organizing principle of their investigations.

Now, to return to the problem of encoding link destinations within files. The most specific textual specification of a link destination occurs in 1.(c), when specific directions inform users of a link destination, usually by inviting them to make their way there. For example, the file entitled "Point of View in 'The Prussian Officer'" begins: "Point of view [link to definition]," and in the file entitled "Neoclassical Couplet" the following sentence appears next to an example of Pope's satire on women: "Follow to see a woman's view: Lady Mary Wortley Montague's couplets."

In addition to such internal means of indication link destinations from within the file, Intermedia provides three external ones—2.(a) link descriptions, 2.(b) menus of link descriptions, and 2.(c) local maps. After choosing a link marker, the user can put down a menu and select a link description rather than follow a link without further information (Figure 8). Many, though not all, link descriptions inform the reader if the linked files contain primarily graphic or text materials. During the first implementation of Intermedia and *Context32* users do not seem to have employed this capacity very much, perhaps because invoking it involves additional delay. A second external means of indicating link destinations involves automatically generated menus that appear when the command is

issued to follow a link marker to which two or more files are linked (Figure 9). A third external indicator takes the form of Intermedia's automatically generated local map (Figure 10), which contains the names of all files linked to the file currently active. Although local maps for files containing relatively few links are easy to use and helpful as a way of discovering link destinations, those for files with many links are difficult to use.

V. ACKNOWLEDGEMENTS

Intermedia is the culmination of two years of intense effort by a large team of developers led by Norman Meyrowitz, Scholar's Workstation Group Manager. I would especially like to thank him and Nicole Yankelovich, our Project Coordinator, for their continual resourcefulness, tireless effort, and unfailing good humor. I would also like to thank Helen DeAndrade, Tim Catlin, Page Elmore, Charlie Evett, Matt Evett, Ed Grossman, Nan Garret, Karen Smith, Tom Stambaugh, and Ken Utting for their contributions to the Intermedia system and David Cody, Glenn Everett, Suzanne Keen Morley, Kathryn Stockton, and Robert Sullivan for their contributions to *Context32*.

The work described in this paper was sponsored in part by a grant from the Annenberg/CPB Project and a joint study-contract with IBM.

VI. REFERENCES

- [Conk86] J. Conklin. "A Survey of Hypertext." MCC Technical report Number STP-356-86, October 23, 1986.
- [Garr86] L. Garrett and K. Smith. "Building a Timeline Editor from Prefab Parts: The Architecture of an Object-Oriented Application." *OOPSLA '86 Proceedings*. Portland, Oregon: 1986.
- [Land86] G. Landow, D. Cody, G. Everett, K. Stockton, and R. Sullivan. *Context32: A Web of English Literature*. Providence, Rhode Island: Institute for Research in Information and Scholarship: Brown University, 1986.
- [Land87] G. Landow. "Context32: Using Hypermedia to Teach Literature." *Proceedings of the 1987 IBM Academic Information Systems University AEP Conference*. Milford, Connecticut: IBM Academic Information Systems, 1987.
- [Lars86] J. Larson. "A Visual Approach to Browsing in a Database Environment." *IEEE Computer*, June 1986.
- [McLu62] M. McLuhan. *The Gutenberg Galaxy: The Making of Typographic Man*. Toronto: University of Toronto Press, 1962.
- [McLu64] M. McLuhan. *Understanding Media: The Extensions of Man*. New York: McGraw-Hill, 1964.
- [Meyr85] N. Meyrowitz. "The Intermedia System: Requirements." Providence, Rhode Island: Institute for Research in Information and Scholarship, Brown

University, September 1985.

- [Meyr86] N. Meyrowitz. "Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework." *OOPSLA '86 Proceedings*. Portland, Oregon, 1986.
- [Ong82] W. Ong. *Orality and Literacy: The Technologizing of the Word*. London: Methuen, 1982.
- [Scho85] R. Scholes. *Textual Power: Literary Theory and the Teaching of English*. New Haven: Yale University Press, 1985.
- [Yank85] N. Yankelovich, N. Meyrowitz, and A. van Dam. "Reading and Writing the Electronic Book." *IEEE Computer*, October 1985.
- [Yank87] N. Yankelovich, G. Landow, and D. Cody. "Creating Hypermedia Materials for English Literature Students." *SIGCUE OUTLOOK*, September 1987.
- [Yank87b] N. Yankelovich, B. Haan, and S. Drucker. "Connections in Context: The Intermedia System." Providence, Rhode Island: Institute for Research in Information and Scholarship, Brown University, 1987.

Homeless Boys



From Sansom's "Victorian Life in Photographs" (Thames and Hudson: London, 1974)

Figure 1. "Homeless Boys" (a file linked to Charles Dickens)

BEARDSLEY'S THREE STYLES: (GPL)

Sir Bedivere (left), a Tennysonian subject, shows the influence of Morris and "Withered Spring" (center) shows that of Blake. How has Beardsley changed his handling of space and the human figure in the last picture, and why is it more "decadent"?



"How Sir Bevidere cast the sword Excalibur into the water"



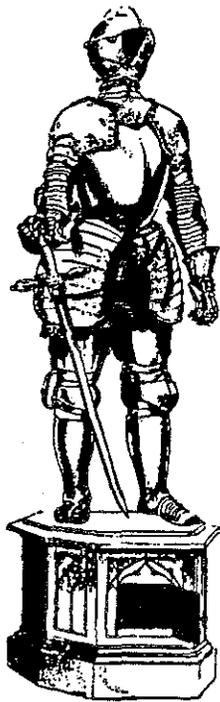
"Withered Spring"
(Beardsley)



"The Mysterious
Rose Garden"

Reade, "Aubrey Beardsley" (The Viking Press, New York 1967)

Figure 2. "Beardsley's Three Styles."



**Victorian Design:
Medieval Revival**

This curiosity from the Crystal Palace exhibition of 1851 is not a suit of armor but a stove built in the shape of one.

What do such bizarre glances back at the past tell us about the Victorian age, which invented the idea of Progress as we know it?

Can you find any examples of such clothing present purposes in ancient forms in the poems you have read?

[GPL]

"Morte d'Arthur"

[GPL]

First draft written early 1834; published in *Poems* (1842); incorporated into the *Idylls of the King* (1870) as "The Passing of Arthur." This is the first of Tennyson's poems to be based on Sir Thomas Malory's *Morte d'Arthur*. (He had written "The Lady of Shalott" in 1833, before he read Malory.)

1. Like "The Lady of Shalott," this poem represents one of Tennyson's early contributions to Medievalism in poetry. In what sense does "Morte d'Arthur" appear escapist and in what committed and immediately relevant to his own age? Does the poem suggest ways in which the modern poet living in an urban, technological, mercantile society can use myth or an idealized past?

2. Tennyson here employs a standard medieval romance literary structure that puts the protagonist through a series of tests that test and educate him. What in particular does Bedivere learn about the relation between keeping faith and being able to believe or have faith? What does this have to do with Carlyle?

3. Arthur the King is, at least in small part, also Arthur Henry Hallam (see the biography). What in this poem is amplified by your knowledge of Tennyson's personal grief for his friend?

4. Do you find a second debt to Carlyle in the connection between Arthur's benediction to Bedivere ("The old order changeth, giving place to new, / And God fulfills Himself in many ways, / Lest one good custom should corrupt the world" and these passages from Carlyle's *Signs of the Times*?

We have a faith in the imperishable dignity of man; in the high vocation to which, throughout his earthly history, he has been appointed. However it may be with individual nations, whatever melancholic speculators may assert, it seems a well-ascertained fact, that in all times . . . the happiness and greatness of mankind at large have been continually progressive. . . . That admiration of old nobleness, which now so often shows itself as a faint dilettantism, will one day become a generous emulation, and man may again be all that he has been, and more than he has been.

What similarities can you find between Carlyle's work and Tennyson's? [GE, GL]

Figure 3. "Victorian Design: Medieval Revival"

Figure 4. "Morte d'Arthur"

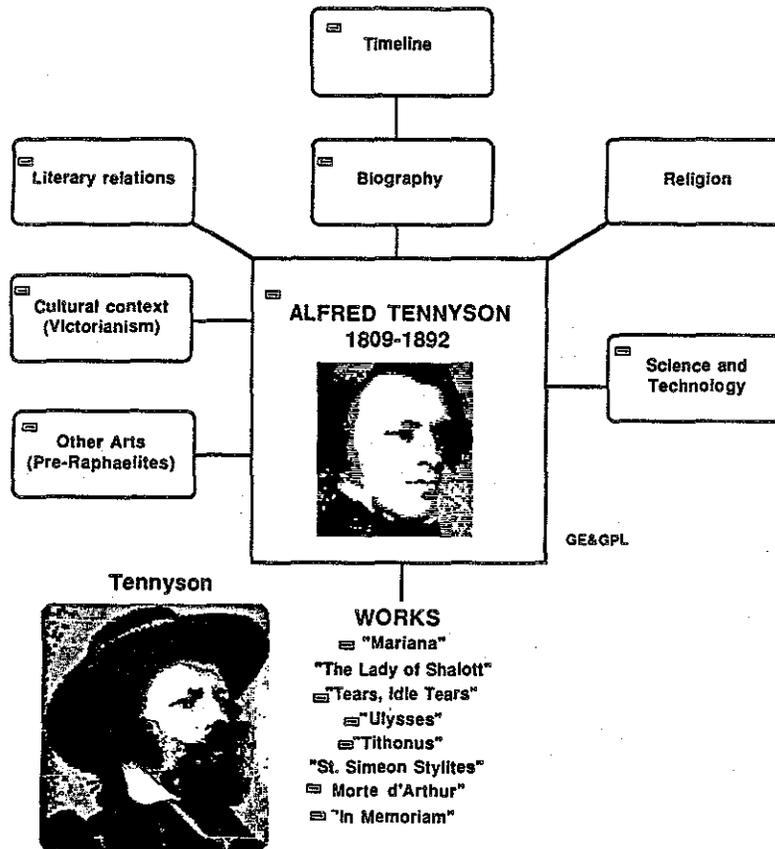


Figure 5. "Tennyson OV" (overview or directory file).

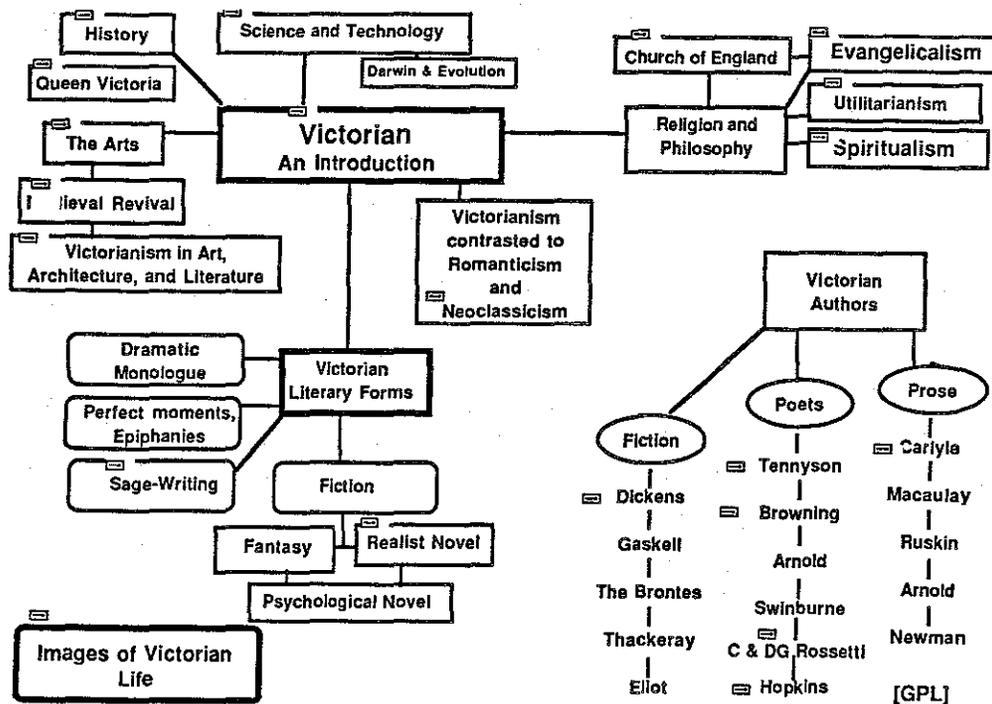
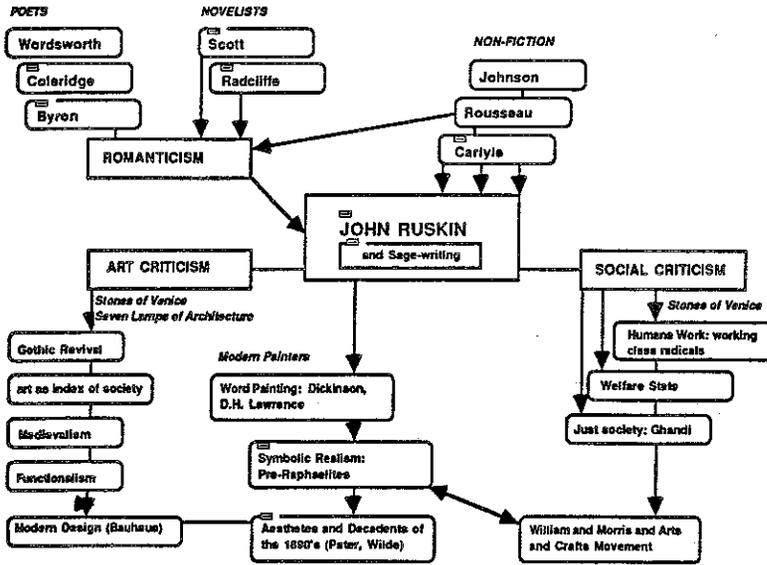


Figure 6. "Victorian OV"

RUSKIN'S LITERARY RELATIONS AND INFLUENCE



TENNYSON'S LITERARY RELATIONS

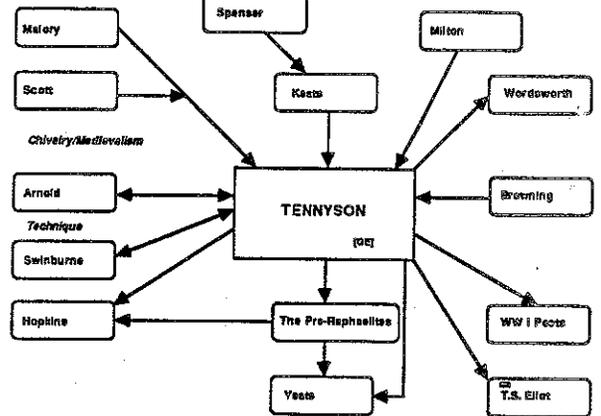


Figure 7. Two InterDraw files with varying degrees of relational text: "Tennyson's Literary Relations" and "Ruskin's Literary Relations"

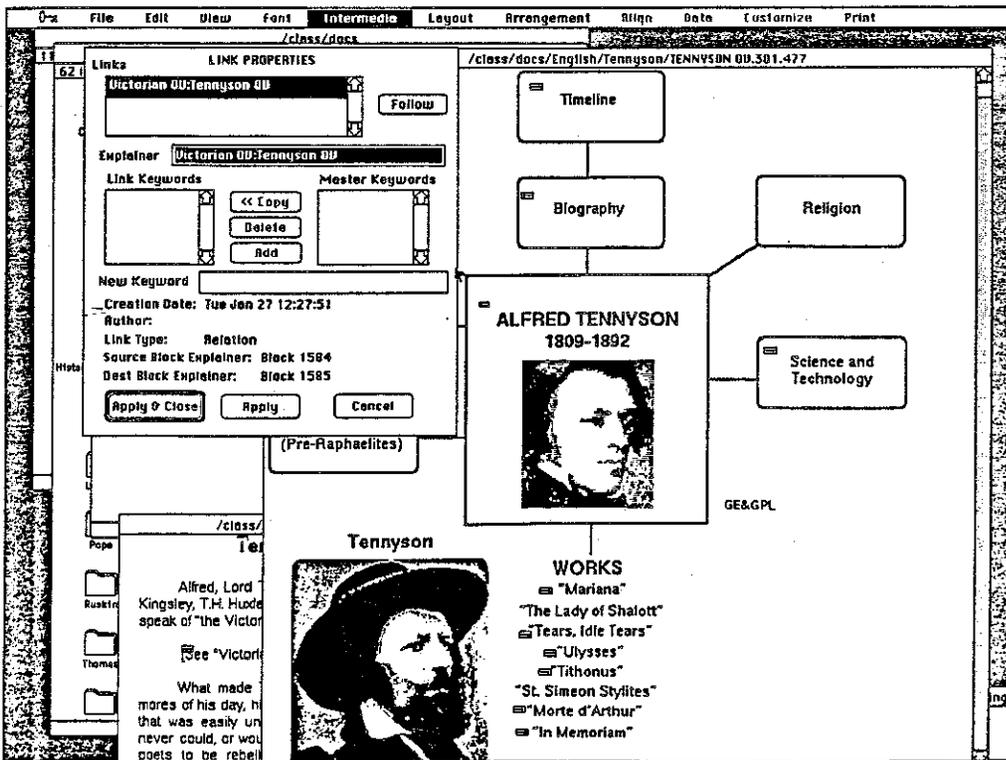


Figure 8. A link explainer (for path between "Tennyson OV" and "Victorian OV").

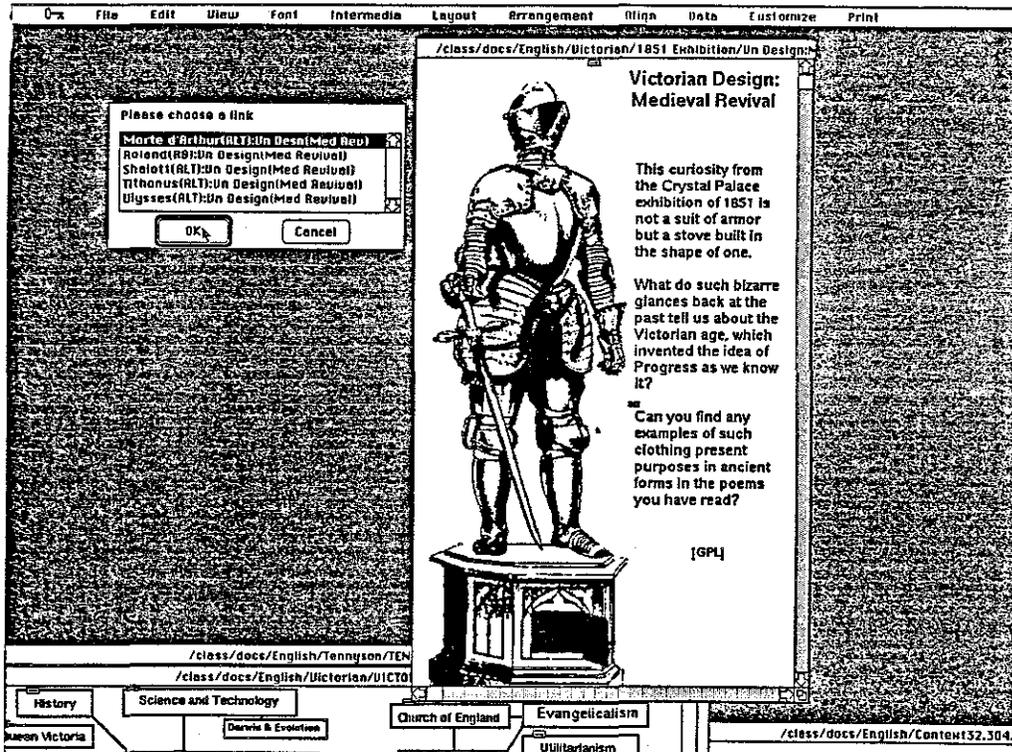


Figure 9. A menu of link descriptions (generated by a link in "Victorian Design: Medieval Revival").

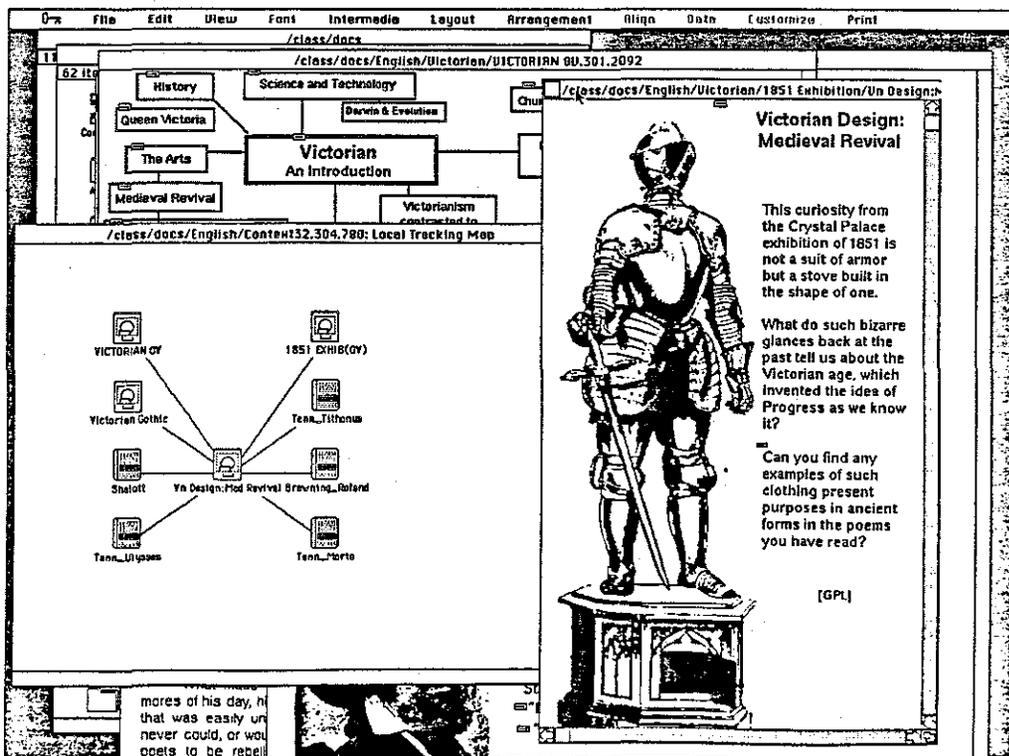


Figure 10. Local tracking map generated by links in "Victorian Design: Medieval Revival"

Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems

Frank G. Halasz*

Microelectronics and Computer Technology Corp. (MCC)
3500 West Balcones Center Dr.
Austin, TX 78759-6509

ABSTRACT

NoteCards is a general hypermedia environment designed to help people work with ideas. Its intended users are authors, designers, and other intellectual laborers engaged in analyzing information, designing artifacts, and generally processing ideas. The system provides these users with a variety of hypermedia-based tools for collecting, representing, managing, interrelating, and communicating ideas.

This paper presents the NoteCards system as a foil against which to explore some of the major limitations of the current generation of hypermedia systems. In doing so, this paper highlights seven of the major issues that must be addressed in the next generation of hypermedia systems. These seven issues are: search and query, composite nodes, virtual structures, computational engines, versioning, collaborative work, and tailorability. For each of these issues, the paper describes the limitations inherent in NoteCards and the prospects for doing improving the situation in future systems.

INTRODUCTION

NoteCards is a general hypermedia environment designed to help people work with ideas. Its intended users are authors, researchers, designers, and other intellectual laborers engaged in analyzing information, constructing models, formulating arguments, designing artifacts, and generally processing ideas. The system provides these users with a variety of hypermedia-based tools for collecting, representing, managing, interrelating, and communicating ideas.

NoteCards is in many ways typical of the generation of workstation-based hypermedia systems that is currently moving from the research lab into widespread use (e.g., Intermedia [Garr86b, Meyr86] and Neptune [Deli86a]). These systems are proving to be extremely useful in application domains ranging from educational courseware through computer-aided engineering to legal argumentation. At the same time, as these systems move out of the lab into the real-world, their limitations and design flaws are becoming increasingly apparent.

* I would like to thank Randy Trigg and Tom Moran of Xerox PARC and Jeff Conklin, Michael Begeman, and Eric Guillchsen of MCC. The ideas presented in this paper were developed during many discussions with these folks over the last year. I would like to especially thank Michael Begeman for pointing out the importance of what I've called virtual structures (and what he calls views).

This paper presents the NoteCards system as a foil against which to explore some of the major limitations of this current generation of hypermedia systems. In doing so, this paper will highlight some of the major issues that must be addressed in designing the next generation of hypermedia systems.

NOTECARDS IN BRIEF

NoteCards was designed to support the task of transforming a chaotic collection of unrelated ideas to an integrated, orderly interpretation of the ideas and their interconnections. Analyzing one's business competitors is a prototypical example. The task begins with the analyst extracting scraps of information about competitors from available sources. The collected information must be organized and filed away for subsequent use. More importantly, the information needs to be analyzed, i.e., the relationships between the various ideas have to be discovered and represented. Once these analyses are complete, the analyst composes and writes a document or presentation that communicates the discovered information and its significance.

NoteCards provides the user with a 'semantic' network of electronic notecards interconnected by typed links. This network serves as a medium in which the user can represent collections of related ideas. It also functions as a structure for organizing, storing, and retrieving information. The system provides the user with tools for displaying, modifying, manipulating, and navigating through this network. It also includes a set of methods and protocols for creating programs to manipulate the information in the network.

NoteCards was developed by Randall Trigg, Thomas Moran and the present author at Xerox PARC. A more detailed discussion of the system, its design goals, and our experiences with its use can be found in [Hala87].

Four basic constructs

NoteCards is implemented within the Xerox Lisp programming environment. The system is designed around two primitive constructs, *notecards* and *links*. In the basic system, these two primitive are augmented by two specialized types of cards, *browsers* and *fileboxes*, that help the user to manage large networks of cards and links.

Notecards. A notecard is an electronic generalization of the 3x5 paper notecard. Each notecard contains an arbitrary amount of some editable substance such as a piece of text, a structured drawing, or a bitmap image. Each card also has a title. On the screen, cards are displayed using standard Xerox Lisp windows as shown in Figure 1.

Every notecard can be 'edited', i.e., retrieved from the database and displayed on the screen in an editor window that provides the user with an opportunity to modify the card's substance. There are various types of notecards, differentiated (in part) by the nature of the substance (e.g., text or graphics) that they contain. In addition to a set of standard card types, NoteCards includes a facility for adding new card types, ranging from small modifications to existing card types (e.g., text-based forms) to cards based on entirely different substances (e.g., animation cards).

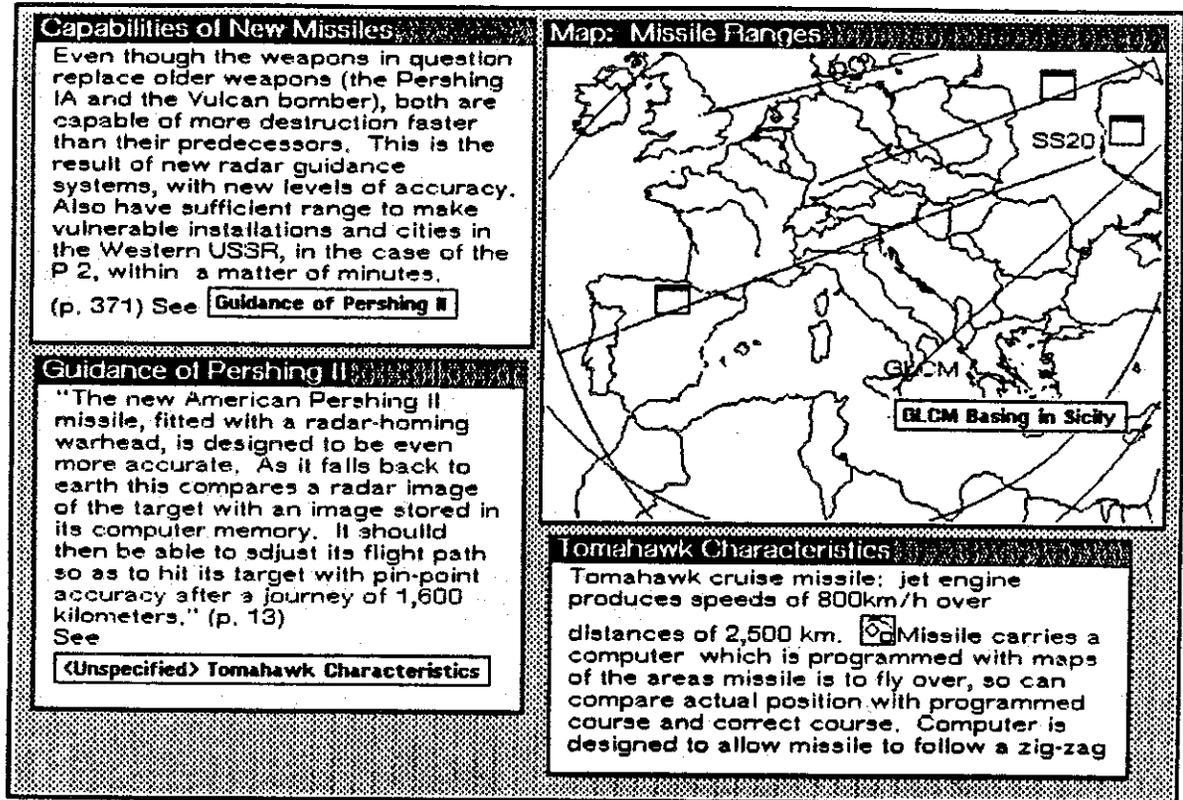


Figure 1: Example notecards with embedded link icons.

Links. Links are used to interconnect individual notecards into networks or structures of related cards. Each link is a typed, directional connection between a source card and a destination card. The type of a link is a user-chosen label specifying the nature of the relationship being represented.

The links are anchored at a particular location in the substance of their source card by a link icon but point to their destination card as a whole. Clicking in the link icon with the mouse traverses the link, i.e., retrieves the destination card and displays it on the screen. In Figure 1, each of the two cards contains two link icons.

Browsers. A browser is a notecard that contains a structural diagram of a network of notecards. Figure 2 shows a Browser card for a network composed of 8 cards and 8 links. The cards from this network are represented in the browser by their title displayed in a box. The links in the network are represented by edges between the boxed titles. Different dashing styles distinguish different types of links.

The diagrams in Browser cards are computed for the user by the system. Once created, browsers function like standard notecards. The boxed titles in the browser are in fact icons for traversable links between the browser and the referenced card.

Browsers support two levels of editing. First, the user can edit the underlying structure of a network of notecards by carrying out operations on the nodes and edges in the browser. Second,

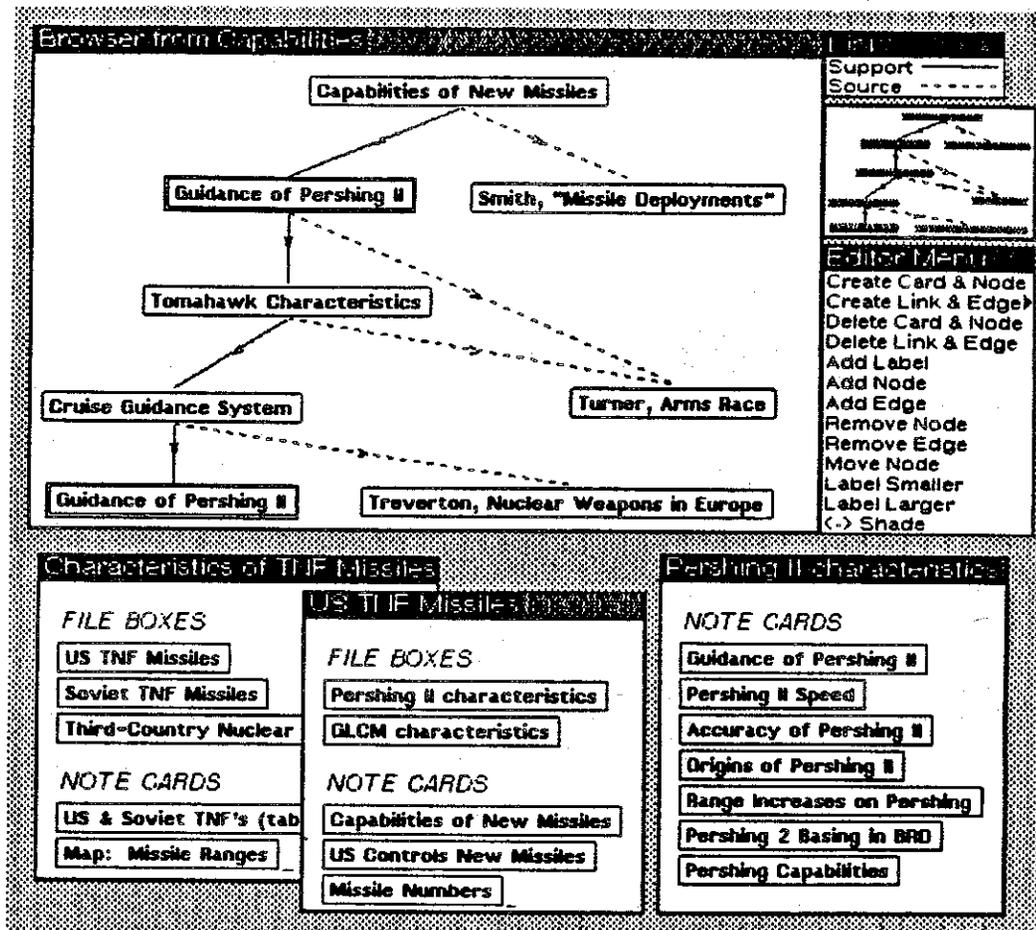


Figure 2: Example browser card (top) and filebox cards.

the user can add and delete nodes and edges in the browser diagram without making corresponding changes to the underlying NoteCards structures.

Fileboxes. Fileboxes are specialized cards that can be used to organize or categorize large collections of notecards. A filebox is a card in which other cards, including other fileboxes, can be filed. NoteCards requires that every notecard (including fileboxes) must be filed in one or more fileboxes. Figure 2 shows 3 fileboxes in addition to the browser.

Fileboxes were designed to help users manage large networks of interlinked notecards by encouraging them to use an additional hierarchical category structure for storage and retrieval purposes.

Interacting with NoteCards

Accessing information. Navigation is the primary means for accessing information in NoteCards. The user moves through the network by following links from card to card. Alternatively, the user can create an overview Browser for some sub-network and traverse the links from the Browser to the referenced cards. NoteCards also provides a limited search facility that can locate all cards matching some user-supplied specification (e.g., a particular string in the card's title or text).

User interface. The NoteCards user interface is mouse- and menu-based. Operations are initiated either by direct manipulation or by choosing commands from menus associated with the various icons and windows on the screen. Whenever possible specification of objects is done by the user pointing to the referent on the screen. Beyond these generalities, the user interface incorporates a diversity of specific interaction styles due to the great variation in user interface styles among the many preexisting Lisp packages incorporated into the system.

Tailorability

NoteCards is fully integrated into the Xerox Lisp programming environment. It includes a widely used programmer's interface with over 100 Lisp functions that allow the user to create new types of cards, develop programs that monitor or process a network, integrate Lisp programs (e.g., an animation editor) into the NoteCards environment, and/or integrate NoteCards into another Lisp-based environment (e.g., an expert system).

There is some degree of (non-programming) user tailorability in NoteCards as well. The system includes a large set of parameters that users can set to tune the exact behavior of the system (e.g., how links are displayed or the default size of notecards). In addition, users often create template cards or structures of cards that can be copied to create instances of the template. See [Trig87] for further details about tailorability in NoteCards.

NoteCards in use

From its inception, the design and development of NoteCards has been driven by the needs of its user community. Currently there are over 70 'registered' users within Xerox. At least 25 of these are 'everyday' users. There are, in addition, an undetermined number of users at various university, government, and industrial sites outside Xerox. This user community has provided invaluable feedback on the strengths and weaknesses of NoteCards as applied to a variety of tasks including document authoring, legal, scientific, and public-policy argumentation, design and development of instructional materials, design and analysis of copier parts, and competitive market analysis. Perhaps the most common use of NoteCards is a database for storing miscellaneous personal information.

A typical example of how the system can be used to support idea structuring and generic authoring is the network created by a history graduate student who used the system to research and write a 25-page paper. Figure 3 shows a browser of the filebox hierarchy created during this project. The author made a habit of keeping this browser on his screen at all times as a way of speeding up the process of filing and accessing cards. This hierarchy was made up of 40 fileboxes and contained 268 (non-filebox) cards.

The cards in Figure 1 are taken from this hierarchy. In general, cards stored in the hierarchy contain a short (average of about 100 words) quote or paraphrase taken from an article or book. About half of the cards have links embedded in their substance. As a rule, these were 'See' or 'Unspecified' links and were placed at the end of the card's text preceded by the word 'See'. There are also a few dozen inter-card links of other types.

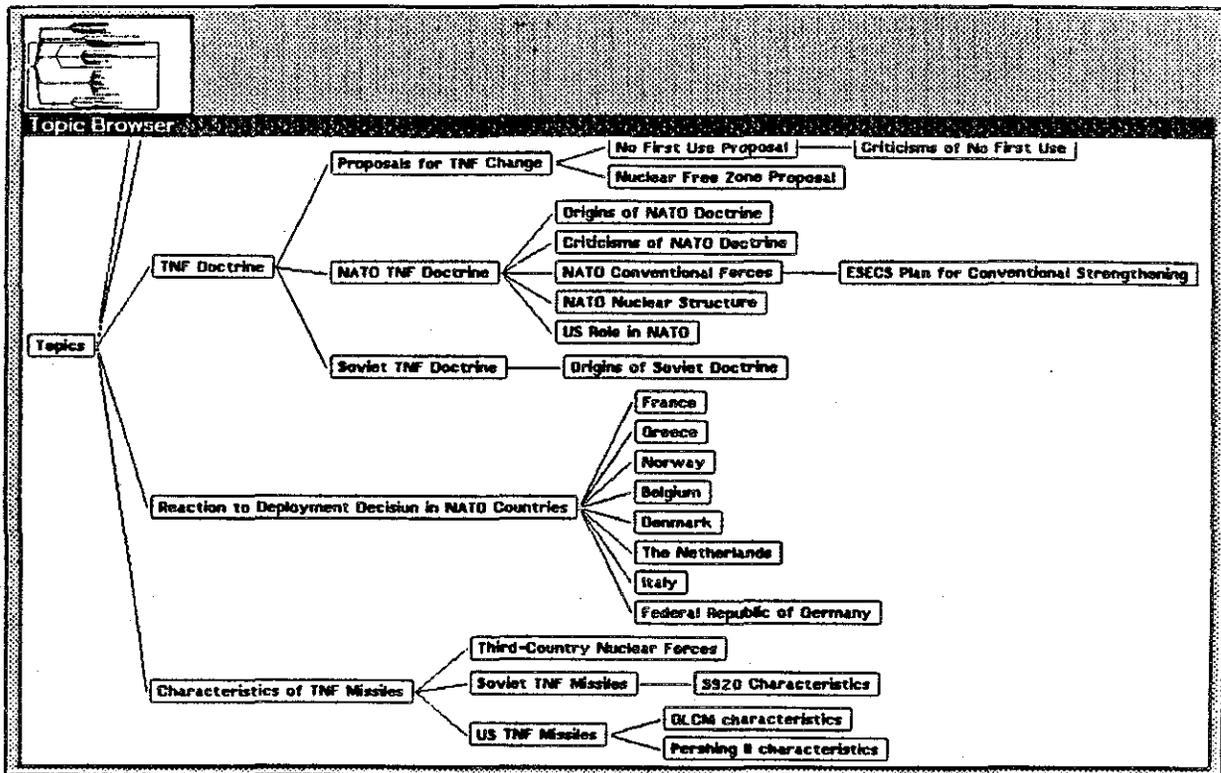


Figure 3: Browser of the filebox hierarchy from the public policy (nato-missiles) notefile.

Further description of this network and the process involved in using to author a paper can be found in [Mont86a] and [Hala87].

NoteCards' niche in the space of hypermedia systems

The term 'hypermedia' has been used to describe a wide variety of different systems and concepts ranging from outline processors through artificial video worlds. In this paper, 'hypermedia' will be used more narrowly to refer to information representation and management systems that organize information into networks of multi-media nodes interconnected by links. Each node generally contains a large chunk of 'content' such as a document, a drawing, or a voice annotation. The links are used to represent interrelations among these nodes.

Even with this restricted definition, there is wide variety in the nature and function of existing systems. NoteCards' niche in this varied space of hypermedia systems can be clarified by partitioning the space along three fundamental dimensions: *scope*, *browsing vs. authoring*, and *target task domain*.

Scope: Hypermedia has been proposed as *the* mechanism for storing and distributing the world's entire literary output [Nels81]. Hypermedia has been proposed as a common information space for teams of programmer's on large software projects [Garg87]. Hypermedia has been proposed as a tool for individuals and small work groups engaged in authoring and idea processing [Hala87]. Although all of these proposals share the notion that information should be organized

into network of nodes and links, they differ radically in *scale*, e.g., in the sizes of their expected information bases and user populations. This extreme variation in scale implies there will be differences throughout these systems, ranging from underlying storage mechanisms through the user interface to conventions for their use.

Browsing versus Authoring: In systems designed primarily for browsing, the hypermedia network is carefully created by a relatively small number of specialized authors in order to provide an information space to be explored by a large number of more or less casual users. These browsing systems are generally characterized by relatively well-developed tools for information presentation and exploratory browsing. Tools for creating and modifying the network tend to be less evolved. Hypermedia instructional delivery systems [Garr86b] and interactive museum exhibits [Shne87].

In systems designed primarily for authoring, the hypermedia network serves as a information structure that users create and continuously modify as part of their ongoing task activities. Hypermedia systems for idea processing [Hala87], document authoring [Smit86], and software development [Deli86a] are primary examples. In such systems, the tools for creation and modification of the network are well-developed. Tools for easy browsing and sophisticated information display tend to be less evolved.

Target task domain: Many hypermedia systems have been designed to support a specific task. For example, WE [Smi86] is an environment designed specifically to support the professional writer. Other hypermedia systems are designed to provide general hypermedia facilities to be used in a variety of applications. Even these generic systems, however, are usually designed with a target task domain in mind. The features and capabilities emphasized in the system often reflect the requirements of this target. Contrast, for example, Intermedia [Garr86b] and Neptune [Deli86a]. Both are general hypermedia systems. But Neptune was designed to support software engineering and thus emphasizes versioning [Deli86b] and node/link attributes. In contrast, Intermedia was designed for multi-user interactive educational applications and thus emphasizes novel interactive displays [Garr86a] and 'annotation' facilities.

NoteCards is designed for use by individuals or small work groups. In this respect, NoteCards is similar to PC-based systems like Guide [Guid86] and to single-user workstation systems like WE [Smit86]. Conversely, NoteCards differs significantly along this dimension from 'global' systems like Xanadu [Nels81], as well as systems designed to support larger groups such as ZOG [Robe81] and NLS/Augment [Engl68]. Although its original design places less-emphasis on multi-user access, NoteCards is very similar in scope to Neptune [Deli86a], Intermedia [Meyr86,Garr86b] and Textnet [Trig83].

NoteCards is first and foremost an authoring system designed to provide its users with facilities for creating and modifying hypermedia structures. In this respect, NoteCards is similar to many of the aforementioned systems (i.e., Augment, Guide, Intermedia, Neptune, WE) and different from online presentation systems such as TIES [Shne86] and interfaces to CD-ROM databases [Lamb86]. ZOG has a slightly different flavor, being simultaneously browser- and author-oriented.

NoteCards was originally designed as a tool to support idea processing and authoring in a research environment. Its original goals were very similar to those of NLS/Augment, although the actual implementations of the two systems are (on the surface) quite different. Systems such as WE and Guide were also designed to support authoring tasks. Intermedia (education) and Neptune (software engineering) were designed with very different application domains in mind, although both systems were designed in part to support document authoring tasks.

Overall, NoteCards is most similar to Intermedia and Neptune despite the differences in their target application domains. The three are very similar in scope and in the type of facilities they provide. This similarity is reinforced by several factors including a common emphasis on extensibility, similar underlying platforms (i.e., workstations), and a contemporaneous development schedule. Although the present paper specifically discusses NoteCards, most of the issues raised are equally relevant to these two systems as well as to most of the current generation of hypermedia systems.

SEVEN ISSUES FOR THE NEXT GENERATION OF HYPERMEDIA SYSTEMS

In the three years since its first release, we have been able to observe NoteCards in use and misuse in a wide range of situations and applications. These observations have provided significant insight into the system's particular strengths as well as its weak points. A brief but balanced assessment of the system is contained in [Hala87]. In contrast, for expository reasons the present paper focuses only on NoteCards shortcomings, i.e., on the ways in which the system falls short in meeting the needs and preferences of its users.

Many of NoteCards' shortcomings are specific to its current implementation and could be corrected by limited redesign or reimplementing of the existing system. However, some of NoteCards' shortcomings reflect fundamental weaknesses in the hypermedia model on which the system is built. It is precisely these fundamental shortcomings and the mechanisms for their correction that will form the basis for designing the next generation of hypermedia systems beyond NoteCards, Neptune, Intermedia, and their cohorts. The following sections describe seven such limitations that have become evident through our observations of the NoteCards user community. Each of these limitations raises a variety of issues for the design of future hypermedia systems.

Issue 1: Search and query in a hypermedia network

The primary method for accessing information in NoteCards is 'navigation' through the network by following links from card to card. Typically, the NoteCards user brings a card onto the screen, examines its content and links, and then traverses the link that is most likely to move closer to the target information. Fileboxes support such localized link traversal by providing a hierarchical structure in which information is located by recursive descent through an increasingly specific category structure. In addition, localized link following is augmented by browsers (and other user-invented overview displays) that provide global maps of the network. Such global overviews allow the user to visually scan for and then directly move to areas of the network in which the target information is likely to be found.

Navigational access to information has been adequate, and occasionally even ideal, for a large number of NoteCards applications. These applications can be divided into three basic classes. First, the navigational access has proven sufficient for the small authoring, note taking, and informal knowledge representation tasks that NoteCards was originally designed to support. In these tasks, an individual or a small (2 to 3 person) workgroup is creating and intensively using a relatively small network (50 to 250 cards). Because the network is small and familiar, users have little problem locating information.

A second class of 'navigational' applications are the display-oriented representation tasks in which the network is centered around a single display, usually a browser, used to represent a structure being designed or studied. The goal of these tasks is to create and manipulate this display. In some sense, the network is secondary to the display and is used only to create the structure to be displayed and to "hide" unimportant details. In these tasks, information access occurs through the central display, with little direct card-to-card navigation. An example of such a network is described in [VanL85].

The third class of navigationally-oriented applications is online interactive presentations. In these applications, the network's author often includes in each card navigational instructions to be used by readers of the network. Such 'guided' online presentations are discussed in [Trig86]. If no such navigational instructions are included, then the network is generally designed to be explored by the user in a non-directed manner.

In contrast to these navigationally-oriented applications, there are a variety of applications for which NoteCards' reliance on navigational access is problematic. These applications are generally characterized by large, unfamiliar, heterogeneously structured networks. Even in a 500 node single-user network, navigational access can be difficult as the network changes and its structure becomes heterogeneous. In these cases, navigational access is problematic because users tend to get lost while 'wandering around' in the network looking for some target information. Often these users can describe exactly what information they are looking for, but simply cannot find it in the network.

An incremental solution to the navigational problems encountered in NoteCards would be to improve and augment the existing navigation tools. For example, browsers could be made significantly more effective by applying techniques such as fish-eye views [Furn86] and graph 'fly-overs' [Fair87]. In addition, new tools such as a voting scheme similar to Synview [Lowe86] could be implemented in NoteCards. Although these changes would alleviate some of the navigation problems, they would not eliminate them entirely.

A more fundamental solution to the navigation problem is to augment navigation by a query-based access mechanism. With query-based access, the user could formulate a query encapsulating a description of the target information and then rely on the system to locate that information in the network. NoteCards already provides some limited query/search facilities. In particular, a user may search for all of the cards with some distinct property, e.g., that contain a particular text string. This search is extremely simple with no boolean expressions and no regular expressions provided. For NoteCards to be a useful in managing large heterogeneous networks, search and

query needs to be elevated to a primary access mechanism on par with navigation. Providing a search and query mechanism in the context of a hypermedia system is an interesting challenge.

There are two broad classes of query/search mechanisms needed in a hypermedia system. The first mechanism is *content search*. In content search, all cards and links in the network are considered as independent entities and are examined individually for those entities whose content or properties match a given query. For example, *all the cards containing the string 'hyper* system'* would be a content query. Content search is more or less standard information retrieval applied to a hypermedia information base. Although many techniques for such searches are well known, there are many innovative approaches that could be explored in a hypermedia environment.

Content search basically ignores the structure of a hypermedia network. In contrast, *structure search* examines the hypermedia network for sub-networks that match a given pattern. For example, a simple structure query might be: *all sub-networks containing two cards connected by a 'supports' links where the target card contains the word 'hypertext'*. This query contains a description of node content (i.e., *contains the word 'hypertext'*). It also contains a structural description of a sub-network (i.e., *two cards connected by a 'supports' link*). A more complicated structure query, involving an indefinite sequence of links, might be something like: *a circular structure containing a card that is indirectly linked to itself via an unbroken sequence of 'supports' links*. This query could be used, for example, to find circular arguments.

The development of a structural query facility is an interesting and difficult task. One major subtask is to design a query language geared towards describing hypermedia networks. This structure query language needs to be useable by the typical hypermedia user, who is unlikely to be facile with the intricacies of structure representation languages. A second major challenge is to design and implement an efficient structure search engine to process these queries. A mechanism that searches a network for any sub-network matching some (possibly complex) pattern is difficult to implement and even more difficult to implement efficiently.

In addition to its role as a mechanism for locating information, search and query be critical to many other aspects of future hypermedia systems. For example, the search and query mechanism will be used as a filtering mechanism in the hypermedia interface. Users will specify a 'query' in order to describe the information of interest to them. The interface would then show only those aspects of the network that 'matched' this query. The NoteCards browser currently operates in this manner, but only with respect to a very limited set of structure queries. A full-blown query mechanism would allow much more interesting browsers to be constructed. More importantly, the search/query mechanism could be linked much deeper into the interface providing for a pervasive information filtering mechanism that is absent in NoteCards and its cohorts.

Search and query will also be a critical component underlying the virtual structures mechanism described in Issue 3 below. Thus, in many ways, the success of the next generation of systems will depend on a good solution for the problem of search and query in a hypermedia network.

Issue 2: Composites — augmenting the basic node and link model

There are only two primitive constructs in NoteCards, cards and links. All other mechanisms in the system, including fileboxes and browsers, are built out of these two constructs. Although, this design has been surprisingly successful, experience suggests that this basic hypermedia model is insufficient. In particular, the basic model lacks a composition mechanism, i.e., a way of representing and dealing with sets (or sub-networks) of nodes and links as unique entities separate from their components.

A typical use for composites can be seen in the task of writing an organized document (e.g., a technical report) in NoteCards (see [Trig87]). In this task, users typically put the text for each subsection and for each figure into a separate card. All of the cards for a single section are then filed in a filebox. These section fileboxes are filed in the appropriate chapter fileboxes, which in turn are filed in a single filebox representing the document. This scheme is workable. It allows the user to focus in on the text/graphics for a particular chapter, section, or subsection. Using the NoteCards document compiler, the user can linearize the network into a single document card containing all of the text and graphics for the document in the appropriate order but without any hierarchical structure. This document can then be manipulated, e.g., read or printed, as a single entity. There is a problem, however, in that the document card is a separate entity from the 'source' cards stored in the document's filebox hierarchy. It contains only copies of the text/graphics from these source cards. Changes made to the text/graphics in the document card are not (automatically) reflected in the corresponding source card.

More importantly, the user can see the entire document at only one level. Despite the elaborate filebox hierarchy, there is no way to 'zoom' in and out of the document structure, examining its contents at different levels of detail (e.g., just the chapters, or all the chapters with their subsection headings, etc). This capability is commonly found in outline processors and is a critical component in many writing and information organization tasks. As a result, a number of writers have abandoned NoteCards in favor of outline processors for their simple authoring tasks.

This example suggests that NoteCards is missing the critical notion of composition. In particular, the system lacks a mechanism by which collections of cards and links can be reified as single composite entity. If such a mechanism were available, the document filebox hierarchy could be replaced by a composition hierarchy. In this latter hierarchy, the document would be a linearly ordered composite of chapter cards. Each of these chapter cards would themselves be a linearly ordered composite of section cards, which in turn would be a linearly ordered collection of the source cards containing (finally) text/graphics. Each composite card in this hierarchy could be displayed showing varying amounts of 'detail' about its (direct and indirect) subcomponent cards. For example, the highest level composite, i.e., the document itself, could be viewed as a list of chapters or, alternatively, as a concatenation of all of the text/graphics in the source cards at the base of the composition hierarchy.

The semantics of composites implies that the components of a composite are included 'by reference' rather than 'by value' as is currently done in NoteCards' document card. Thus changes to a source card would by definition be reflected in any composite that contains that card. Con-

versely, changes to text/graphics viewed from within the context of a composite would by definition be changes to the source card actually containing the text/graphics.

The use of a composition mechanism to augment the basic node and link mechanism is a critical advancement in the hypermedia model. The filebox concept in NoteCards was designed (in part) to provide some of the characteristics of a composition mechanism in the context of encouraging hierarchical organizational structures. But the filebox concept was flawed because it failed to take into account the differences between *reference* relations and *inclusion* relations. In particular, inclusion implies a part-whole relationship in which characteristics of and operations on the whole will be true of the part as well. Reference implies a much looser relationship in which the participating entities allude to each other but remain essentially independent.

The difference between (and the confusion among) inclusions and references arises very frequently in designing hypermedia networks and applications. In NoteCards, a browser in some sense 'contains' the sub-network it displays. For example, one can edit the network by editing the browser, which implies an inclusion relation. But confusion arises because the actual implementation uses standard links (i.e., references) to connect the browser to the nodes it 'contains'. The NoteCards filebox concept was a source of great confusion among users for similar reasons.

On a less system-oriented level, NoteCards users have frequently requested the ability to refer to sub-networks or collections of unlinked cards as unique entities with names and properties of their own, separate from the names and properties of their component nodes and links. For example, a legal application might involve building a network of arguments (one argument per card) for each case. Ideally, this entire network would be 'included' in composite card, which the user could then utilize to refer to the network in its entirety. For example, the user might want to attach properties to the network as a whole marking its validity or its convincingness. In the absence of a composition mechanism, the user must invent her own mechanism for dealing with structures made up of more than one card. In NoteCards, users frequently utilize the root card of a sub-network as the representative of the entire sub-network structure.

In the next generation of hypermedia systems, composition should join nodes and links as a primitive mechanism. NoteCards, Intermedia, and Neptune all currently lack a notion of composition. Designing a composition mechanism appropriate for inclusion in these systems raises a host of interesting decisions and issues including, for example:

Can a given node be included in more than one composite?

Do links necessarily refer to a node *per se* or can they refer to a node as it exists within the context of a given composite? If the latter is possible, what does it mean to traverse a link?

How does one handle versions of composite nodes? E.g., does a new version of an included node necessarily imply a new version of the composite?

Should composites be implemented using specialized links or is a whole new mechanism necessary?

These issues present a challenging design problem for future hypermedia systems. However, the task is not impossible. NLS/Augment [Engl68] has (of course) already pioneered much of the

territory. The notion of composition is used heavily and effectively in the NLS/Augment system, although in ways that are not always directly relevant to NoteCards and its cohorts.

Issue 3: Virtual structures for dealing with changing information

NoteCards requires its users to segment their ideas into individual 'nuggets' to be stored away, one per card. Each of these cards then needs to be assigned a title and filed in at least one filebox. Empirical observations [Mont86b] have shown that these three seemingly trivial tasks pose significant problems for many users. In particular, a user in the very early stages of working with a particular set of information may not sufficiently understand the content and structure of that information. Knowledge about the critical dimensions of the idea space, the characteristics which distinguish one idea from another, and appropriate naming schemes develops over time as the user becomes familiar with her information. The problem arises because the segmentation, titling, and filing tasks all require the user to have such knowledge 'up-front'. As the user's knowledge of the information space evolves, previous organizational commitments (e.g., titles and filing categories) become obsolete.

Experienced NoteCards users get around this problem by adopting various strategies to delay the segmentation, titling, and filing of information. To avoid premature segmentation, these users will place the entire idea stream in a single text card. They will go back and review the entire stream before segmenting into separate cards. To avoid premature filing, experienced users file all cards in a single filebox and then use a sketch card in order to organize the cards. In this sketch card, they organize links (used as representatives of the target cards) into piles based on some judgment of similarity or 'belongingness'. These piles can be easily shifted or rearranged when new information comes in. When the piles are stable, they can be transferred into a filebox structure.

In a sense, NoteCards encourages its users towards premature organization of their information. This occurs because users' conceptual structures have a tendency to change faster than their corresponding NoteCards structures. The result is that the NoteCards structures are often obsolete with respect to the user's current thinking. To some extent, this situation is unavoidable because it will always be easier, quicker, and less tedious to change one's internal conceptual structures than it will be to update the external representations of these conceptual structures.

The pressure towards premature organization also reflects limitations in the NoteCards user interface. Relaxation of the strict titling and filing requirements is an often requested NoteCards 'enhancement' that would certainly help minimize this pressure. Providing less-stringent organizational structures such as the similarity piles described above would also provide a more natural environment for some organizational tasks. Increasing the ease by which structures could be modified (e.g., improving browser-based editing) would make it easier for users to track their changing internal structures.

At a more fundamental level, however, users experience difficulty in coping with change because the basic hypermedia model in NoteCards is inherently fragmentary and static. By definition hypermedia imposes a structure in which information is encoded into a collection of more or less independent nodes interconnected into a static (although changeable) network. This encoding is static because the segmentation and linking is represented directly in the data structure. The

encoding can be changed only by altering the data structure, i.e., by altering one static encoding to make a new static encoding.

The static nature of hypermedia networks could be largely eliminated (when appropriate) by including in the hypermedia model a notion of *virtual or dynamically-determined structures*. In the current model, nodes and links are extensionally defined, i.e., nodes and links are defined by specifying the exact identity of their components. In contrast, virtual structures would be defined intensionally, i.e., by specifying a *description* of their components. The exact subcomponents of a virtual structure are determined by a query/search procedure whenever the structure is accessed or instantiated. For example, a possible virtual composite node might be defined by a specification of the form: *a sub-network containing all nodes created by someone other than me in the last 3 days*. Each time this composite was accessed, its structure and content would be recomputed.

The notion of virtual structures for hypermedia is a direct adaptation of the concept of views (a.k.a. virtual tables) in the world of relational database systems (e.g., [Date81]). In the relational database world, a view is a table constructed at instantiation time by applying a stored view definition to data explicitly stored in base (or non-view) tables. From a user's perspective, a view-based table is nearly identical to a base table. All the same operations apply, including updates. Although virtual structures in a hypermedia network would be significantly more complex than the views in a relational database, the same principle of non-differentiation at the interface should apply. Any operation possible on a base hypermedia entity should apply as well to virtual structures.

The notion of virtual structures in hypermedia would be possible only in a system that supported a substantial search/query mechanism over the hypermedia network. The definition of the components in virtual structures are in fact queries. Instantiating a virtual structure involves satisfying these queries and constructing a dynamic entity from the results. Although it is not a strict requirement, it would make sense if the 'query' language used for virtual structure descriptions was the same as the basic query language.

Virtual structures are a particularly powerful mechanism when combined with the notion of composites. A virtual composite allows the user to create nodes that are dynamically constructed at access time from other nodes, links, and composites that are stored in the network. Such virtual composites are true hypermedia entities and not simply a display of results from a query. Thus, the user can add links, properties or additional static descriptions to a virtual composite. Browsers, for example, could be implemented as virtual composites built from the results of a structure query.

The notion of virtual links in a hypermedia structure has already been explored in ZOG [Robe81]. ZOG includes a small set of navigational links that are constructed whenever a node is accessed and displayed. These links connect the displayed node to nodes that the user has recently visited, thereby allowing the user to move quickly back from whence she came. In future systems, the notion of virtual (or computed) links will extend to non-navigational situations as well. Rather than linking to a specific node, the user will have the capability to link, for example, to *the most recently created node containing the string 'hypertext'*.

Implementing virtual nodes, links, and composites will be a difficult problem for the next generation of hypermedia systems, especially when response time is an important factor. Nevertheless, virtual structures will provide these systems with an ability to adapt to changing information in a way that is simply not possible with the current static hypermedia model. Although virtual structures will not replace static structures (since not every relation can be described by a query), they will be critical components of future hypermedia networks.

Issue 4: Computation in (over) hypermedia networks

NoteCards is basically a passive storage and retrieval system. It provides tools for users to define, store, and manipulate a hypermedia network. In service of this goal, it does some processing of the network and the information it contains. For example, browsers involve computing the transitive closure defined by a root node and a set of links types. The system, however, does not *actively* direct the creation or modification of the network or the information contained therein. Unlike an expert system, for example, NoteCards does not include an inference engine that actively processes the network in order to derive new information.

Although the basic system is relatively passive, NoteCards is frequently augmented by more active computational engines for a particular application or task domain. In one case, for example, NoteCards was augmented to function as the delivery vehicle for computer-assisted instruction [Russ87]. In this case, the applications developers implemented a driver that retrieved from the network and interpreted special 'script' cards. These script cards orchestrated the display of other cards containing instructional and test material. Students were expected to answer the test material and their answers were stored in the appropriate cards in the network. The driver then used these answers together with the instructions in the script cards to determine what material to display next. In a more advanced version of this system, the driver was a rule-based system that examined a number of cards in the network, including the cards containing the student's previous answers, in order to determine the what material to present.

In the foregoing example there is a clear separation between NoteCards *per se* and the computational engine embodied in the driver. The computational engine is not integrated into NoteCards. Rather it serves to consume and produce cards and links through the programmer's interface in much the same sense that a (human) user of the system consumes and produces cards and links through the user interface. Aside from the programmatic access to information in the network, NoteCards contains no special support for computational engines of this sort.

NoteCards does provide some haphazard support for integrating event-driven computations into the system. In particular, a (programming) user can create new classes of nodes whose methods carry out some auxiliary computation whenever an instance of the class is created, accessed, or modified. An example is the Query card, which performs a query on an external database to refresh its contents whenever it is brought up for display. While this ability to embed auxiliary computations in the methods of a card class is useful, it is limited in a number of ways. First, since links are not first class objects in NoteCards, it is not possible to trigger computations during the access or modification of links. Second, triggers are defined at the class level with no general support for triggering computations attached to specific node instances. Third, computation trig-

gered inside an access method cannot generally access the full functionality of NoteCards. Because of these limitations, relatively few users have implemented event-driven computations within NoteCards.

It is unclear whether the level support that NoteCards provides for computational engines is appropriate for future hypermedia systems. Designers of such systems could follow the NoteCards model and continue to support computational engines as separable external entities that create, access, and modify information via the standard programmatic interface. Alternatively, one could design a hypermedia system incorporating a more active computational component that automatically processes (e.g., make inferences from) the information stored in the network. In this case, the hypermedia system would function more like an expert system, both storing information and actively processing that information.

To a great extent the choice between a passive and an active hypermedia system is an efficiency versus generality trade-off. The ultimate functionality for the approaches is approximately the same. However, the distribution of responsibility and effort is different. Computation built into the hypermedia system it is likely to be more efficient, especially when that computation involves extensive access to information in the network. In contrast, an external computational engine is less restricted because no commitment to a particular computational engine needs to be made when the system is implemented. Thus, the choice between an active and a passive hypermedia system will be determined largely by the intended applications and the performance needs of these applications.

Both the internal and external computation models will involve challenging design issues. For systems that support an external engine, the triggers and hooks provided for this engine will have to be carefully chosen. Hooks need to be provided at both the instance and the class levels for all entities in the network (i.e., nodes, links, compositions, storage compartments, etc). Triggers should also be provided for particular events (rather than entities) to make it easier to implement event-driven systems. In addition, the system might provide some basic data structures that make it easy for an external computational engine to store temporary state information associated with entities in the network. For example, a cycle-detection engine might need to store markers on nodes it has visited. Handling these markers inside the hypermedia system might result in efficiency gains.

The design issues for the internal computation model are significantly broader and less crisp. Most importantly, the nature of the hypermedia computational engine needs to be determined. One possibility is a rule-based system in which the rules specify patterns in the network (akin to structure queries) and produce new network structures or modify existing structures. The rule interpreter for such a system could run continuously and asynchronously or it could be triggered by storage and retrieval events occurring in the hypermedia network. Other computational engines are also possible (e.g., truth-maintenance engines). The choice of an appropriate engine depends on the intended application and on the degree to which the concepts inherent in the computational engine can be integrated with the concepts inherent to hypermedia systems.

Issue 5: Versioning

NoteCards has no versioning mechanism. Each card and link exists in only one version and is altered in place when modified. As a result, the range of applications that can be easily supported in NoteCards is severely limited. For example, NoteCards has never been used for maintaining software due, in part, to the overwhelming importance of versioning in configuration management. The lack of versioning has had a lesser impact on the authoring, argumentation, and idea processing tasks for which NoteCards was originally designed. Although users in these applications frequently request versioning support, they have been able to make significant progress in its absence.

NoteCards lags behind its cohort systems in the versioning arena. Both Neptune and Intermedia provide some versioning support. Neptune, for example, provides a time-based linear version thread for individual nodes and links. The system also provides a partitioning scheme called 'contexts' [Deli86b] that allows users to begin a independent version thread for a given set of nodes. Intermedia, like NoteCards, keeps only a single version of each node. Intermedia, does however, have a notion of alternative named versions of the 'web', i.e., the set of links that interconnect a collection of documents. This allows each user, project, or application to work with its own network structure over a common set of document nodes.

PIE [Gold84] included a more extensive versioning mechanism than either Neptune or Intermedia. In the PIE model, versioning in a hypermedia network occurs at two levels: the level of individual entities (nodes, links, composites) and the level of changes to the hypermedia network considered as a whole. At the lower level, each entity has its own version history. In PIE the version history was a linear thread, but in the general case it could be an arbitrary version graph. Although the issue was not specifically addressed in PIE, it is important to provide (virtual) entities corresponding to both specific versions of an entity and the differences (deltas) between successive versions. Both the versions and the deltas should be addressable within the system, i.e., be possible hits for a search. For example, in a software engineering context it should be possible to search for either *the version that implements Feature X* or *the set of changes that implements Feature X*. One possibility would be to treat the deltas between successive versions as special hypermedia nodes capable of being annotated and referenced.

Providing a branched version history for all entities in a hypermedia network raises some very difficult issues regarding the semantics of references between entities. In particular, a reference to an entity may refer to a specific version of that entity, the newest version of that entity, to the newest version of that entity along a specific branch of the version graph, or to the (latest) version of the entity that matches some particular description (i.e., query). Which of these reference types is supported is a decision that affects the entire hypermedia system, but it is an especially critical decision in the design of links and composites. Moreover, composites raise the related problem of propagating version changes from subcomponents to their composites. For example, a (significant) change in an individual software module implies the creation of a new version of the system of which that module is a part. In contrast, updating the spelling of a few words in a paragraph may not require the creation of a new version of the document(s) containing that paragraph.

Maintenance of a version tree for individual entities, however, is not sufficient support. In general, users will make coordinated changes to a number of entities in the network at one time. For example, a software developer may implement a new feature by making coordinated changes to a number of separate modules. The developer may then want to collect the resultant individual versions into a single 'version set' for future reference. This set would be a snapshot of the collection of entities at some particular point in time, what in the software domain is often called a 'release'.

An alternative to version sets, is the 'layer' mechanism used in PIE. A layer is a coordinated set of *changes* to one or more entities in the network. For example, all of the changes made to various modules in a software system could be collected into a single layer described as *the changes that implement Feature X*. The resulting layer could then be applied to the pre-change versions of the entities to get the post-change versions.

In a layer-based system, the primary issue is the mechanisms available for managing and for composing layers. In PIE, there were constructs called 'contexts' that were essentially sequences of layers. The hypermedia network for a given context was determined by applying the first layer to the base network and then applying the next layer to the result and so on through the sequence of layers in the context. New contexts could be created by mixing and matching the layers from other contexts and then producing a hypermedia network by successive applications of the layers in the context. However, not all such constructions made semantic sense. PIE aided the user in determining which contexts were sensible and which weren't.

One important application of the notion of contexts is to provide a collaborative system (see Issue 6) in which each user maintains a private context through which she interacts with the hypermedia network. Each user's context contains a base layer that is the public hypermedia network. On top of this base are one or more personal layers which alter the base network to provide a personalized view. At any time a user can make her view public by applying the layers in her view to the layers in the base system (and informing her collaborators to use this new base network in their contexts).

Although the PIE versioning scheme is not perfect, it is richer and more complete than those provided by most hypermedia systems. One of the design issues facing the next generation of systems is how to improve the level of versioning support in hypermedia networks. The PIE model seems like an appropriate place to start in resolving this issue.

Issue 6: Support for collaborative work

In its original design, NoteCards focused almost exclusively on supporting individuals working alone on idea processing and information management tasks. Collaboration was seen as occurring outside of the NoteCards environment, e.g., through face-to-face conversations, electronic mail, or hardcopy documents. In practice, however, this has rarely been the case. Most idea processing and information management tasks are inherently collaborative, with groups of varying from two to ten people working on a common topic or project.

The problems and prospects surrounding support for collaborative work in NoteCards have been discussed in detail by Trigg, Suchman, and Halasz [Trig86] and hence will not be described here. It is important to note, however, that support for collaborative work remains a critical issue for the next generation of hypermedia systems. Although many existing systems have provided for the mechanics of multi-user access to a hypermedia network, none has adequately addressed the issue of support for the social interactions involved collaboratively sharing a hypermedia network. The challenge for the next generation of hypermedia systems is to provide adequate support for these social processes. This support should include system-level features such as the automatic maintenance of change histories, the tracking of individual contributions to the network, and mechanisms for enforcing collaborative conventions. It should also include a 'rhetoric of hypermedia' that provides guidelines or conventions for creating hypermedia networks that will be useable or understandable by others who share the 'rhetoric'. Interestingly, the development of these conventions is a crucial issue that can only be solved by accumulated experience in using hypermedia systems in real-world tasks.

Issue 7: Extensibility and tailorability

NoteCards was designed to be an extensible system. The expectation was that users would tailor the generic functionality of NoteCards to better match the requirements of their application and their preferred interaction style. The major mechanism provided for this purpose was the NoteCards programmer's interface. In practice, the programmer's interface has been very successful. A number of tailored systems, both major and minor, have been built on top of NoteCards using this interface [Hala87]. Trigg, Moran, and Halasz [Trig87] contains a lengthy discussion of tailorability in NoteCards, including examples of its use, problems with its implementation, and prospects for its improvement. Therefore, no further discussion of these issues will be included here.

However, one of the issues raised by Trigg *et al.* represents a critical challenge for future hypermedia systems. NoteCards was designed to make tailorability available to the entire range of users from non-programmer's to experienced system builders. The goal was to build tailoring mechanisms having the characteristic that small changes to the system could be accomplished with a small amount of work by users without extensive knowledge of programming and the system's implementation. This design goal was simply not met. Tailoring NoteCards remains a non-trivial programming task that requires some degree of expertise. This appears to be the case for other hypermedia systems as well. The challenge, then, for the next generation of hypermedia systems is to discover how to enhance the 'small' tailorability of hypermedia systems, i.e., how to make it easy for the typical (non-programming) user to extend the system to match her task requirements or interaction style.

CONCLUDING REMARKS

Hypermedia has suddenly become quite popular. There is something alluring about the concept of navigating through an information network following links from node to node until you find something of interest. But as the current crop of hypermedia systems move into more widespread use, their limitations will become quite evident to serious users. The simple node and link model

is just not rich and complete enough to support the information representation, management, and presentation tasks that these users will want to accomplish using their hypermedia system. This has been the experience with NoteCards over the last three years and there is no reason to believe that other systems will fare significantly better. The seven issues presented in this paper are an attempt to move the hypermedia model beyond simple nodes and links to a model that will better suit the needs and preferences of these users. In this sense, these seven issues represent an agenda for the next generation of hypermedia systems.

REFERENCES

- [Date81] Date, C.J. *An Introduction to Database Systems Vol. 1.* Reading, MA:Addison-Wesley, 1981.
- [Deli86a] Delisle, N. & Schwartz, M. Neptune: a hypertext system for CAD applications. *Proceedings of ACM SIGMOD '86*, Washington, D.C., May 28-30, 1986, 132-142.
- [Deli86b] Delisle, N. & Schwartz, M. Contexts - a partitioning concept for hypertext. *Proceedings of the Conference on Computer-Supported Cooperative Work*, Austin, TX, December 3-5, 1986, 147-153.
- [Engl68] Englebart, D.C. & English, W. A research center for augmenting human intellect. *Proceedings of 1968 FJCC*, 33, Part 1, Montvale, N.J.:AFIPS Press, 1968, 395-410.
- [Fair87] Fairchild, K. F., Poltrock, S. E., & Furnas, G. W. SemNet: Three-dimensional graphic representations of large knowledge bases In R. Guindon (Ed.) *Cognitive Science and its Applications for Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, in press.
- [Furn86] Furnas, G.W. Generalized fish-eye views. *Proceedings of the 1986 ACM Conference of Human Factors in Computing Systems (CHI '86)*, Boston, MA, April 13-17, 1986, 16-23.
- [Garg87] Garg, P.K. & Scacchi, W. A hypertext system to manage software life cycle documents. Paper submitted to the 21st Hawaii International Conference on Systems, 1987.
- [Garr86a] Garrett, L. N. & Smith, K.E. Building a time-line editor from pre-fab parts: The architecture of an object-oriented application. *Proceedings of the Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '86)*, Portland, OR, September 29 - October 2, 1986, 202-213.
- [Garr86b] Garrett, L. N., Smith, K.E., & Myrowitz, N. Intermedia: Issues, strategies, and tactics in the design of a hypermedia document system. *Proceedings of the Conference on Computer-Supported Cooperative Work*, Austin, TX, December 3-5, 1986, 163-174.
- [Gold84] Goldstein, I & Bobrow, D A layered approach to software design. In D. Barstow, H. Shrobe, & E. Sandewall (Eds.) *Interactive Programming Environments*. McGraw-Hill: 1987, pp 387-413.
- [Guid86] Guide Users Manual. Owl International, Inc. , Bellevue, WA., 1986.

- [Hala87] Halasz, F.G., Moran, T.P., & Trigg, R.H. NoteCards in a Nutshell. *Proceedings of the 1987 ACM Conference of Human Factors in Computer Systems (CHI+GI '87)*, Toronto, Ontario, April 5-9, 1987, 45-52.
- [Lamb86] Lambert, S. & Ropiequet, S (Eds) *The New Papyrus*. Redmond, WA: Microsoft Press, 1986.
- [Lowe86] Lowe, D. SYNVIEW: The design of a system for cooperative structuring of information. *Proceedings of the Conference on Computer-Supported Cooperative Work*, Austin, TX, December 3-5, 1986, 376-386.
- [Meyr86] Meyrowitz, N. Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework. *Proceedings of the Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '86)*, Portland, OR, September 29 - October 2, 1986, 186-201.
- [Mont86a] Monty, M.L. & Moran, T.P. A longitudinal study of authoring using NoteCards. *ACM SIGCHI Bulletin*, Vol. 18 No. 2, October 1986, 59-60.
- [Mont86b] Monty, M.L. Temporal context and memory for notes stored in the computer. *ACM SIGCHI Bulletin*, Vol. 18 No. 2, October 1986, 50-51.
- [Nels81] Nelson, T.H., *Literary Machines*, T.H. Nelson, Swarthmore, PA., 1981.
- [Robe81] Robertson, G, McCracken, D., & Newell, A. The ZOG approach to man-machine communication. *Int. J. of Man-Machine Studies*, 14, 1981, 461-488.
- [Russ87] Russell, D.M., Moran, T.P., & Jordan, D.S. The instructional design environment. Xerox PARC working paper, April 1987.
- [Shne87] Shneiderman, B. *User interface design and evaluation for an electronic encyclopedia*. Technical report CS-TR-1819, Dept. of Computer Science, University of Maryland, College Park, MD, March, 1987.
- [Smit86] Smith, J.B., Weiss, S.F., Ferguson, G.J., Bolter, J.D., Lansman, M., & Bea, D.V. *WE: A writing environment for professionals*. Technical report 86-025, Dept. of Computer Science, University of North Carolina, Chapel Hill, N.C., August, 1986.
- [Trig83] Trigg, R. *A Network-based Approach to Text Handling for the Online Scientific Community*. PhD thesis, Dept. of Computer Science, University of Maryland, November, 1983.
- [Trig86] Trigg, R., Suchman, L. and Halasz, F. Supporting collaboration in NoteCards. *Proceedings of the Conference on Computer-Supported Cooperative Work*, Austin, TX, December 3-5, 1986, 147-153.
- [Trig87] Trigg, R.H., Moran, T.P., & Halasz, F.G. Tailorability in NoteCards. Paper to be presented at Interact '87, Stuttgart, West Germany, August, 1987.
- [VanL85] VanLehn, K *Theory reform caused by an argumentation tool*. Xerox Palo Alto Research Center Technical Report, ISL-11, 1985.

DEVELOPING AND DISTRIBUTING HYPERTEXT TOOLS: LEGAL INPUTS AND PARAMETERS

Henry W. (Hank) Jones, III, Esq.

Morris, Manning & Martin, Corporate/Technology Group
Suite 1600, East Tower, Atlanta Financial Center
3333 Peachtree Road, Atlanta, Georgia 30326
(404) 233-4220

ABSTRACT

To realize the promise of hypertext, researchers and developers must understand how their work is impacted by copyright, products liability, and other sets of legal rules. Certain key legal problems, and corresponding possible solutions, are analyzed.

I. INTRODUCTION

"First thing we do, let's kill all the lawyers". —*William Shakespeare*

The attention of most system designers, programmers, and computer scientists who are involved in research or product development activities relating to hypertext usually is focused on the functionality, human factors, commercialization, and other aspects of this contemplated new technology.

Many professionals in the computing and information environments approach the identification and handling of legal issues as a later-stage project activity. At first view, this assumption might seem well-founded: patent applications, copyright registrations, and distribution contracts occur only at the end of their activities, in the course of product finalization and commercialization.

This article suggests that if the promise of hypertext actually is to be realized and disseminated to users, then early-stage identification of legal issues and compliance with legal obligations will be necessary.

This article also identifies primary legal pitfalls and related solutions that researchers, companies, and consultants should recognize in order to increase the odds of success in their activities.

II. PROPRIETARY RIGHTS CONSIDERATIONS

"To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries".—1787—*U.S. Constitution, Art. I, sec. 8*

"[S]cientific discoveries and technological developments have made possible new forms of creative expression that never existed before. . . . Authors are continually finding new ways of expressing

themselves, but it is impossible to foresee the forms that these new expressive methods will take".—1976 - *House of Representatives Report 94-1706, p. 51 (legislative history of 1976 Copyright Act)*

Many researchers and programmers approach hypertext activities largely as an opportunity to provide a new tool for information management, education, and other purposes to computer users, researchers, and the public at large.

On the other hand, marketing executives, corporations, and other entities see hypertext as a new technology offering significant potential profits.

Regardless of one's goals and perspective—inventor or investor—recognition of the ownership rights that may exist in hypertext products, processes, and research is essential to understanding the environment in which research and product development activities are funded and managed.

A discussion of some of the key intellectual property law considerations impacting hypertext is set forth below.

A. Copyright Law Questions

1. Background. The concept of "copyright" as a protectable economic interest—that is, as property—is linked with the era of Gutenberg, when the results of an author's labors were tangible, and physically separate from other authors' creative works.

The 1909 U.S. Copyright Act retained the assumption of discreet creative works, designed and created by the initial author (or painter, composer, playwright, etc.), and thereafter experienced by the public in a manner determined by that initial creator.

The development of videodiscs, computer programs, on-line electronic databases, satellite broadcasting, and other technologies that did not exist in the early twentieth century led to a complete overhaul of American copyright law in the mid-1970's. The 1976 Copyright Act, which became effective on January 1, 1978, specifies new "operating procedures" which regulate rights and obligations in software, user manuals, data bases, and other creative expression.

The importance of copyright law to the development of hypertext tools is underscored by the fact that this portion of federal law regulates not merely the right to reproduce a work (e.g., to "copy"). In fact, a copyright is a bundle of five distinct, severable rights. The other four are the right to adapt or modify a work (i.e., to create a "derivative work"), and the rights to publicly distribute copies of a work, to display it (e.g., on a computer terminal), or to perform it.

The ongoing development of new information technologies has tested the design and "coding" of this mid-1970's legislation [Offi86]. As researchers and companies in software, electronic publishing, CD-ROM media [Jone87d], image scanners, desktop publishing [Divo87], and other information-related fields have learned, identifying and managing copyright considerations can be essential to reaping expected rewards for their efforts, and to avoiding disputes and disappointed expectations. [Jone87c].

Particularly important copyright issues in the hypertext field include:

2. Where Will "Look and Feel" Stop (or Where Does The Code Stop And The "Structure, System and Organization" Start)? Copyright law only protects the author's "expression", but not any idea, procedure, innovation, or system.

The five exclusive rights provided to a copyright owner under the 1976 Copyright Act do not necessarily constitute a broad assurance of a return on the investment of time, talent, and creativity by programmers and system designers. Rather, copyright law only precludes third parties from exercising one of the five

exclusive rights (i.e., reproducing, modifying, publicly distributing, publicly performing or publicly displaying a work) without the author's prior permission (unless the "fair use" defense is applicable; see below).

Developers of videogame software [Jones83], application software with special "human interface" characteristics, and other information technology developers have learned that the copyright statute is not a clear-cut arbiter of what innovations may be freely shared within society.

Recent litigation regarding the susceptibility to copyright protection of a software product's "look and feel", and of the pagination and abstracting attributes of the Westlaw print and on-line data systems, evidences the lack of a consensus regarding the proper scope of copyright protection. As reported by many industry, computer, and software trade magazines, a number of major corporations have filed copyright lawsuits against competitors who have mimicked the screen displays of their products. Some commentators have suggested that lawsuits filed by Lotus Development Corporation, Digital Communications Associates, Inc., West Publishing Company, Broderbund Software, Inc., CADAM, Inc., and other companies may impair the development of new technologies, new user interfaces, and new products, and may increase artificially the cost of user training.

Hypertext researchers should strive to avoid a "look and feel" dispute. Judges and juries face a heady "learning curve" when attempting not only to fit new information technologies within rules and categories established for older information media by prior legislative designs and court rulings, but also to understand the technology at issue. Moreover, a recent federal appellate court ruling, involving a competitive software product for dental laboratories written by the plaintiff's ex-employees in a different programming language for a different computer, has resulted in a ruling that the protected "expression" of a computer program includes not merely the actual code, but also the product's "structure, system and organization". [Jones & Dailey 86, 87a, and 87b].

Due to the rapid developments in this area of "computer law", product managers and attorneys must monitor the evolution of this issue (including progress in the Lotus lawsuits, and any results of the September 9-10, 1987 Copyright Office hearings on this subject).

Moreover, companies seeking to develop and protect what they believe to be proprietary interfaces must take active steps to identify and retain the proprietary aspects of their developments, including maintaining archival records of their creative efforts, using separate copyright notices for screens, and possibly seeking separate copyright registrations.

3. Where Does Hypertext "Fit"? The Copyright Act's definitions, current Copyright Office regulations, current copyright registration forms, and other elements of copyright protection are rule-based; they operate based on particular *categories* of creative expression.

"Computer program", "compilation" (e.g., a database), and "collective work" (e.g., a set of data bases, or a collection of application products) are defined terms, for which attorneys, businesspeople, and judges can evaluate, explain, and provide guidelines.

Hypertext developers must expect to undergo the same inconveniences as developers of firmware, compact discs, and other new information technologies have faced.

4. Can The User Be An "Author"? Both hypertext and expert systems may trigger the necessity to grapple with an old conundrum that "computer law" practitioners have identified for many years: whether the original programmer of a software tool, or the user of the product, is the "author", and hence the copyright owner, of the creative material generated by the use of that tool.

This issue initially was encountered by federal judges in the early 1980's, as litigation asserting infringement of copyrights in videogame software worked its way through the courts. The defendants in these cases often were accused of counterfeiting or "cloning" the ROM chips on which game-playing

software was stored. The defendants in this novel copyright litigation argued that the screen output in those games was "created" by the strategic decisions, hand movements, and creativity of the game player, rather than the companies and individuals who created the software. Fortunately for the companies developing and marketing such products, the courts uniformly held that the user was not an author for copyright law purposes; the parameters of the user's activities had been controlled by the initial design and coding by the plaintiff companies' programmers. [Jones83].

Given the increased sophistication, features, and flexibility of current and contemplated hypertext tools, and in view of a user's creative flexibility in selecting, culling, and merging sets of information through searches and other information processing activities, the reproduction of these earlier court results cannot be assured.

5. Will The "Fair Use" Doctrine Prevent The Stifling Of Research and Competition?

Despite the perhaps threatening idea of federal court litigation, Copyright Office filings, and the involvement of lawyers, the 1976 Copyright Act does contain one element that may minimize any possible impediment to progress.

The "fair use" rule is a "safe harbor" from liability for acts which otherwise would constitute copyright infringement. Noting the societal, cultural, and academic importance of many copyrightable works, the U.S. Congress, in drafting the 1976 Copyright Act, endorsed the body of judge-made (i.e., "common law") exceptions to liability that had been enunciated under the 1909 Copyright Act. The current copyright statute permits a judge to withhold liability, despite a company's or individual's acts in violation of a copyright owner's rights, according to the following formula:

Notwithstanding the provisions of section 106 [which specifies the five rights of a copyright owner], the fair use of a copyrighted work, . . . for purposes such as criticism, comment, news reporting, teaching (including multiple copies for classroom use), scholarship, or research, is not an infringement of copyright. In determining whether the use made of a work in any particular case is a fair use the factors to be considered shall include:

- (1) the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes;
- (2) the nature of the copyrighted work;
- (3) the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and
- (4) the effect of the use upon the potential market for or value of the copyrighted work.

However, it is important to note that the perspective of many researchers regarding proper delineation of the "public interest", as opposed to private intellectual property rights, may not be shared by the particular federal judge who hears the litigation arising from hypertext activities. Moreover, under the U.S. legal systems, a judge is required to follow prior court rulings ("res judicata"), which serve as the "operating system" for the maintenance and further definition of the law. In particular, research activities targeted at product commercialization may have significantly reduced changes for legal immunity through the "fair use" doctrine.

The moral: if you plan to reproduce, modify or display the code, proprietary structure and design, or documentation of another developer, you may need a written license agreement. In any event, early discussion and coordination with copyright counsel, based on a specific review of both your planned activities and the materials originally created by third parties, can help identify your obligations and rights.

6. **Who Me, A "Contributory Infringer"?** Copyright infringement does not require the specific mental intent to violate the copyright law. Unauthorized exercise of any one of the five rights established by the copyright statute automatically constitutes infringement (unless "fair use" or one of the other specific, narrow statutory privileges absolves the activity).

A corollary of this principle is that institutions and companies should also avoid indirect copyright infringement.

Vendors of "copy-unlocking" software (i.e., application programs written to defeat "copy-protection" systems), videocassette recorders, and other equipment which enables third parties to infringe copyrights, have on occasion, been sued for "contributory infringement". Happily, such lawsuits are infrequent.

III. PRODUCT DESIGN AND AVOIDANCE OF PRODUCTS LIABILITY

"Murphy was an optimist".

"In a network of 1000 nodes, it is easy to imagine that information could become hard to find or even forgotten altogether".—*A Survey of Hypertext*, p. 48 [Conk87]

Concerns about juries, liabilities, and insurance are not limited to developers and manufacturers of such products as pharmaceuticals and automobiles, or such services as neurosurgery and architecture. The minimization of products liability exposures - beyond being an obvious objective of most corporations and institutions - is a significant current goal of the managers of many information-related companies.

For example:

Item: In its recent public offering, Oracle Corporation chose product "bugs" and the burn-in/debugging process as the first source of risk to be considered by potential investors—beyond the threats to company revenues posed by its competition, financial requirements, and the challenge of cultivating creative talent.

Item: Industry leader Lotus Development Corporation has already been sued for the alleged failure to design its Symphony product in a manner that would preclude incorrect use by a customer. In the case of *James A. Cummings, Inc. v. Lotus Development Corporation and International Business Machines*, the plaintiff construction company allegedly incurred approximately \$250,000 in losses from using software in a way that was possible (given the product's design), but that contradicted instructions specified in the user manual. "Computer law" specialists noted that the lawsuit raised, for the first time, the issue of whether information technology vendors must design their products so to minimize the risk of harm to users and third parties.

The moral of the story is that technology designers and developers can assist their organizations in maintaining company profitability and reputation—in addition to helping novice users avoid difficulties in their data processing and management—by adopting "preventive design" techniques. [Jone88]

For example:

- In sensitive applications, the user's commands and "path" could be trapped and available to the user on a real-time basis (e.g., as a "help" function).
- To assist users in maintaining fragile data, hypertext systems might be "enhanced" by adding a number of "preventive" functions—such as automatic interim data saves, two-command data erasure procedures, and screen notices reminding system operators to back up their data onto physically separate media and to store such media off-site.

- To help protect companies from unfounded claims and lawsuits (an increasing concern with the fast growth of "computer litigation"), product designers should keep complete records of the code delivered to customers (to be able to prove any later code-modification by users that introduces errors into a previously-working system), "update" their contracts to comply with recent court rulings (e.g., the September, 1985 *RRX Industries, Inc. v. Lab-Con, Inc.* case, in which a custom software programming company was held liable for money damages in excess of their services fees, their contract "boilerplate" to the contrary), and "enhance" their contracts by specifying the user's responsibilities for proper utilization of the product.

IV. OTHER LEGAL CONCERNS

"There's many a slip between cup and the lip".—*Old proverb*

The legal problems and partial recommended solutions discussed above are limited to issues that seem most likely to confront a large number of researchers, programmers, and companies. However, the legal environment includes a number of other parameters that should be recognized.

For example:

- The rapid evolution of information technology has engendered a number of disputes regarding the meaning and impact of computer-related contracts. For example, fierce negotiations and actual litigation have been spawned by different interpretations (and varying economic interests) relating to the significance of contractual commitments to supply "updates, enhancements and new versions". With the advent of "diskless workstations", CD-ROM-stored data bases and application software, and other new tools and techniques, the importance of defining contract vocabulary, obligations, and expectations has become more important.
- Disappointments regarding expected royalties have plagued creative personnel in the movie, book, and other industries for years. Should developers of hypertext tools who hand over their hard-wrought inventions to others for marketing have concerns about shortfalls in royalty payments?

Absolutely. First, "per-copy" royalty schemes are threatened by advances in local area (and dispersed) networks, distributed data processing (including new large-computer environment distributed operating system software), and new data communications protocols (such as IBM's proposed Local Unit 6.2 Advanced Peer-to-Peer protocol) that undercut the need for "one copy for each box". [Dyso87]

Second, the industry is full of business relationships marred by diverging opinions regarding the manner and degree that a particular product should have been marketed (resulting, for example, in litigation about the impact of contractual or implied "best efforts" marketing obligations).

Third, recent interpretations of federal bankruptcy law threaten the basis of many technology licensing arrangements, as the trustee appointed for a company in reorganization proceedings may have the power to unilaterally reject prior contractual commitments.

- As Data General Corporation, Apple Computer, Inc., and other companies have learned, "antitrust and trade regulation" law is not a concern solely for back-room, cigar-chomping monopolists of the industrial era. Rather, current court rulings, federal legislation, administrative standards, and state codes regulate advertising content, product pricing, distribution practices, and even product design. For example, in certain circumstances, product "bundling" may contravene federal rules against "tying" two products together. Apple successfully defended its policy of prohibiting mail-order dealers, but only at the cost of protracted, expensive litigation, and Data General has paid out over \$50,000,000 to date to settle litigation arising from their refusing third parties access to the interface specifications for the operating system for a new minicomputer.

- Some foreign countries eager to develop their national economies have seized upon development of the indigenous computer industry as a strategy for creating a stronger economic base. As a technique to accelerate the computer industry, a number of Latin American countries have implemented "technology transfer" regimes, which limit the conditions under which software, hardware, and other information technologies may be imported—and how long ownership rights and contractual agreements will be honored after such importation. Electing to market your product in Brazil, Venezuela, Columbia, or other countries may forfeit your proprietary rights, on a world-wide basis, five years after entering that particular marketplace.

V. CONCLUSION

"Those who do not understand are doomed to repeat it".—*Winston Churchill*

Copyright © 1987 Henry W. Jones, III, Esq., Atlanta, GA, USA. All rights reserved. This article is provided solely for educational purposes, and only represents the author's current personal opinions, which should not necessarily be attributed to his clients or his firm. The author serves as a Columnist for *CD-ROM Review* and *International Computer Law Advisor*, and is a member of the Board of Editors of *The Computer Lawyer* and *Software Protection*. His Atlanta-based law firm represents software, on-line publishing, optical storage, hardware, data communications, and other computer-related companies.

REFERENCES

- [Conk87] Conklin, Jeff, "A Survey of Hypertext," Microelectronics and Computer Technology Corporation Software Technology Program Technical Report STP-356-86, Rev. 1, Austin, TX, February, 1987.
- [Divo87] Divoky, Diane, "Image Copyright: Are You Breaking the Law?," *Publish!*, August, 1987.
- [Dyso87] Dyson, Esther, "Text Tools: The Joy of Linking," *Release 1.0*, EDventure Holdings Inc., New York, NY, September, 1987.
- [Jones83] Jones, Henry W., III, "The Idea of Games, the Expression of Aliens, and the Underlying Computer Software: The 1976 Copyright Act and Videogame Litigation," *Protecting Computer Software and Games*, Practising Law Institute, November 1983 and *1 Journal of Copyright, Entertainment and Sports Law* 17, Tennessee Bar Association, Nashville, TN, May, 1983.
- [Jones87a] Jones, Henry W., III, "Preventive Law Steps for High Technology Companies: Maintaining Your Firm's Profitability and Viability by Pro-Active Legal Risk Management," St. Paul Fire and Marine Insurance Company High Technology Risk Management Symposium, Atlanta, GA, February, 1987 (available from author).
- [Jones87b] Jones, Henry W., III, "Contract Vocabulary: New Industry Focus," *Softshare*, Southeastern Software Association, February, 1987.
- [Jones87c] Jones, Henry W., III, "Copyright Protection for Information Products: Current Problems—and Solutions," *Information Times*, Information Industry Association, April, 1987.

- [Jone87d] Jones, Henry W., III, "Copyrights, Contracts and Other Legal Parameters in Designing, Developing, and Distributing CD-ROM Products," *CD Data Report*, April, 1987.
- [Jone&Dailey86] Jones, Henry W., III, and Dailey, Michael A., "Software Copyright Debate: Federal Law Protects Program Design," *Software Publishers Association News*, September, 1986.
- [Jone&Dailey87a] Jones, Henry W., III, and Dailey, Michael A., "*Whelan Associates, Inc., v. Jaslow Dental Laboratory, Inc.*: Towards a New Definition of Copyrightable Expression," *European Intellectual Property Review*, Cambridge, U.K., February, 1987.
- [Jone&Dailey87b] Jones, Henry W., III, and Dailey, Michael A., "*Whelan Associates, Inc., v. Jaslow Dental Laboratory, Inc.*: The Expanding Range of Copyrightable Expression," *European Intellectual Property Review*, Cambridge, U.K., May, 1987.
- [Offi86] Office of Technology Assessment, Congress of the United States, *Intellectual Property Rights in an Age of Electronics and Information*, U.S. Government Printing Office, April, 1986.



Software

Abstraction Mechanisms in Hypertext

Pankaj K. Garg

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0782
garg@cse.usc.edu

ABSTRACT

Abstraction is the means by which information can be stored and retrieved from an information structure at different levels of detail and from different perspectives. As such, abstraction mechanisms in hypertext are interesting to study and evaluate. In this paper we study the abstraction mechanisms in hypertext from a theoretical perspective. Abstractions then become various first-order logic formulae. Specifically we consider abstractions: sets, sequences, aggregations, generalizations, revisions, and information structures. Interesting results of this work are the definition of level of generality of a hypertext node, the demonstration of revision histories as a partial order, and the notion of compatible-similar nodes. Also defined in this paper is the notion of primitive hypertexts versus application hypertexts, and the usage of attributes of nodes (illustrated by the use of keywords) across various abstractions. An illustration of the concepts is given using the contexts mechanism suggested by Delisle and Schwartz [DS87].

1 INTRODUCTION

The notion of abstraction has been of considerable value to the fields of programming languages [Sha84], databases [AH84,MS82,SS77] and knowledge representation [Bra85,SFG85]. In the case of hypertext systems, abstraction mechanisms can be useful for the following reasons:

1. Information Relevancy:

As Bush pointed out in his vision of the Memex system [Bus45], filtering information to suit the needs of the user is important for an effective information retrieval system. Filtering information in a hypertext can be considered equivalent to defining *views* in a database or using a query language to retrieve information from a database. The utility of the query languages in databases can hardly be debated, leading us to the hypothesis that similar mechanisms will be fruitful for hypertext.

2. Structure of Information:

This is best illustrated from the research domain where an author/reviewer wishes to look at the structure of an article rather than its contents. The use of outlines or table of contents demonstrates such a need. Note that this is a concept different from a database schema as the schema of a database does not change with the evolution of the database. Also, a schema is the structure of multiple information instances while here we are talking of the structure of independent information instances.

3. Collection of Information Units:

One of the chief uses of a hypertext as espoused by researchers in the field [Con87,Nel81,Tri83] is that the readers of the information in a hypertext can impose their own structure on the hypertext by selecting appropriate nodes. This idea is similar to the concept of aggregation in databases [SS77]. Aggregation is the abstraction by which a collection of nodes of the hypertext can be collectively referenced by a single name. As pointed out by Smith and Smith [SS77], the collection (in aggregation) refers to an abstract real world entity instead of just being an arbitrary collection of objects. Arbitrary collection of object-nodes can be represented as sets or sequences.

4. Multiple Streams of Development:

An important part of hypertext systems is the support that they provide for collaborative work [DS86,FS86,Tri83]. Most of the time, different authors of a hypertext will be working on different parts of the hypertext and without 'collisions'. Sometimes, however, the authors would be working on the same part (of the hypertext). For instance a software module might be developed in parallel by two programmers. In such cases it is imperative that provision be made for multiple authors to create their nodes and later merge their efforts. Delisle and Schwartz [DS86] have proposed a concept of *contexts* for this end. The concept of *layers* developed by Goldstein and Bobrow [GB84] is also relevant here.

5. Domain Information rather than the information in a particular Hypertext:

One area that (to our knowledge) has not been investigated by hypertext researchers is the consideration of information of the domain embedded in the hypertext as opposed to an instance of the information for the domain. Domain knowledge is of use to the user of the hypertext (author or reader) who is novice in the domain of the hypertext. For example, an author who is writing for a new journal-X can benefit from the fact that the editors of journal-X like the related work section to appear after the exposition of the main ideas of the paper. The notion of 'primitive links' defined by Trigg [Tri83] can be extended to the notion of primitive links and *primitive nodes* to provide mechanisms for this purpose.

6. Keeping different revisions:

This aspect of hypertexts is crucial to software hypertexts¹. It involves the ability to keep different revisions in a manner semantically suitable for the application. In some operating systems, backup versions of files are kept automatically. The introduction of tools such as *sccs*, *rcs*, and *diff* for UNIX [UNI] has led to more sophisticated methods for revision maintenance. However, the use of these tools for a collection of related files is still quite laborious. In hypertext, on the other hand, because of the strong structural nature of the information base, we are in a position to define the semantics of revisions at a level closer to the application domain. The utility of such a manner of keeping revisions has been pointed out by many researchers [Con87,DS87,Nel81].

In this paper we will consider the mechanisms by which these abstractions can be supported in a hypertext. For this purpose, a theoretical model of a hypertext is presented in Section 3. A brief description of set theory required for this model is given in section 2. In section 4 we discuss aggregations, generalizations, revisions, compatibility, and Primitive Hypertexts versus Application Hypertexts. In section 5 we discuss how information filtering can be supported in our model. In section 6 we illustrate the use of the model by providing a definition of Contexts in our framework. In section 7 we discuss previous work which has influenced our model and finally in section 8 we conclude and suggest directions for future work.

2 MATHEMATICAL PRELIMINARIES

We use the following notions in subsequent sections. The reader familiar with these can skip to section 3 and the reader uncomfortable with them is referred to [Sto79].

An *object* is a concept which can stand for anything depending on the context and has an identifier by which it is accessed or referred to. When we say *A* is an object we mean that there exists an object or thing which is referred to by the symbol *A*. The object may be physical (like a file or a text string) or may be a conceptualization (like a the tree object representing the structure of an article). We use *A, B, ...* to represent object identifiers and *X, Y, Z* to represent variables for object identifiers.

We use the following set operations:

1. A one place predicate, *SET*, such that $\forall X(SET(X) \Leftrightarrow X \text{ is a set of objects})$
2. A function *MEMBERS* which maps a set object to the set of its elements.
3. Union, \cup is the operator on two sets which results in a set which has elements from both the sets.

¹A hypertext containing information related the development, use, and maintenance of a software system.

4. Intersection, \cap is the operator on two sets which returns a set which has the elements common to both the sets.
5. Two sets are equal iff they have the same elements.

We define the set of finite sequences of objects, O_q . The following are defined for sequence objects:

1. A predicate SEQUENCE such that $\forall X(\text{SEQUENCE}(X) \Leftrightarrow (X \in O_q))$
2. A function LENGTH from O_q to non-negative integers such that $\text{LENGTH}(X) =$ the number of objects in X . Note that the existence of this function implicitly defines the sequences to be countably infinite.
3. A function LIST from O_q to $O \times O \times O \cdots$ such that $\text{LIST}(X) =$ an ordered tuple of the elements in X , denoted by $\langle X_1, X_2, \dots \rangle$ or X_1, X_2, \dots as convenient.
4. An operator 'in' such that $(X \text{ in } Y)$ is true if the object X is in the Sequence Y and false otherwise.

We also use the notions of a partial order,

Definition 2.1 (Partial Order) *An ordered pair $\langle S, R \rangle$ where S is a set, and R is a relation from S to S , is a partial order, if*

1. R is asymmetric,
2. R is reflexive, and
3. R is transitive.

The following definition of the cover of an object is used to determine the height of an object in a partial order.

Definition 2.2 (Cover) *For a partial order $\langle S, R \rangle$, an element X of S covers an element $Y (\neq X)$ of S , if $R(Y, X)$ and for no Z in S ,*

$$(Z \neq X) \wedge (Z \neq Y) \wedge R(Y, Z) \wedge R(Z, X)$$

This will be written as $\text{covers}(X, Y, \langle S, R \rangle)$ which can be read as "X covers Y in the partial order $\langle S, R \rangle$."

The height of an object in a partial order is defined as:

Definition 2.3 (Height) *For a partial order, $\langle S, R \rangle$, the height of the objects in S is a function from S to the set of natural numbers, such that:*

1. $(R(X, Y) \wedge (X \neq Y)) \Rightarrow \text{height}(X) < \text{height}(Y)$; and

2. If X covers Y then $height(X) = height(Y) + 1$.

Minimal objects in S are assigned the height of 1.

3 A MODEL FOR HYPERTEXT

Definition 3.1 A Hypertext η , is a set consisting of,

1. A set of primitive objects, P_o and information objects, I_o ; $P_o \cap I_o = \emptyset$.
2. A set of predicates, π ; and
3. A set of attributes (or properties), A .

For a hypertext η , $P_o[\eta]$ represents the set of primitive objects of η , $I_o[\eta]$ represents the set of Information objects of η , $A[\eta]$ represents the set of attributes in η , and $\pi[\eta]$ represents the set of predicates in η .

3.1 Predicates

π is a set of predicates. We will sometimes also refer to them as relations. π is composed of three sets:

1. π_1 : a set of 1-place predicates which characterize objects, e.g., $SET(X)$ is true if X is a set object, false otherwise.
2. π_2 : a set of 2-place predicates such that if $P(X_1, X_2)$ is true, then the relationship P exists between objects X_1 and X_2 , and if it is false then the relationship *does not* exist.
3. π_3 : a set containing the element 'Property' which is a 3-place predicate such that if $Property(X, Y, Z)$ is true then object X has Z as the value of its property Y . If $Property(X, Y, Z)$ is false then the object X does not have Z as the value of property Y .

3.2 Attributes

Attributes of objects are properties (of objects) which can be used to identify the objects from different perspectives. This provides a mechanism above object identifiers to identify the objects. Attributes are distinguished from relationships in that, relationships exist between objects whereas the range of attributes may not always be objects. For instance, the number of lines in an object is better treated as an attribute rather than relationships between objects. Attributes have ranges which are dependent on the attribute, e.g., number of lines attribute has the range *natural numbers*. In case the value of an attribute is unknown it is represented by a special symbol \perp . The range set of a property P is denoted by $RANGE(P)$. Sometimes attributes are inherited from other objects (as shown later). We define a *well-attributed hypertext* as one in which the attribute values of all

the objects can be determined as either a value from the range set of the attribute or as \perp . We'll assume a well-attributed hypertext in the sequel.

3.3 Objects and their Instantiations

P_o is a set of distinct symbols denoting the primitive objects of η . I_o is a set of symbols which denote information objects (objects containing information: text, graphics, audio signals, etc.). The set

$$O = P_o \cup I_o$$

is called the set of objects. In the sequel when we talk of information objects we mean text-objects only, unless otherwise stated.

Information objects (I_o) can be related to the primitive objects (P_o) by the predicate $INSTANCE_{OF}$ and the function $INSTANCES$. $INSTANCE_{OF}(X, Y)$ means that X is an information object which is an instance of the primitive object Y . $INSTANCES(X) = \{X_1, X_2, \dots\}$ means that $INSTANCE_{OF}(X_1, X)$, $INSTANCE_{OF}(X_2, X)$, \dots .

Definition 3.2 A Hypertext η is a strongly defined hypertext iff

$$\forall X((X \in I_o) \Rightarrow (\exists Y(INSTANCE_{OF}(X, Y))))$$

This captures the notion that all information objects have their primitive objects defined. Alternatively, all information nodes are chosen from a pre-existing pool of primitive nodes, and the user of the hypertext is not allowed to create a new node unless a primitive node of the same category is first created. In the sequel whenever we refer to hypertexts, we'll mean strongly defined hypertexts unless otherwise stated.

An information object inherits the attributes of the corresponding primitive object unless the information object overrides the property by defining a new one. This relationship between the primitive objects and the information objects can be expressed by the following axiom:

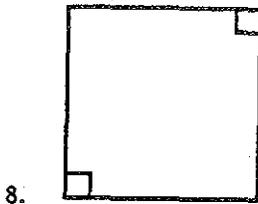
Axiom 3.1 An information object shares the attributes of the corresponding primitive object, unless otherwise stated.

$$\forall X, Y [(X \in I_o) \wedge (Y \in P_o) \wedge INSTANCE_{OF}(X, Y) \wedge \exists Z, P(\text{property}(Y, P, Z) \wedge (Z \in RANGE(P)) \wedge \neg \exists S(\text{property}(X, P, S)) \Rightarrow \text{property}(X, P, Z))]$$

The information content of an information object is represented as the property INFO for the object. The value of the property INFO is a set containing 2-tuples (*position, value*) where the (*position*) is a real number identifying the position of the (*value*) in the information object. For instance, consider the following information object:

1. A square is a

2. :geometrical
3. figure which is a
4. :rectangle
5. *(pointer to the object containing information about a rectangle)*.
6. with all sides of the rectangle being equal.
7. It can be graphically represented as:



If this information object is referred to as SQUARE then $\text{property}(\text{SQUARE}, \text{INFO}, \text{square-info})$ is true where square info is a set object² such that:

$$\text{members}(\text{square_info}) = \{ \langle 1, \text{"A square is a"} \rangle, \langle 2, \text{: geometrical} \rangle, \dots, \langle 4, (\text{IS-A}, \text{RECTANGLE}) \rangle, \dots, \langle 7, (\text{DEPICTED-AS}, \text{Figure_object}) \rangle \}$$

Keywords are marked in the text with a ':' in front of the keyword. Links and figures are also embedded in the text. This automatically asserts the truth value of certain predicates in π_2 . For instance, if there is an object called RECTANGLE, then the above definition would assert the relationship IS-A(SQUARE, RECTANGLE).

Definition 3.3 (line) *An instance of the tuple (position, value) is called a line.*

Definition 3.4 *If S is a set of lines, then position[s] is the projection of the position field of the lines in S.*

If S is a set of lines, then value[s] is the projection of the value field of the lines in S.

The position numbers are used once only. That is, even if a line is deleted, its position number is not used for another line, but a new one generated as desired.

Notice that it is not necessary that the keyword (or any line for that matter) be displayed for a particular presentation of the information object. It is quite possible to use a language such as the one presented by Shasha [Sha85] to select the lines to be presented (which are called 'fragments' by Shasha).

²It could equally well have been treated as a sequence object

4 ABSTRACTIONS

We now define the abstractions that can be supported based on the notions developed in the previous section.

4.1 Aggregations

Aggregation is the mechanism by which a collection of objects can be referenced by an identifier. For example, a set of variable declarations coupled with a set of functions can be referred to as a module in the LISP programming language. Hence we can talk about a module without having to bother about the details of what constitutes the module. Aggregate objects are actually Sequence objects with special axioms (constraints) imposed on them.

Hence O_x , the set of all aggregate objects, is a subset of O_q . Hence, all the operations defined for elements of O_q are valid for elements of O_x . In addition, aggregations must satisfy the following intuitive constraints:

Axiom 4.1 *An aggregate object has unique constituents.*

$$\forall X, Y [AGGREGATE(X) \wedge AGGREGATE(Y) \wedge (LIST(X) = LIST(Y)) \Rightarrow (X = Y)]$$

AGGREGATE is a one place predicate which is true if the argument object is an aggregate object. This axiom distinguishes simple sequences from aggregate objects. As opposed to simple sequences of objects, aggregate objects must correspond to some real world abstraction and therefore the same collection of objects should not be viewed as two different aggregate abstractions (to maintain the integrity of the hypertext).

Axiom 4.2 *If an aggregate object is a primitive object, then all the constituents of that object must be primitive objects.*

$$\forall X (AGGREGATE(X) \wedge (X \in P_o) \Rightarrow \forall Y [(Y \text{ in } LIST(X)) \Rightarrow (Y \in P_o)])$$

This axiom keeps the distance that needs to be maintained between the primitive objects and the information objects. If a primitive object is defining an aggregate, then it should be composed entirely of primitive objects and should not rely on the information that is to be filled in any of the primitive objects.

The next axiom formalizes the notion of how instances of aggregate objects can be formed from the primitive aggregate objects.

Axiom 4.3 *The instance of an aggregate object is formed by the instances of the constituent objects.*

$$\begin{aligned} \forall X, Y \quad & [(AGGREGATE(X) \wedge (X \in INSTANCES(Y))) \Rightarrow \\ & (AGGREGATE(Y) \wedge (Y \in P_o) \wedge \forall Z [(Z \text{ in } LIST(X)) \Rightarrow \\ & [(Z \text{ in } LIST(Y)) \vee \exists T [(T \text{ in } LIST(Y)) \wedge Z \in INSTANCES(T)]]])] \end{aligned}$$

Thus instances of aggregate objects are allowed to have primitive objects as their constituents. This is because if we insist on instances of aggregate objects not having primitive objects in the constituents then we cannot cater for the situation when the hypertext is partly defined, i.e., only some nodes of the hypertext have information stored in them and others do not.

Notice that an aggregate object closely resembles the notion of tuples in relational databases. Hence we can define projections, joins, selections, etc. for aggregations also. The extension is quite straightforward and the operators required can be determined by the application. We leave this to the interested reader.

4.2 Generalizations

A Generalization is the abstraction by which a collection of objects is referred to by a generic object which captures the essential similarity between the objects. For example, a document is a generalization of research article, survey article, lab report, etc. Notice that, unlike aggregation, generalization makes the individual objects lose their individualness. General objects share the properties of the objects that they generalize.

By explicitly naming a generalized object we gain the following:

1. The possibility to apply *operators* to generic objects: e.g., give me a *list* of all the documents written by author X (the query does not have to enumerate all the kinds of documents that exist in the system).
2. The specification of *attributes* to generic objects, e.g., total *number* of documents written so far.
3. The specification of *relationships* which the generic objects participate in, e.g., a document *refers to or cites* another document.
4. The definition of *default properties* of the objects: unless otherwise stated assume that a *document is written in 1987*.

One form of generalization is apparent in the definition of the INSTANCES function. It is a special kind of generalization in which a primitive object is a general object that has instances as specialized objects. The information object can inherit properties from the primitive object unless it overrides the property (Axiom 3.1).

Generalizations are defined by a (partial) function from O to O , such that: if $GENERALIZE(X) = Y$ then Y is the generalization X . Hence the sample generalization can be stated as:

$$GENERALIZE(\text{research-article}) = \text{document} \wedge$$

$$GENERALIZE(\text{review-article}) = \text{document} \wedge$$

$$\begin{aligned} \text{GENERALIZE}(\text{survey-article}) &= \text{document} \wedge \\ \text{GENERALIZE}(\text{comment-article}) &= \text{document} \end{aligned}$$

Axiom 4.4 *The relation GENERALIZE is asymmetric, non-reflexive and transitive.*

The domain and range objects of GENERALIZE could be abstractions themselves.

First we define the relationship between primitive objects and generalizations which we mentioned before:

Axiom 4.5 *If X is an information object which is an instance of the primitive object Y, then Y is a generalization of X.*

$$(\text{INSTANCE}_{OF}(X, Y) \Rightarrow (\text{GENERALIZE}(X) = Y))$$

Axiom 4.6 *If Y is a generalization of X, and X is an information object and Y is a primitive object, then X is in the set of instances of Y.*

$$\begin{aligned} &[[\text{GENERALIZE}(X) = Y] \wedge (X \in I_o) \wedge (Y \in P_o)] \\ &\Rightarrow (X \in \text{INSTANCES}(Y)) \end{aligned}$$

Theorem 1 *If X is a generalization of Y and Y_m is in the instances set of Y, then Y_m is in the instances set of X.*

$$\begin{aligned} \forall X, Y, Y_m &[[\text{GENERALIZE}(Y) = X] \wedge (Y_m \in \text{INSTANCES}(Y))] \\ &\Rightarrow (Y_m \in \text{INSTANCES}(X)) \end{aligned}$$

PROOF: Follows from the application of axiom 4.6 and the transitivity of the generalize relation. \square

Lemma 1 *Define the relation*

$$\text{GENERAL}(X, Y) = [(X = Y) \vee (\text{GENERALIZE}(X) = Y)]$$

The ordered pair $\langle O, \text{GENERAL} \rangle$ is a partial order.

The proof follows from the definition of GENERALIZE as asymmetric, transitive, and reflexive relation.

This last lemma allows us to define the level of generality of an object as follows:

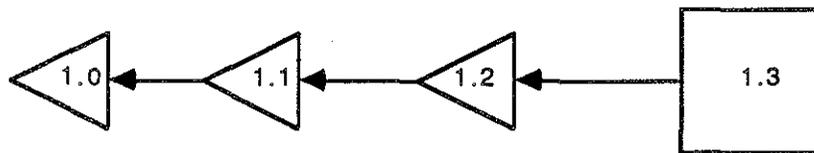
Definition 4.1 (Level of Generality) *The level of generality of an object $X \in O$, is defined as the height of the object in the partial order $\langle O, \text{GENERAL} \rangle$.*

4.3 Revisions

Revision of an information object is an information object which has basically the same information as in the original object, with a few changes. We note that revision maintenance is a form of abstraction because the most recent revision of an information object when viewed, hides the detail that there is a history associated with its existence.

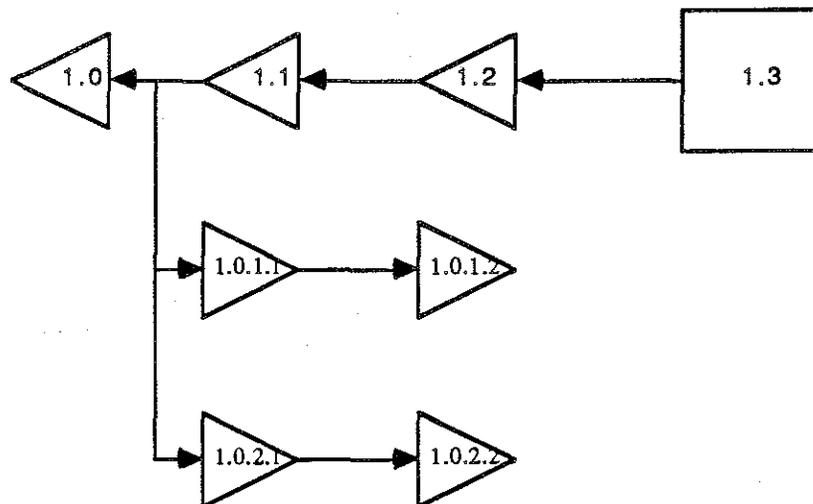
Tichy [Tic82] has used a model for a Revision Control System (RCS) for storing the history of a file (and a set of files) in the Unix operating system environment. We formalize the same model in our framework. The advantages of Tichy's model are: (1) it stores changes to information instead of multiple copies, and (2) allows for merging of two revisions which originated from the same file but changed the files in different places.

The model starts with an information object (file) and as it is modified, creates new *revisions* of it and stores the elder objects as a set of lines to be added and deleted from the new object to get the old object. Graphically, this is represented as:



where the triangle represents a *delta*. The history thus maintained is called a "trunk". The numbers are automatically assigned to the newer objects as they are created (unless modified by the user). This is the simplest way one can maintain a history in RCS.

The next level of complexity is introduced when "branches" are allowed in the trunk. This allows the development of information objects to follow different paths at the same time. Hence we could have a *revision tree* as:



The numbering can be changed by the user with certain constraints.

To use the model of RCS in our framework, we define a primitive object of the type 'delta'. These objects will have the properties 'add' and 'delete' where add is a set of lines to be added and delete is a set of lines to be deleted. A revision of an information object is then a relationship between three objects, the old information object, the delta object, and the new information object. This we can translate into two relationships, (1) HAS_{DELTA} , the relationship between the old information object and the delta that is needed to create it, and (2) $HAS_{REVISION}$, the relationship between the old information object and the new information object that is needed to create it. Clearly these two relationships have a correspondence with the model that underlies RCS. The arcs in the above diagram represent relationships, the numbering represents information objects, and the delta's are represented as relationship to a delta object. Notice that the new information object itself might contain a delta and a pointer to another information object.

To formally define a model we first introduce the attribute *date* to the set of objects. We define date as:

Definition 4.2 (Dates) *For all objects X in O , there is a property $DATE$ in A such that $Property(X, DATE, I)$ where I is a natural number. If A and B are two distinct objects and A is created before³ B , then:*

$$(Property(A, DATE, I_1) \wedge Property(B, DATE, I_2)) \Rightarrow (I_1 < I_2)$$

From this definition it is easy to see that:

Lemma 2 *If HAS_{date} is a relation from the set of objects, O , to the set of natural numbers, N , such that:*

$$HAS_{date}(O_1, I_1) \Leftrightarrow Property(O_1, DATE, I_1)$$

then the ordered pair $\langle O, HAS_{date} \rangle$ is a total ordering.

Now we define the relation $HAS_{revision}$.

Definition 4.3 *If objects X and Y are revisions of each other (in the sense of RCS) then $HAS_{revision}(X, Y)$, considered as "X has revision Y", if*

$$HAS_{date}(X, I_1) \wedge HAS_{date}(Y, I_2) \wedge (I_2 > I_1)$$

From the definition one can easily see that:

Axiom 4.7 *$HAS_{revision}$ is not symmetric, is not reflexive, and is transitive.*

$$\begin{aligned} \forall X, Y [(X \neq Y) \Rightarrow (HAS_{revision}(X, Y) \Rightarrow \neg HAS_{revision}(Y, X)) \wedge \\ \neg HAS_{revision}(X, X) \wedge \\ \{(HAS_{revision}(X, Y) \wedge HAS_{revision}(Y, Z)) \Rightarrow HAS_{revision}(X, Z)\}] \end{aligned}$$

³We assume that simultaneous creation is impossible

Next we define the notion of the "ANCESTOR" of an object.

Definition 4.4 *An object (Y) is the ancestor of an object (X) if X is zero or more steps removed from Y by the relation HAS_{revision}.*

$$\forall X, Y [HAS_{ancestor}(X, Y) \Leftrightarrow [(X = Y) \vee HAS_{revision}(Y, X) \\ \vee \exists Z (HAS_{revision}(Z, X) \wedge HAS_{ancestor}(Z, Y))]]$$

The relation $HAS_{ancestor}(X, Y)$ is read as "X has ancestor Y".

Based on these definitions we can observe the following:

Observation 4.1 *If both X and Y have a revision Z, then they must share a common ancestor.*

$$\forall X, Y, Z (HAS_{REVISION}(X, Z) \wedge HAS_{REVISION}(Y, Z) \Rightarrow \\ \exists T (HAS_{ANCESTOR}(X, T) \wedge HAS_{ANCESTOR}(Y, T))$$

Observation 4.2 *The ordered pair $\langle O, HAS_{ancestor} \rangle$ is either:*

1. *An equivalence class if:*

$$\forall X, Y [(X \in O) \wedge (Y \in O) \wedge (X \neq Y) \Rightarrow \neg HAS_{ancestor}(X, Y)]; \text{ or,}$$

2: *A partial order.*

Let $ANCESTORS(X) = \{Y | HAS_{ancestor}(X, Y)\}$.

Observation 4.3 *The Revision History of an object X is the partial order $\langle ANCESTORS(X), HAS_{revision} \rangle$.*

4.4 Compatibility

We define Similar objects as objects which share a common ancestor:

Definition 4.5 (Similar Objects)

$$\forall X, Y [SIMILAR(X, Y) \Leftrightarrow \exists Z (HAS_{ANCESTOR}(X, Z) \wedge HAS_{ANCESTOR}(Y, Z))]$$

An obvious concern for revisions is whether two objects are compatible with each other. The syntactic compatibility can be checked by comparing the deltas that need to be used to derive the objects. If there is no conflict in the deltas and the objects share a common ancestor then the two objects are compatible with respect to that ancestor. This means that the objects can be combined using the merge operation of RCS. To check this we first define the delta-sequence as the sequence of deltas that needs to be combined to obtain an information object. If the delta-sequence is empty then the information object does not require the application of any delta.

Definition 4.6 (Delta Sequence) *A delta-sequence of an object X with respect to an object Y , is the sequence of delta's that needs to be applied to X to change it into Y (in the right order). If the delta sequence is empty then either $X=Y$ or X and Y are not similar objects.*

The 'application' of a delta simply means the removal of the 'delete' lines and the addition of the 'add' lines contained in the delta.

Using our notions of set and sequence objects, we can define the 'add' and 'delete' sets for the delta-sequence as follows:

Definition 4.7 *The add set of a delta-sequence, $\langle d_1, d_2, \dots \rangle$ is defined as the projection on the (position) field of the lines in the union of the add lines of $\langle d_1, d_2, \dots \rangle$, and is represented as $ADD(\langle d_1, d_2, \dots \rangle)$.*

The delete set of a delta-sequence, d_1, d_2, \dots is defined the projection on the (position) field of the lines in the union of the delete lines of $\langle d_1, d_2, \dots \rangle$, and is represented as $DELETE(\langle d_1, d_2, \dots \rangle)$.

Now we can define the notion of two compatible-similar information objects as follows,

Definition 4.8 (Compatible-Similar) *Two information objects X and Y are compatible-similar objects with respect to an information object Z if,*

1. $HAS_{ANCESTOR}(X, Z)$ and $HAS_{ANCESTOR}(Y, Z)$; and
2. If D_x is the Delta sequence to change X to Z and D_y is the delta sequence to change Y to Z then

$$\{ADD(D_x) \cup DELETE(D_x)\} \cap \{ADD(D_y) \cup DELETE(D_y)\} = \emptyset$$

4.5 Primitive Hypertext and Application Hypertext

An interesting dichotomy of perspectives on databases is provided by the "schema" of the database and the set of tuples "in" the database. A similar dichotomy exists in hypertexts where the first one corresponds to the structure of an hypertext and the second to the information in an hypertext. These perspectives can be offered through our model by classifying the relationships defined through the set π .

In order to define the perspectives, we define the following:

Definition 4.9 Relationships: *A relationship is either:*

1. $P(X)$ where $P \in \pi_1$ and $P(X)$ is true; the object of the relationship is X ; or
2. $P(X_1, X_2)$ where $P \in \pi_2$ and $P(X_1, X_2)$ is true; the objects of the relationship are X_1 and X_2 ; or

3. *Property*(X, Y, Z) where $\text{Property} \in \pi_3$ the object X has Z as the value of property Y ; the object of the relationship is X .

Definition 4.10 Instance Relationships: *The instance level relationships are those relationships in which the objects of the relationship are from I_o .*

Definition 4.11 Primitive Relationships: *The primitive relationships are those in which the objects of the relationships are from P_o .*

Definition 4.12 Hybrid Relationships: *The hybrid relationships are those in which the objects of the 2-place relationship have one object from P_o and the other object from I_o .*

From the definitions it is clear that,

Lemma 3 *The three sets of relationships are mutually disjoint.*

Using this formalism we can partition a hypertext such that the three partitions reflect the partitioning of the relationships in the hypertext. This will give us the perspectives similar to the schemas and sets of tuples in a relational database.

Lemma 4 *A hypertext η can be considered as the union of three hypertexts:*

η_I : *A hypertext which has $P_o[\eta_I] = \emptyset, I_o[\eta_I] = I_o[\eta], A[\eta_I] = A[\eta],$ and $\pi[\eta_I] = \pi[\eta].$*

η_P : *A hypertext which has $I_o[\eta_P] = \emptyset, P_o[\eta_P] = P_o[\eta], A[\eta_P] = A[\eta],$ and $\pi[\eta_P] = \pi[\eta].$*

η_H : *A hypertext which has $P_o[\eta_H] = P_o[\eta], I_o[\eta_H] = I_o[\eta], A[\eta_H] = \emptyset,$ and $\pi[\eta_H] = \pi_2[\eta]$ with the condition that for any 2-place predicate if the objects are both from P_o or both from I_o then the value of the predicate is false.*

This proposition gives us the perspectives that we were looking for as well as other interesting perspectives on hypertext systems. For instance, if we want to look at the properties of primitive objects only, we can look at the set $\pi_3[\eta_P]$. Similarly if we want to look at the hybrid relationships only, then we can look at the set $\pi[\eta_H]$. In this framework η_P corresponds to the notion of schemas in databases and η_I to a particular instance of a schema.

5 FILTERING

Information filtering can be provided in this model from some of the abstractions defined in this paper. The filtering that we are going to discuss in this section is the one that is achieved by qualifications attached to the objects and links in the hypertext. Hence most of the other abstraction mechanisms are outruled by this definition.

5.1 Keywords

Keywords have found extensive use for bibliographic information retrieval [Sal86]. In this model keywords can be attached to Information Objects by the KEYWORD attribute which has the range set as the set of strings formed with the alphabets $a \cdots z$. Keywords can be generated automatically or be defined manually [Sal86] but that is not the issue here. We are only interested in the fact that they are presented to our model as dictated by the definition of the 'line' in section 3.3. From this, we define the set of keywords for a information object as follows:

Definition 5.1 (Information Objects) *If X is a object and $X \in I_o$ then K is in the set of keywords for X iff $\langle n, : K \rangle$ is a line of X .*

A concern for this paper is how do the keywords reflect on the abstraction mechanisms presented in section 4. For this end, we posit the following definitions which will aid in formalizing the notions of keywords for the mentioned abstractions.

Definition 5.2 (Aggregations) *If $X = \langle X_1 \cdots X_n \rangle$ is an aggregate object and X is in I_o then the set of keywords for X is the union of the sets of keywords for $X_1 \cdots X_n$.*

This reflects our intuitive notion that the aggregate object represents a collection of objects 'collectively'.

Definition 5.3 (Generalizations) *If $X = GENERALIZE(X_1 \cdots X_n)$, and there is no Y such that $X = GENERALIZE(Y)$ and $Y \neq$ some X_i , then the set of keywords for X is the intersection of the sets of keywords for $X_1 \cdots X_n$.*

This reflects our intuitive notion that the generalized object captures the essence of the collection that it generalizes. From this definition and our understanding that primitive objects are generalization of the instances that they have, we can draw an immediate corollary that:

Corollary 5.1 *If X is a primitive object and is related to the objects $X_1 \cdots X_n$ by the relation $INSTANCE_{OF}$ then the set of keywords for X is the intersection of the sets of keywords for $X_1 \cdots X_n$.*

Definition 5.4 (Revisions) *Keywords for revisions can be assembled using definition 5.1 and noticing that any revision of an information object is available in its full.*

If $HAS_{revision}(X, Y)$ then the keywords of X are determined by definitions 5.1 through 5.3 and that of Y are determined by applying the appropriate delta sequence with only the keywords.

This gives us a clean semantics for the keywords to be used for abstract objects in the hypertext. The time complexity of computing the keyword set for any abstraction is dependent on how the set operations union and intersection are implemented and also on the storage mechanism for the lines

of an information object. But it is intuitively evident that the complexity will be polynomial in the number of lines in the information objects and the number of keywords in each object.

5.2 Object Attributes

In the previous sub-section we outlined how one particular attribute (KEYWORD) can be used to effectively form a filtering mechanism for information objects and their abstractions. Similarly the user of a hypertext can define other attributes depending on the needs of the domain for which the hypertext is built. For example, number of lines in an information object is also an attribute which can form a filtering mechanism.

5.3 Linkages

Linkages provide a mechanism for browsing through the structure of information rather than the content of the information. The links in our model have been defined as 2-place predicates between objects. In the definition of the INFO attribute of the information objects we have defined lines which carry information about the links for that object. This facilitates the definition of links across revisions in a manner analogous to the definition of keywords for revisions. The important question is how do links allow the filtering that they are intended to provide? An obvious answer is that the links can be used in a predicate calculus-like language to identify the appropriate objects. This requires further study but we hypothesize that the usage of a language such as PROLOG is apt for this purpose. Basically the predicates in the hypertext form the database of 2-place predicates for a PROLOG engine.

6 CONTEXTS: AN EXAMPLE

In [DS87] Delisle and Schwartz have suggested the use of contexts for partitioning a hypertext. We frame the definition of contexts as given by them, into our formalism. They have developed two notions of contexts, (1) A derive model, and (2) a merge model. Since the merge model subsumes the derive model, we'll consider the merge model alone.

Definition 6.1 (Contexts) *A context, ζ of an hypertext η is a subset of η , where*

$$O[\zeta] \subseteq O[\eta] \wedge \pi[\zeta] \subseteq \pi[\eta] \wedge A[\zeta] \subseteq A[\eta]$$

Merging and Instances Delisle and Schwartz define two issues with respect to this:

1. A single object can have multiple instances in different contexts; and
2. An object has an (global) identifier which identifies all the instances of that object in all the contexts, and a identifier (local) which identifies the instance of an object in a given context.

As we have seen in the model given in this paper, multiple instances of an object can be modeled through the primitive object and the instance object. Then the local identifier becomes the object identifier for the instance object and the global object becomes the primitive object for the instances. For this, we propose the following axiom:

Axiom 6.1 *For all objects X in a context ζ , X is an instance object and there is a primitive object Y in η such that $INSTANCE_{OF}(X, Y)$.*

Notice that links are not really first class objects in our model and so there is no corresponding notion of links for contexts, although links are embedded in different contexts through the objects that they relate and hence are different. Also note that this model is slightly more general than that of Delisle and Schwartz as they provide for only *revisions* of objects in different contexts.

Merging and Version (Revision) Histories As pointed out by Delisle and Schwartz, the maintenance of revisions (what they call versions) becomes semantically confusing when mixed with contexts. We have provided a simple semantics of revisions based on the relations HAS_{DELTA} and $HAS_{revision}$. To answer their question of what we should view the revision history of an object, we have formally defined the revision history of an object as in observation 4.3. A simple restriction to belong to a particular context(s) can tune the revision history to the users' needs.

7 RELATED WORK

This work is largely influenced by the work by Abiteboul and Hull on a theory for semantic databases [AH84]. An obvious difference between their work and ours is that their notions are developed for a object oriented database and not for hypertext systems. As such their model does not delve into issues which concern with information inside an object which has been of concern to us. This also leads to the disparity between the two models in that IFO does not concern itself with recording revisions of objects or the fact of maintaining linkages across revisions.

Tichy's work on RCS [Tic82] has had an impact on the model that we have used for revisions of objects.

Much of the work in hypertext systems has influenced our ideas in this paper [Con87]. Although the efforts of the researchers has not been for a theoretical model (except Sasha [Sha85]), the notion of aggregation is similar to the notion of paths suggested by Trigg [Tri83]. The motivation for the notion for revisions came from the work on Contexts by Delisle and Schwartz.

8 CONCLUSIONS AND FUTURE WORK

In this paper we have presented a set theoretical model of a hypertext and defined interesting abstractions on it to provide mechanisms to retrieve and input information to the hypertext from

various perspectives and level of details. The theoretical model was built to subjugate the complexity of grasping the conceptualizations that are possible on hypertext nodes and links. This also gives us the framework to explore new ideas about abstractions on hypertext and to be able to theoretically evaluate the ideas.

There are several directions in which future work along this direction can be pursued. An obvious thing to do is to implement these ideas into a hypertext system. An interesting aspect of this is to provide visual interfaces to these abstractions. We do not discuss this issue in this paper, but the interested reader is referred to [BH86,GS87a]. Another direction is to enrich the notions of abstractions that this model provides. This could be done by fine tuning the given abstractions or building new ones (such as contexts) on the old ones. A third possibility (which we are investigating) is to see how this model could be enriched by encoding domain knowledge for particular hypertexts. We are pursuing this for the domain of software engineering, where a hypertext of information related to the software system is built along with the development of the system (the system itself being part of the hypertext). The domain knowledge can be in the form of situation calculus [McC85] such that activities of the agents can be precoded in the hypertext. The hypertext can then become an active participant in the building up of itself (so to say). For example, if it is known that in order to define the specifications of a software system, the agent (specifier) must give specifications A, B and C; and if the agent has given only specifications A and B; the hypertext can request the agent for the specification C. Or if the hypertext 'knows' that after writing out a memo agent A always sends it out to agent B; then it can automatically send a memo created by A to B without A having to explicitly do it. This has led us to the notion of Intelligent Software Hypertext Systems (I-SHYS⁴) which is the subject of our future work [Gar87,GS87b].

9 ACKNOWLEDGMENTS

Comments from Salah Bendifallah on earlier versions of the paper have improved the presentation. The work reported here has been supported by Hughes Radar Systems Group, El Segundo, under contract number KSR576195-SN8 and by the USC graduate school through the All-University-Pre-Doctoral Merit Fellowship.

References

- [AH84] Serge Abiteboul and Richard Hull. IFO: A Formal Semantic Database Model(Preliminary Report). In *Proc. ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, 1984.

⁴pronounced eye-shis

- [BH86] D. Bryce and R. Hull. SNAP: A Graphic-Based Schema Manager. In *Proc. of the Second Intl. Conf. on Data Engineering*, pages 151-164, February 1986.
- [Bra85] R. Brachman. On the Epistemological Status of Semantic Networks. In R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, pages 192-215, Morgan Kaufman Publishers, Inc., 95 First Street, Los Altos, CA 94022, 1985.
- [Bus45] V. Bush. As we may think. July 1945. *Atlantic Monthly*, no. 176, pp. 101-108.
- [Con87] Jeff Conklin. Hypertext: An Introduction and Survey. *Computer*, 20(9):17-41, September 1987. Also available as MCC Technical Report no. STP-356-86, Rev. 1.
- [DS86] N. Delisle and M. Schwartz. Neptune: a Hypertext System for CAD Applications. In *Proceedings of ACM SIGMOD '86*, pages 132-142, Washington, D.C., May 1986.
- [DS87] N. Delisle and M. Schwartz. Contexts: a Partitioning Concept for Hypertexts. In *Computer Supported Cooperative Work Conference*, 1987.
- [FS86] G. Foster and M. Stefik. Cognoter, theory and practice of a collaborative tool. In *Proceedings of the Computer Supported Cooperative Work Conference*, pages 7-15, 1986.
- [Gar87] P. Garg. Theoretical foundations for Intelligent Software Hypertext Systems. 1987. Computer Science Department, USC, In preparation.
- [GB84] I. P. Goldstein and D. G. Bobrow. A layered approach to software design. In D. R. Barstow, H. E. Shrobe, and E. Sandelwall, editors, *Interactive Programming Environments*, pages 387-413, McGraw-Hill Book Company, 1984.
- [GS87a] P. Garg and W. Scacchi. Software Hypertext Environments for Configured Software Descriptions. 1987. Submitted for publication, 1987.
- [GS87b] P. Garg and Walt Scacchi. On Designing Intelligent Hypertext Systems for Information Management in Software Engineering. 1987. To be presented at *Hypertext '87*.
- [McC85] J. McCarthy. Programs with Common Sense. In R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, pages 300-307, Morgan Kaufman Publishers, Inc., 95 First Street, Los Altos, CA 94022, 1985.
- [MS82] D. McLeod and J. M. Smith. Abstraction in Databases. In *Workshop on Data Abstraction, Databases, and Conceptual Modeling*, pages 1-7, June 1982.
- [Nel81] Ted Nelson. Literary machines. 1981. Available from author, P. O. Box 128, Swarthmore, PA 19081.

- [Sal86] G. Salton. Another look at Automatic Text-Retrieval Systems. *Communications of the ACM*, 29(7):648-656, July 1986.
- [SFG85] A. Sathi, M. Fox, and M. Greenberg. Theory of activity representation in project management. *IEEE PAMI*, September 1985. Special issue on principles of knowledge based systems.
- [Sha84] Mary Shaw. Abstraction Techniques in Modern Programming Languages. *IEEE Software*, 10-26, October 1984.
- [Sha85] D. Shasha. Netbook: A Data Model for text exploration. 1985. VLDB.
- [SS77] John M. Smith and Diane C. P. Smith. Database Abstractions: Aggregations and Generalizations. *ACM Transactions on Database Systems*, 2(2):105-133, June 1977.
- [Sto79] R. R. Stoll. *Set Theory and Logic*. Dover Publications, Inc., 1979.
- [Tic82] W. Tichy. Design, Implementation, and Evaluation of a Revision Control System. In *6th International Conference on Software Engineering*, pages 58-67, Tokyo, Japan, 1982.
- [Tri83] R. H. Trigg. *A Network-Based Approach to Text Handling for the Online Scientific Community*. PhD thesis, Maryland Artificial Intelligence Group, University of Maryland, November 1983.
- [UNI] *UNIX Users Manual*.

Manipulating Source Code in DynamicDesign

James Bigelow and Victor Riley

Computer Aided Software Engineering Division
Design Automation Group
Tektronix, Inc.
P.O. Box 4600, MS 94-480
Beaverton, OR 97075
jimbi@copper.tek.com
victorr@copper.tek.com

ABSTRACT

DynamicDesign is a Computer-Aided Software Engineering environment for the C language with a layered system architecture for modularity and versatility. DynamicDesign is composed of facilities to edit hypertext objects, maneuver thorough hypertext graphs, build a hypertext graph from a set of existing C source files, and browse source code, documents and system requirements. This paper discusses the DynamicDesign facilities that deal with the source code, sourceBrowser, and source tree builder utilities.

GraphBuild is a utility used to convert C source code into a hypertext source graph, based on the program's call tree. A data dictionary is constructed for the program that contains its local and global variables.

The source browser allows the user to traverse, view, and edit a source code tree. Additional facilities for understanding and maintaining the source code and its auxiliary documentation are provided by the browser.

INTRODUCTION

DynamicDesign is a Computer-Aided Software Engineering (CASE) Environment based on hypertext (Nels81). DynamicDesign stores C (Kern78) source code, requirements, and documentation in a hypertext database; the Hypertext Abstract Machine (Deli86) (HAM) developed by Tektronix, Inc. The HAM, a medium grained, entity-relationship-like data model, is a transaction-based hypertext database server that allows arbitrary structuring of information and keeps a complete version history of both information and structure. The HAM is used in the layered system architecture shown in Figure 1. DynamicDesign is one of the many possible applications that use the HAM by communicating through an Inter-Process Communication (IPC) mechanism. This method allows for extreme modularity and independence of software components and, as such, the HAM can be considered to be separate and distinct from DynamicDesign.

The following paragraphs introduce the hypertext objects in the HAM, then introduce DynamicDesign and one of its information structures. Additional paragraphs discuss two of the utilities built into DynamicDesign, graphBuild and sourceBrowse. In conclusion, possible directions for development of DynamicDesign and its utilities are indicated.

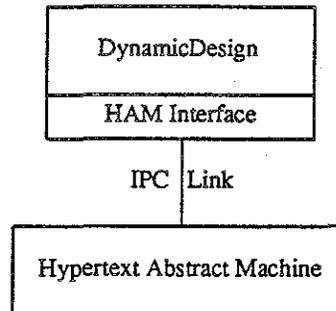


Figure 1. DynamicDesign system architecture.

HAM CONCEPTS AND TERMINOLOGY

The basic components of the HAM are nodes and links. Nodes provide a means to store data, and links provide the relationship between the data in different nodes. The HAM allows classification of nodes and links by providing the ability to associate an *attribute/value* pair with a node or link. For example, a node may have an attribute name that is given a value such as *Input Routine* to identify the contents as the source code for the input routine. Information is grouped into configurations by using *contexts*, disjointed collections of nodes, links, and contexts. Since nodes and links may be thought of as directed graphs, collections of nodes, links, and contexts are called a *graph*.

DYNAMICDESIGN—A HYPERTEXT CASE ENVIRONMENT

DynamicDesign is a hypertext CASE environment for the C programming language and has all of its project components in the HAM. These include:

- Specifications and requirements
- Design notes and documents
- Implementation notes
- Source and object code
- User documentation

Nodes are used to contain project components; links depict the relationships between the components; and contexts allow groupings such as components, configurations, versions and variations. Attributes are used to label the types of nodes, links, and contexts. Table 1 shows the possible values of three attributes.

Table 1. Three attributes and their possible values.

Ham Object	Attribute Name	Possible Values		
Node	projectComponent	requirement, design assumption, object,	spec, comment, symbolTable,	designNote, source, documentation,
Link	relation	leadsTo, calls, isdefinedBy,	comments, followsFrom, isdefinedAs	refersTo, implements,
Context	projectCategory	specifications, user doc, object code,	design doc, implement notes, symbol tables,	program doc, source code, product

In DynamicDesign, Nodes have an attribute, *projectComponent*, (which identifies the type of project component they contain). Links have an attribute, *relation* (which shows the type of relation the link provides). Contexts are identified with the attribute, *projectCategory*. For example, sequential information may be assoc-

iated by connecting two nodes with a link whose attribute, *relation* has a value of *leadsTo*.

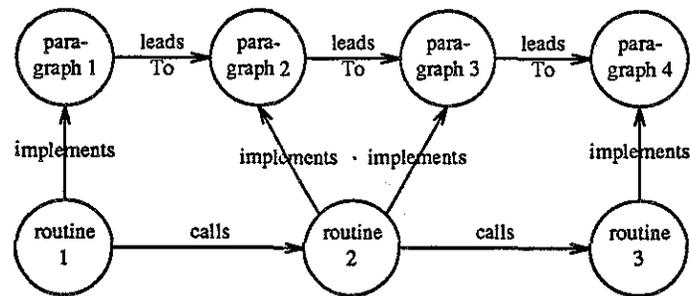


Figure 2. Linking specifications and source code with nodes and links.

In Figure 2, routine 1 calls routine 2 so they are both stored in nodes and connected by a link having the attribute value *calls*.

The relationship between a specification and the code that implements it can be shown with links. The node whose attribute, *projectComponent* value is *spec* contains a portion of the specification. A node with the attribute *projectComponent* having the value, *source* contains the code that implements the portions of the specification. These two nodes are related with a link that has an attribute, *relation* having the value, *implements*. Figure 2 show how routine 1 in the code relates to paragraph 1 in the specification.

DynamicDesign Project Category Interconnections

A project component is any peice of information or data associated with a project. Broad categories for data placement include:

- Specification and requirements
- Design, program, and user documentation
- Implementation notes
- Source code
- Object code
- Products

Within each of these categories are the actual documents, memos, papers, binaries, etc. that make up the project. By placing all the components of a project in hypertext, they are archived, recoverable, and available for use within other parts of the project.

Interconnections between project components exist even in a project that uses paper documents. However, there is also much duplication of both effort and documentation. Additionally, many opportunities to point out the relationships between components are missed because the effort involved is too great for the time permitted.

DynamicDesign has all the information concerning a project in its hypertext database. Contexts are used to group data into the categories mentioned above, and listed in Table 1 as possible values. In Figure 3 the lines used to connect the ellipses (representing contexts) show the direct interconnection and interrelationships between the data in the contexts. Due to the use of links, one piece of data can be present in several contexts. Therefore, a paragraph about a design may do triple duty; as a comment in the program documentation and as a paragraph in both the user and design documentation.

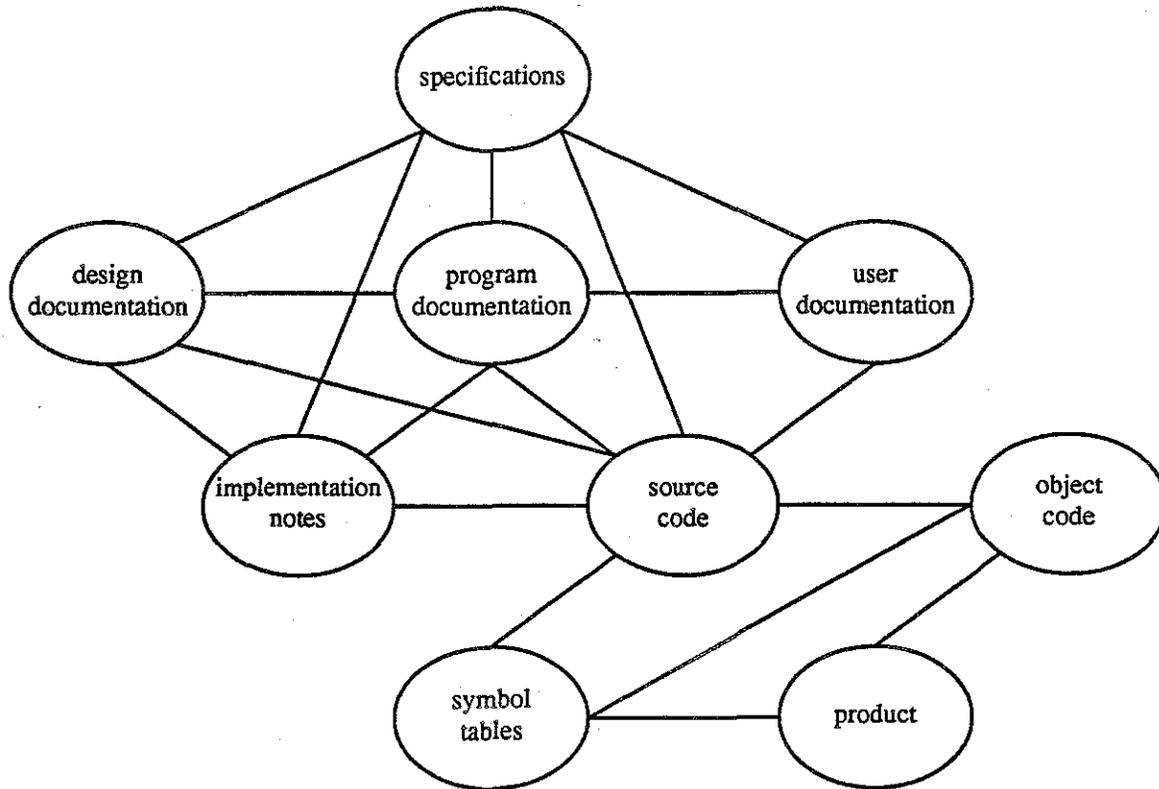


Figure 3. Interconnections of project categories in DynamicDesign.

DynamicDesign Utilities

The emphasis herein is on the *source code* context and the utilities that deal with the information structures in that content. The main information structure in the *source code* context is the source code tree, which can be created from existing code with the graphBuild facility. The tree can also be created or extended by using the SourceBrowser.

SOURCE CODE GRAPH STRUCTURE

Within the source code context, code is stored in the form of a tree. The format for the tree is based on the call tree for the set of C functions comprising the code, modified slightly by any C preprocessor commands in the source code. Variable dictionaries are created to hold all variable definitions and are linked into the source code tree by links from the point of reference to the point of definition.

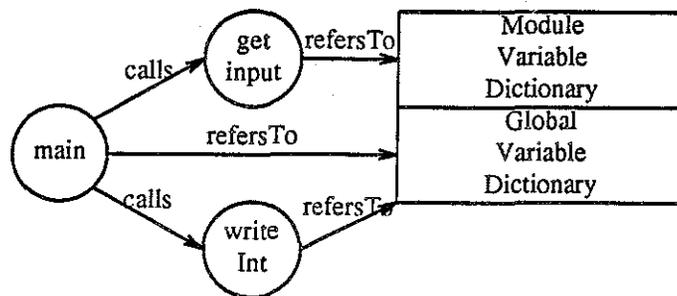


Figure 4. Source code tree built from the code in Figure 5.

Figure 4 Shows a source code graph created from the code in Figure 5 (which contains three functions: *main*, *getinput* and *writeInt*). Since *main* calls the other two functions, it is the root of the source code graph and the other two functions form the leaves.

```

#define BUFFER_SIZE 132      /* size of the I/O buffer */
#define INPUT 0             /* file descriptor for input */

int global_data = 0;        /* global data, defined in this file */
extern extern_data;        /* global data, defined elsewhere */
static int private_count = 0; /* private data, global to these functions */

extern char *getinput();
extern int writeInt();

main()
{
    int x, y;

    y = atoi(getinput());
    x = y + 1;
    writeInt(x);
    extern_data = 0;
}

char *
getinput()
{
    static char iobuf[BUFFER_SIZE];

    if((private_count = read(INPUT,&iobuf, BUFFER_SIZE)) < 0 ) {
        perror("example");
        exit(1);
    }

    return &iobuf;
}

int
writeInt(x)
    int x;
{
    fprintf(stdout,"%d", x);
    global_data++;
}

```

Figure 5. Three C functions with external data.

Modifying the Tree With Conditional Compilation Statements

The structure of the source code tree is modified by preprocessor conditional compilation statements in the code. These statements cause the code contained within the body of the statement to be placed in a separate node. For example in Figure 7 there are three nodes created from the code in Figure 6. The node labeled VAX code contains the VAX independent code from the body of the statement “`#ifdef VAX`”. The node labeled PDP11 code contains the PDP11-dependent code. The last node contains the statements that preceded and followed the two `ifdef` statements in the original code. Links are created as shown in Figure 7. This type of link has an attribute named for the preprocessor command (*ifdef*, *if*, or *ifndef*, for example). The attribute takes on the value of the constant expression or name that is the argument to the command.

```
code A
#ifdef VAX
    VAX-dependent code
#endif
#ifdef PDP11
    PDP11-dependent code
#endif
code B
```

Figure 6. Example usage of preprocessor conditional commands.

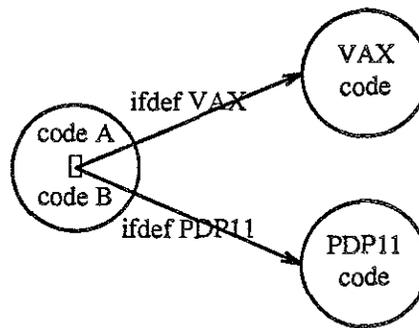


Figure 7. Example node subtree for conditionally compiled code.

Variable Dictionaries

Variable declarations for each scope, global, module, and function, are placed in variable dictionaries for each level and links are created from the points of reference back to the entry in the variable dictionary. Global variables such as *extern__data* and *global__data* in Figure 5 are placed into the global variable dictionary. Module variables, such as *private__count* in Figure 5 are placed in a module variable dictionary as in Figure 4. Local variables go into a local variable dictionary. A separate dictionary is created for each module and function.

Preprocessor macro definitions are also placed in a variable dictionary and links are created from the point of usage to the entry in the variable dictionary. Which dictionary the macro goes into depends on its point of definition. If the macro is defined at the start of a file, then it is placed in the module variable dictionary, otherwise, it is placed in the function's local variable dictionary. For example, in Figure 5 the preprocessor macros *BUFFER__SIZE* and *INPUT* would be placed into the variable dictionary for this module.

Determining What to Place in a Node

Nodes are atomic data units, so the issue of node contents is important. If a piece of data is referenced in more than one place (e.g., a section of text is in both the requirements and the comments for a section of code) the data should be in a node by itself. However, the application that uses hypertext (by determining the unit of incrementality used when processing the information) is the final arbitrator of how much should

be placed in one node. For example, in the case of an incremental compiler, which can recompile a changed procedure individually without recompiling the entire module that contains the procedure (Schw84)(Medi81), the unit of incrementality is a procedure. Other compilers may enforce a larger increment, such as a module.

GRAPHBUILD

An existing C program can be stored in the hypertext database by importing it with graphBuild, thereby creating a source code tree. Using this method, conventional editors such as *vi*, *ed*, or *emacs* are used to write the program, which is then presented to graphBuild for storage in the hypertext data base. Based on the syntactic and semantic rules of C, graphBuild reads a source file, and uses it to create a source code tree.

Mapping Files Into Hypertext

For the most part, files containing C source code fall into two categories, header files and source files. Obvious exceptions are *lex* and *yacc* files that are not covered in this version of DynamicDesign. How header and source files are mapped into the source code tree is discussed in the following paragraphs. In general, header files are used to create variable dictionaries and links with the attribute, *relation* with the values: *refersTo*, *isdefinedAs* and *isdefinedBy*. Source files are used to create the nodes of a source code tree.

Header Files

Header files provide a method to declare functions, variables, constants, and data types, so that other files can make use of those services declared in the header files (and defined elsewhere) by including the header files. Given the desire for separate compilation, header files and the keyword *extern* enable a function to use a variable that is not defined within its scope. The functionality of a header file and external references can be attained by an adroit use of links when the source code tree is built, thereby removing the need for either. By following the links from the variable reference to the variable dictionary, a compiler can discover from the variable dictionary entry exactly what type of variable is referenced and its scope. Because the information in a header file is either placed in a variable dictionary (in the case of defines) or represented by links (as with external variables) header files are not kept intact in the source code tree.

External Name Problem in C

An important issue with external names in C is ensuring consistency among declarations of the same external name in several files. It is a well-known deficiency in C that defining and referencing occurrences of external variables are difficult to distinguish. (Refer to *C: A Reference Manual* (Harb84) for a complete discussion of how compilers deal with the problem.) DynamicDesign simplifies the referencing problem by creating a link directly from a variable reference to its defining statement in the variable dictionary. By using a variable dictionary with links the problem of defining and referencing a variable is greatly simplified, since multiple ambiguous variable references don't exist.

Mapping Source Files

Source files contain the actual code for a program in the form of external and private data declarations and functions. GraphBuild, reads a source file and splits the file into variable dictionaries and source nodes. A source code file is considered to be a module. Since static variables declared in a file are local to the module and global to the functions within the module, they are placed in a module variable dictionary accessed only by the functions in the module.

Mapping Functions to Hypertext Nodes

GraphBuild attempts to place one function per node. The node has the attribute *projectComponent* that is given the value *source*. The node's attribute, *name* is given the value of the function's name. When a function is referenced, a link is created from the function reference to the node that contains the function and its definition. The link between the function call and function definition has an attribute, *relation*, that is given a value, *calls*. For example, in Figure 4 the three functions from Figure 5 are placed into three nodes (named for

the functions) and the nodes are connected by links with the attribute, *relation* containing the value *calls*. In this manner the graph of a program is its call tree.

One problem exists: how to handle calls to functions that have not yet been defined, and therefore do not have a node with which to link. If the function definition is encountered later on and a node created to hold it, then a link must be created from the previously encountered reference to the newly created node. GraphBuild keeps a list of functions encountered and their node names. When a function call is encountered one of two things happens: either a link is created from the position in the node where the function call is located, to the node holding the function definition or the location of the function call is stored so that when the function is defined, a link can be made to it. If the function definition is never encountered, as with library calls (e.g., *read*, *perror*, and *fprintf* in Figure 5) then no link will be created since there is nothing to link to. One possible solution is to create a dummy node named for the library the function is assumed to be defined in. This solution, however, is not recommended since it is prone to error. Rather, the function is left unlinked. A report of unlinked functions is available at the end of processing. This feature is user-selectable at run time, by means of a command line flag.

Mapping Data Declarations

There are two types of data declarations, private and global (i.e., *static* and *extern*) and both are placed in variable dictionary nodes. Global declarations are placed in a node whose attribute, *projectComponent* is *source* and its *name* is *global variable dictionary*. Declarations private to the module are placed in a module dictionary named for the source file. Function declarations are handled similarly; they are placed into a dictionary named for the function. For example, if the source code in Figure 5 were in the file, *example.c*, then the node containing the module dictionary would be named *example module dictionary*. Links are created from a point of reference to a declaration point. The link's attribute, *relation* is given a value, *refersTo*.

Figure 4 shows three source code nodes and two dictionaries, global and module. There are links with the attribute, *relation* containing the value, *refersTo* from the source nodes to the variable dictionary nodes, since each of the functions contained in the source nodes refers to a variable that is contained in the variable dictionary nodes. Variables are placed in global or module variable dictionary nodes by graphBuild when they are declared or defined external to any function (as shown at top of Figure 5).

Figure 8 is an expanded view of portions of two nodes from Figure 4 showing how a link with an attribute, *relation* containing the value, *refersTo* is attached at one end to a variable reference and at the other end to the definition of the variable.

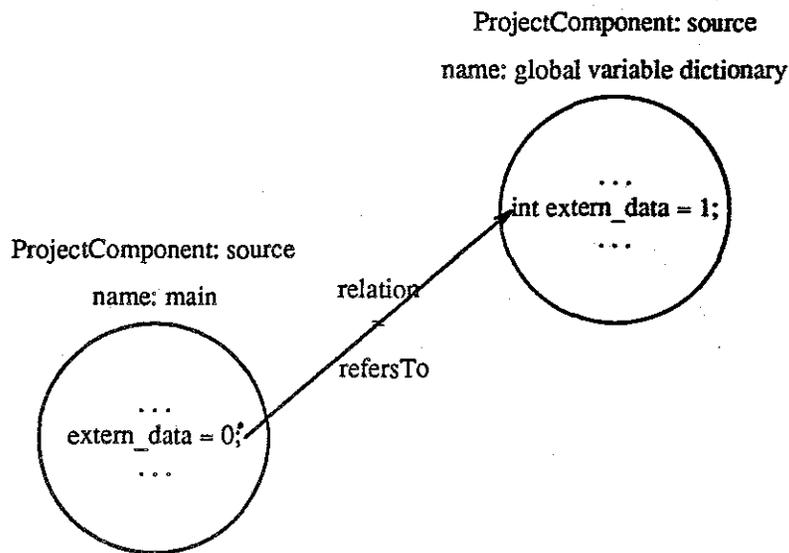


Figure 8. Connecting variable use with declaration by a link.

SOURCE CODE BROWSER

The source code browser is used to peruse source code trees created by GraphBuild or by the source code browser itself. It has the ability to traverse links to other sections of code, view or edit modules and functions, and answer questions about the code. Some of the questions it can answer are:

- What is the reason behind a routine?
- Who calls a routine?
- What is the purpose of a variable?
- Who uses a variable?

Browsing the Source Code Tree

In C source code there are several implicit connections between routines, modules, and accompanying documents. When viewing the source code it may be necessary to look at these connections simultaneously. Some example connections include: the comments explaining a specific piece of code, the definition of a variable within the code and the code itself, all references to a particular variable or function, the relationship between a function call and a function definition, and the header files that document the dependencies and interrelationships of global variables and functions for a module. All of these connections are explicitly represented by links in the source code tree rather than implicitly as in a source code listing.

One of the main functions of the browser is to traverse the source code tree. The browser may start at the root or at a point in a subtree. Since the amount of information within the source code tree is often overwhelming, the browser attempts to aid the viewer by pruning the traversal tree. The pruning is controlled by the specification of different attribute/value pairs on the links. For example, if the viewer only wished to see a version of the code tailored for the VAX, then all the conditional compilation links that did not have that value would be hidden from the viewer. In this same manner, if the viewer only wished to see source code s/he had written, then code that was not personally created would not be shown.

Requirements traceability can be of major interest when trying to prove that a system fulfills its specifications. Therefore, the source code browser also has the capability to follow links to other sections of the graph outside of the source code section. These other sections of the graph can be specifications, design documents, test specifications, documentation, or comments, which can then be viewed. Since DynamicDesign allows the creation of links from a specification to a design document such as a data flow diagram, as well as links from the design to the code implementing the design, and then from the code back to the specification, traceability graph cycles can be created. When the browser starts following a traceability cycle, it will choose the appropriate presentation capabilities for each type of node. For instance, if while viewing a source node, a designer wished to know what specification this code fulfilled, s/he could look for and select a traceability cycle link and follow it either back to the design context or back to the specification. The direction is unimportant, what is important is that by following the traceability cycle the designer will arrive back at the point of departure, thus proving that a particular requirement has been fulfilled. Conversely, the absence of traceability cycles shows that either that requirement has not been fulfilled or the correct linkages have not been created.

Creating Links From Data Dictionary to Variable Dictionary and Vice Versa

As source code and variables get added to the system, links need to be created between the definition in the variable dictionary and the data dictionary stored in design context. The source code browser does this automatically. As new code is accepted into the graph, the browser verifies that each newly defined variable is in the proper variable dictionary and then creates a link to the data dictionary. Conversely, the browser acts as a program design tool by allowing the association of modules with data flow mini-specifications. Then, based on the association, variables that correspond to the data dictionary entries for the mini-spec can be created automatically in the variable dictionary and linked back to the data dictionary. As existing code gets modified, consistency checks are made to verify that a deleted variable is removed from the proper variable dictionary and its links are removed.

Editing Source Code

One of the primary functions of the source code browser is to edit the source code being displayed. The browser's editor is able to edit not only the text of the source code but the links that form the relations between sections of code. Also, the browser is able to group pieces of code together into an editable module, based on attribute/value pairs. For instance, upon request, the browser would only present the code for a UNIX system for editing and the code for other systems would be hidden.

The editor/browser displays the links associated with the code under consideration. The manner in which the links are displayed is controlled by the browser and selectable by the viewer. Therefore, a viewer can decide how, and whether or not, the link is shown. Links can be represented by configurable icons or the link name. Furthermore, they can be hidden from the viewer by the browser upon request. As the code gets modified, the browser modifies the accompanying links to maintain consistency with the updated code.

Answering Questions

A useful ability of the source code browser is its ability to take advantage of the built-in variable and function cross reference in the source code tree provided by the links forming the tree. The browser can query the source code tree to answer questions about the source code being viewed. This capability allows the viewer to understand the code better by quickly answering questions about such things as the calling structure, definitions of functions, and variable usage, without the need to stop looking at the code being viewed. This replaces the five-finger method of code reading, where a person ends up with all five fingers sticking into interesting places in a listing while the other hand is used to flip back and forth between the places.

The following paragraphs consider some of the more interesting questions that the source code tree and the browser can answer.

Who or What Am I?

When a function is viewed sometimes it is not easy to understand just what its purpose is. It may be necessary to look at some documentation about the function. In DynamicDesign the user asks the function "What are you?" and all links that have the attribute, *relation* with the value, *comments* or *implements* are displayed. If there is only one, then the contents pointed to by the link are displayed. Otherwise, the user is presented with the link choices.

For example, while viewing routine 2 in Figure 2, and selecting the routine *name*, then asking "What are you?", the two links labeled *implements* would be displayed. The user then has the capability to traverse either or both links to get to the documentaiton.

Who Calls This Function?

When the browser is displaying a function, the user can ask to look at all other locations this function is called from by asking "Who calls you?". All links entering the function whose attribute, *relation* have the value, *calls* are then returned. The user then selects the location s/he is interested in and the segment of code containing the call is displayed.

If the user was viewing the function, *getinput()* in Figure 5 and asked "Who calls you?", the link named *calls* (in Figure 4) that connects *main* and *getinput* would be highlighted and the location of the call in *main()* would be displayed.

What is This Variable?

Selecting a variable and asking "What are you?", causes the browser to look in the variable dictionary, find the link back to the data dictionary and display the definition. If any links happen to exist whose attribute, *relation* have the value, *comments* they are also returned. The user can then select and view any of that information as well.

By selecting the variable, *global data* in Figure 5 and asking "What are you?", the link *refersTo* (in Figure 4) connecting *writeInt* and *global Dictionary* is highlighted. The link between the variable in the globalDictionary and the data dictionary is then followed and the definition in the data dictionary is displayed.

Who Uses This Variable?

By selecting a variable and asking "Who uses you?", a variable can be cross referenced. The information is obtained by following the link to the variable dictionary and then listing all links that point to that particular variable in the dictionary. The proper dictionary is selected based on the scope of the variable selected. For instance, by selecting the variable *private__count* in Figure 5 and asking "Who uses you?", the link *refersTo* (in Figure 4) connecting *getinput* and *moduleDictionary* is highlighted. Any other links coming into *private__count* in the *moduleDictionary* with the value, *refersTo* for the attribute, *relation* are also displayed. The user can then select the one(s) to view from that list.

Displaying the Code

How the code is displayed is very important to a good browser. If the code is presented in an unreadable fashion it will appear confusing and hard to understand to the user. Therefore it is important that the code be displayed in a clean, clear, and understandable format, one that is somewhat tailorable to the users desires.

Combining #ifdef Code

Generally, source code being developed for multiple environments has sections that are dependent on a specific host or operating system. Actual code may have *#ifdef*'s listed several times within it, but the source code browser will partially preprocess the code to present an un-*#ifdef*'ed view of the code to the user. This makes code reading much more understandable and the true function easier to comprehend. The user decides what the values for the *#ifdef*'d sections of code are, prior to the displaying of the code. This can be accomplished before or while browsing a section of code.

Viewing the Code

After the code has been preprocessed it is formatted using a style similar to the way the UNIX* program *indent* (Thom79) formats code. UNIX-like features make viewing large amounts of code much easier. Some of the tailoring features include:

- Indentation levels
- Alignment of comments
- Insertion of spaces within expressions
- Matching of braces
- Breaking up of declaration lists

These features are tailorable to the users desires by modifying options in the command. This way each user can have the source code taylored to their own desires, with a separate format for each user. If the user does not wish to format the code, or if the code has already been formatted these features can be disabled.

CONCLUSION AND SUMMARY

The preceding paragraphs have introduced a hypertext-based CASE environment, *DynamicDesign*, discussed its information structures for storing C source code, and introduced utilities for manipulating the information. It is contended that hypertext provides a viable data model and offers great promise for meeting the needs of CASE.

Future directions for work on *DynamicDesign*, or any hypertext CASE environment should include work on a consistent, expandable method for deciding what should be placed in a node. This method could solve problems arising from situations where more than one application makes use of the same type of node each with different demands on the granularity. Expanded support for compilers, linkers, and other conventional tools would enhance the environment's usefulness, as well as expanding support for *lex* and *yacc* files and even other languages such as C++, Ada, or Modula2.

Work on building systems using hypertext can focus on how to automate the creation of sequential and relational links. Sequential links show that one node logically follows another and relational links show that

* UNIX is a trademark of Bell Laboratories.

two nodes are logically related, but not sequentially. By automating the linking process based on the way a node is used, the users is spared repetitive linking. However, there may always be the need for a means of creating a link at the user's command that would point out a relationship the system has missed and only a user can see.

A hypertext weakness that has not been addressed is how to represent fine-grained information. One solution is to create a partnership between hypertext and relational databases. A relational database can hold fine-grained information such as definition-use links in an incremental compiler's symbol tables. A relationally complete query language extends the functionality of hypertext to provide even more capabilities.

REFERENCES

- T.H. Nelson, *Literary Machines*, T.H. Nelson, Swarthmore, PA., 1981.
- B.W. Kernighan and D.M. Ritchie, *The C Programming Language*, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.
- N.M. Delisle and M.D. Schwartz, "Neptune: a Hypertext System for CAD Applications," *Proc. ACM SIGMOD '86*, pp. 132-143, May 1986.
- M.D. Schwartz, N.M. Delisle, and V.S. Begwani, "Incremental Compilation in Magpie," *Proceedings of the SIGPLAN '84 Symposium on Compiler Construction*, vol. 19, no. 6, pp. 122-131, June 1984.
- R. Medina-Mora and P.H. Feiler, "An Incremental Programming Environment," *IEEE Transactions on Software Engineering*, vol. SE-7, no. 5, pp. 472-482, Sep. 1981.
- S.P. Harbison and G.L. Steele, *C: A Reference Manual*, Prentice-Hall, Englewood Cliffs, New Jersey, 1984. Section 4.8
- K. Thompson and D.M. Ritchie, *UNIX Programmer's Manual*, Bell Laboratories, Murray Hills, NJ. Seventh Edition, 1979.

On Designing Intelligent Hypertext Systems for Information Management in Software Engineering

Pankaj K. Garg and Walt Scacchi

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0782
garg or scacchi@cse.usc.edu

ABSTRACT

Information management in large scale software engineering is a challenging problem. Hypertext systems are best suited for this purpose because of the diversity in information types that is permitted in the nodes of a hypertext. The integration of a hypertext system with software engineering tools results in a software hypertext system. We describe the design of such a system called DIF. Based on our experiences in using DIF, we recognized the need and the potential for developing a hypertext system that could utilize knowledge about its users and their software tasks and products. Such a system might then be able to act as an active participant in the software process, rather than being just a passive, albeit useful storage facility. As such, we define an Intelligent Software Hypertext System (I-SHYS¹) as a software hypertext system which is knowledgeable about its environment and can use such knowledge to assist in the software process. This knowledge is partly embedded in the design of an I-SHYS (in terms of the 'agents' that I-SHYS supports) and partly defined during the use of I-SHYS (in terms of tasks that agents perform). We present a framework for defining and organizing this knowledge, describe potential uses of such knowledge, identify limits of our approach, and suggest methods for circumventing them.

1 INTRODUCTION

Hypertext systems are useful for information management in large scale software engineering because of the diverse types of information permitted in hypertext nodes [BEH*87]. If all the information related to a software system is stored in the same hypertext then we call it a *Software Hypertext*. A Software Hypertext System which supports the management of a software hypertext can, in conjunction with various software engineering tools, provide an Integrated Software Engineering Environment [Hen86]. The advantage of this combination (hypertext system + software engineering tools) is that one can exploit the facilities of tools for automated processing of information, while facilities of the hypertext system can be used for storing and retrieving information. We have

¹Pronounced eye-shis

designed and implemented such a software life cycle Documents Integration Facility (DIF) in the System Factory at USC [GS88].

Our experiments with DIF led us to the notion of an Intelligent Software Hypertext System (I-SHYS). DIF is a passive system with little explicit knowledge about its surrounding environment. In I-SHYS we wish to design an *active* hypertext system, which participates and assists in the process of engineering large software systems throughout their life cycle². As such, a *software hypertext environment* consists of:

1. The software engineering tools that process documentable software descriptions stored as a hypertext; and
2. The software engineering tasks that people perform through it.

If we encode knowledge about the hypertext environment into the hypertext system, such that the system can actively assist in the activities of its environment, then we get an Intelligent Hypertext System. For an Intelligent Software Hypertext System (I-SHYS), we have identified three perspectives of such knowledge:

1. Knowledge of the capabilities and uses of software tools that the environment provides;
2. Knowledge about the roles people play in the software process;
3. Knowledge about the tasks and actions people perform at different stages in the software process.

Before we can formalize and encode such knowledge in an I-SHYS, we need to understand the software process from these perspectives. This paper describes our current understanding in this regard. In Section 2 we present our understanding of a Software Hypertext System by giving an overview of DIF. Issues about interfacing tools with a hypertext system are discussed in this section. Section 3 describes the software process by categorizing the roles of *agents* in the process, describing the tasks that they perform, and detailing how their tasks can be broken down into actions performed on a software hypertext. It also discusses the attributes which can be used to categorize interactions. Finally in Section 4 we summarize the discussion and suggest future work in this direction.

2 DIF: A SOFTWARE LIFE CYCLE DOCUMENTS INTEGRATION FACILITY

DIF is a software hypertext system which helps integrate and manage the documents produced and used throughout the life cycle of software projects. It was designed for use in the System Factory, an experimental laboratory created at USC to study the development, use, and maintenance of large

²For simplicity, we use the term "software process" as shorthand for the process of engineering large software systems throughout their life cycle.

software systems [Sca86]. It has been used in the System Factory to support the software process for more than a dozen software systems, resulting in the creation of some 40Mbytes of software hypertext³.

DIF provides an interface to a hypertext-based information storage structure and to a structured documentation process. A hypertext of software information is built by teams of software engineers over eight life cycle activities. DIF provides several features which allow users to view information related to a software system in an integrated manner within and across projects. The document nodes are internally organized as a tree of Unix directories and files (see Figure 1). Users of DIF enter software process information into pre-defined (but redefinable) nodes of a software hypertext that are internally treated as files. Subsequently, all routine file management (e.g., creation of directories and naming of files for related document nodes) is handled by DIF. In total, the capabilities of DIF described below enable software engineers to document their software process in ways that support: (a) analysis of the consistency and completeness of formalized document nodes, (b) intra- and inter-document traceability, (c) formatting and display, (d) indexed or query-driven browsing, (e) documentation standards, (f) multi-version documents with/without sharable annotations, (g) reusable software component catalogs, and (h) online software inspections and walkthroughs.

The integration of software hypertext nodes as files and directories is invisible to the user of DIF. For instance, when entering information for the "operational requirements" of the system, the user does not have to create the file for storing the text associated with the operational requirements; this chore is automatically handled by DIF. The user need only be concerned with creating or manipulating software descriptions without being concerned about how they are stored or where. This provides an object oriented environment of persistent software system descriptions rather than simply a loose collection of files and directories. This is reflective of the 'Next Generation Operating System' envisioned by Balzer [Bal86].

DIF also allows software engineers in the System Factory to develop parts of documents in parallel without worrying about concurrent access or integration issues. Hence person A could be writing the operational requirements of the target system while person B is writing the non-operational requirements. The individual efforts are automatically merged in the same hypertext.

2.1 System Factory Structure

The organizational structure supported by DIF in the System Factory is shown in Figure 2. In the System Factory the project manager prescribes what needs to be described in each document. In turn, these prescriptions implicitly represent the software process in effect within the structure of the software hypertext. There are also potentially several projects in the factory at the same time.

³We have also utilized its facilities to "publish" hard-copy, laser-printed renditions of this encyclopedic software documentation, where one complete printing produced a series of listings about 4 feet tall after binding.

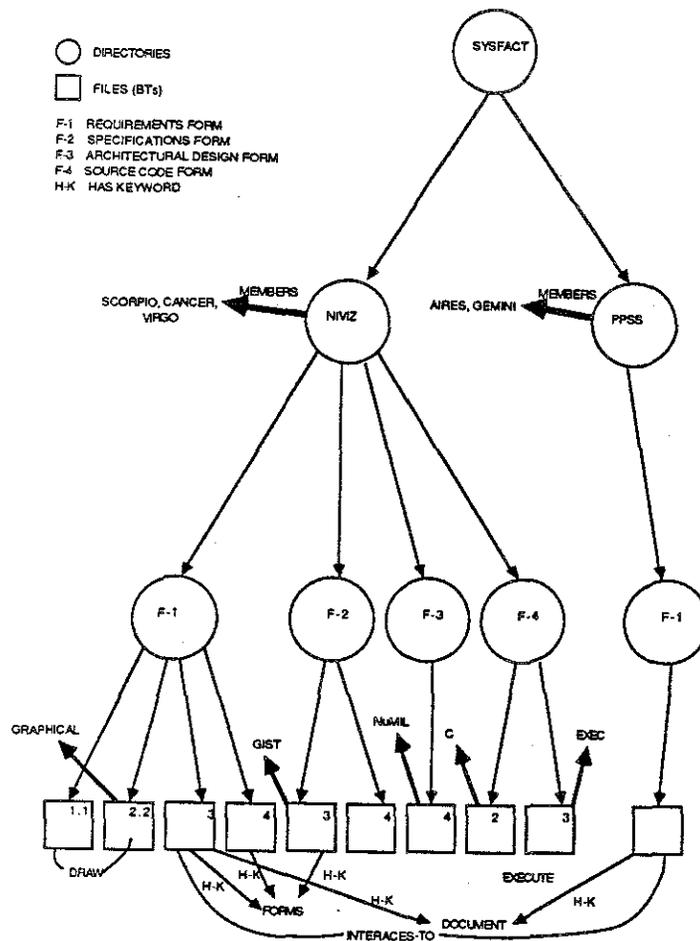


Figure 1: Hypertext of Software Documents in DIF

Several software engineers work on each project, and all projects can be constrained to follow the same software (documentation) process.

DIF supports two roles for users: a Super User role and a General User role. The two roles can be compared to the database administrator and end user respectively. In the Super User role, users define the factory structure (what projects are in the factory and who is responsible for each), and the structure of the documents (what needs to be documented). In the General User role, users exploit DIF to create, modify, and browse through the information hypertext. There are two levels at which a general user can operate: (1) at the information level, and (2) at the structure-of-the-information level. The rest of this section describes the functionalities of DIF.

2.2 Forms and Basic Templates

A Super User defines the *forms* and the *Basic Templates* (BTs) in the factory. One of the concerns of the System Factory was to ensure that all the projects have the same (standardized) structure of

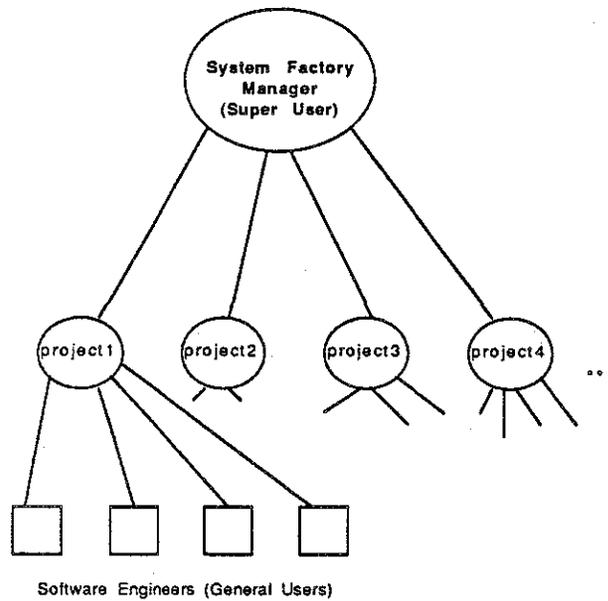


Figure 2: System Factory Structure

documents. Thus each document is defined as a *form*. A form is a tree structured organization of *Basic Templates (BTs)* to be filled with information. A collection of related forms then implicitly models the software process, or tasks therein. For example, in the System Factory, the functional specifications form is shown in Table 1. Such forms provide a way of specifying the process tasks to be followed by team members working on related projects.

Section Number	Section Heading
Overview and summary of functional specification	1.0
Informal narrative specification	2.0
Narrative specification	2.1
Functional diagrams	2.2
Type lattice	2.2.1
Functional network diagrams	2.2.2
State-transition diagrams	2.2.3
Formal specification	3.0
Gist Processor Results	3.1
BT Number	BT Heading

Table 1: Functional Specifications Form

The Super User defines each form only once and all the projects inherit that form. When defining BTs the super user also defines the nature of information that needs to be given in each BT. This entails providing the attribute of the BT as being one of:

- Narrative Text
- NuMIL Structural Specifications
- C Source Code
- Graphical Diagrams
- Gist Functional Specifications
- Executable Object Code

This tells DIF which editor (e.g., a Gist language-directed editor) to use for a BT and which software tools to process the information of the BT.

2.3 Project Information

Super Users provide project information which consists of a list of projects and the software engineers working on them. This information enables DIF to check the read/write privileges of the users. General users are allowed to create, modify, and revise information related to their projects only. There is no restriction on reading the information of any project. Super Users have read/write privileges for all information.

2.4 Information Level

Facilities are provided in DIF for a user to enter, modify, and use the information required by the forms as dictated by the super user. Language-directed emacs-like editors are provided for all the formal languages that are used in the System Factory (e.g., Gist, NuMIL, and C [Sca86]). The general user enters the information in BTs without worrying about the files that need to be created. DIF automatically generates a unique filename depending on the project and the BT.

Whole forms can be stored in the revision control system, RCS [Tic82] for backup and incremental revisions. Functions supported by DIF (through RCS) include:

1. Checking in of a form. The user can ask DIF to check in a form into RCS.
2. Checking out of a form. The user can check out a whole form from RCS.

Options such as retrieving revisions through user defined identifiers, cut-off dates, etc. are available through the interface. This is an example of 'interface transparency' that DIF provides for the tools that it interfaces to (section 2.6).

Request to process the information in a BT through a software tool can be made within DIF itself without entering the operating system. For example, if a BT contains C code, the user can request its compilation. Some such requests are handled through editor interfaces [Sta84], some are built into DIF (those which require the service of a System Factory tool as opposed to a Unix tool).

Interfaces to nroff/troff, spell, etc. provide the user with a text processing environment akin to the documenters workbench [Dwb], while interfaces to mail, rn, and talk, support asynchronous and synchronous communications among project participants. Other tools available in the System Factory (e.g., application generators, computer animation environment) [Sca86] can also be interfaced with DIF in a straightforward manner.

2.5 Structure-of-Information Level

The structure-of-information⁴ level allows the general user to navigate through the hypertext of information that is stored in DIF.

The user can navigate through the information in a project in the following ways:

1. **Links:** The user (super or general) can define links between BTs. The links are similar to the links allowed by most hypertext systems. Hence a person browsing the hypertext can add *annotations* to the currently visited BT, add links to other BTs, etc. For instance, the operational requirements of the system can be linked up to the code/modules that support the capability. This, therefore, provides a mechanism for maintaining consistency and traceability across documents. There are two key differences between the definition of links prevalent in hypertext literature [Con87] and those in DIF:

- Links define relationships between existing nodes. Except for annotation links, links are precluded from creating new nodes.
- Links are allowed to be *operational links*. This readily supports the cases where executable descriptions need to be linked to the source code. For example, a C code BT can be linked to the object code BT that represents that code. Such a link is defined in DIF as an operational link. Visiting that link results in the execution of the linked BT. Arbitrary shell procedure attachments to links will be supported in future versions of DIF.

2. **Keywords:** For each BT the user can define keywords which describe the semantics of the information contained in that BT. The decision to allow for user defined keywords rather than providing automatically generated keywords was based on the results of studies reported by researchers in information science [Sal86].

DIF stores the keywords associated with BTs in an Ingres relation. This allows the user of DIF to use the querying facilities of Ingres for navigating through documents using keywords. For example, the user can look for all BTs (within and across projects) which have a particular keyword, list the keywords of a BT, search for BTs which have keywords satisfying a pattern, etc. Standard functionalities such as form-based querying are provided by DIF for users not trained in Quel [Que].

An interesting feature of DIF is that it allows the reader of documents also to create keywords of their own. This allows new personnel in a project team to quickly tune the documents to their needs.

⁴This is different from a 'database schema'. In a schema the structure does not change with the information, whereas here the structure is dependent on the currently defined links.

3. **Forms and Configurations:** A Form is a tree-structured organization of BTs. This provides the user with a convenient way of viewing the documents relating to each software process activity.

To fully utilize the potential of the hypertext of information in DIF, the user can define his/her own *configuration* of BTs. A configuration is similar to a form, except that it is not enforced on all projects but is associated with the individual user who is browsing the documents.

Configurations can be defined, not unlike forms, by defining the constituent BTs. Configurations can also be defined on the basis of the trail which a user has followed while browsing through the information hypertext. Configurations are mainly used as a mechanism for printing hardcopy documents, much like the *path* facility suggested by Trigg [Tri83].

The user information space is restricted to the project currently being 'visited' by the user. To use the information of another project the user has to explicitly visit that project. This means that the user cannot use the information level commands on the information of projects other than the one that is being visited. Structure-level information is available regardless of which project is being visited. This is done to avoid the risk of the user getting lost in the information space [Con87]. As an additional guidance, the current project and BT are displayed in the main menu.

2.6 DIF+Tools

The basic idea in DIF is to provide a system such that all the life cycle activities can be done through DIF itself. In this sense, DIF can be considered a software engineering environment [Hen86]. With the progress of the target software system through the various life cycle activities, DIF provides a uniform interface to access the appropriate tools as necessary, e.g., a functional specification analyzer or NuMIL Processor. In the interfaces to these tools, it supports the notion of 'interface transparency' i.e., DIF provides unobtrusive use of the tool that it interfaces to, while providing mechanisms that the tool itself lacks.

Figure 3 shows the organization of DIF with respect to the other tools in the System Factory.

The basic set of tools comprises [Sca86]:

1. A Gist Specification Analyzer and Simulator which facilitates the development and use of the formal functional specifications of software (sub)systems under development [BGW82,Sca85];
2. A Module Interconnection and Interface Definition processor that supports the design and evolution of multi-version system (module) configurations described in the NuMIL language [NS87b,NS87a];
3. An EMACS-like language-directed editing environment which helps in the construction and revision of structured documents and system description languages such as Gist, NuMIL and C; and

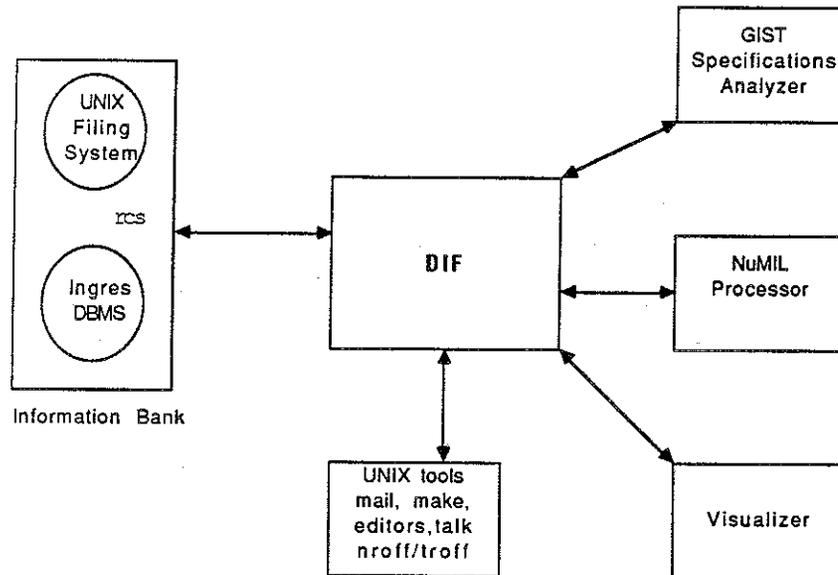


Figure 3: Organization of DIF

4. A system visualizer which graphically presents system configurations expressed in NuMIL [GS87];
5. Unix tools such as Rcs, Make, Spell, Nroff/Troff, Talk and Mail. The interface to the mailing system helps individuals to coordinate their activities by structured messages consisting of BTs.

DIF supports the notion of *Extensibility* in its interfaces to tools. It is simple to add new tools to the environment.

Tool Options: Most tools allow for the input of "switches" to tune the behavior of the tool for the application at hand. For example, while compiling a C program on Unix, the user can give a 'g' option which informs the compiler that it should generate symbol table information to be used by the symbolic debugger. An interesting way to view the switches is to consider them as means of informing the tool of some aspect of the environment in which the processing of information is taking place. For example, in the above case, the compiler is being informed that the code being compiled is an experimental one and is currently being debugged. Such information is required not only for purposes of the switch but is also useful in other places such as reporting the status of the project. The hypertext system can store this information generically, and then 'automatically' generate appropriate switches for the compiler. As another example, consider the 'c' option of the C

compiler on the Unix system, which informs the compiler that the code in the file is part of a bigger system and the compiler should not load the file using a loader. This information is required at another level, viz. the architectural design of the system. Hence, the information need be given to the hypertext system only once and it can use it at multiple places. This is currently not provided by DIF and is planned for I-SHYS.

2.7 Summary

In this section we have presented an overview of a Hypertext System to manage software life cycle documents and suggested interesting ways in which smart interfaces to software tools can be provided in such a system. Our next concern is that the process model considered by DIF (described to it by way of Forms and BTs) is at a level of granularity too high to provide anything but a passive repository and processing environment that organizes document products developed through a software process. In the following section, we describe ways of breaking down the process model into finer grained actions such that they can be better supported by an I-SHYS. The treatment in this paper is semi-formal; a formal presentation is given in [Gar87b].

3 AGENT-TASK-PRODUCT PERSPECTIVE OF THE SOFTWARE PROCESS

DIF is a passive software hypertext system in that it waits for users to enter software descriptions into its network of linked nodes. However, it is incapable of interacting with its users to help elicit emerging software descriptions, nor can it explicitly represent and utilize knowledge of what roles its users play when performing different software process tasks. Instead, we would like to have a system that not only subsumes the capabilities of DIF, but does so in ways that it can eventually become *an active agent that participates in the software process*. Thus, such a hypertext system should be knowledgeable about its users, their tasks and products, and be able to ask/answer questions, simulate software process tasks, and to reason about and explain its behavior.

We first seek capture and represent knowledge about the 'agents' participating in the software process. Agents are people or intelligent systems that play well defined *roles*. An individual in the software process can play the role of more than one agent. For example, a person who has designed, implemented, and used a personal database management system is at once a designer, implementor, manager, and a user. We consider the following four categories of agents [GLB*83]:

1. *Users*: Agents who will use the target software system (end users) and/or agents who want the system developed (clients).
2. *Managers*: Agents who are responsible for software project management. There are two roles of managers that we consider, agents who coordinate the activities of other agents in the process (Process Manager, PM), and agents who analyze and evaluate the status of the process (Quality-Assurance Manager, QAM).

3. *Developers*: Agents who develop the system. Their tasks involve design and innovation, where they transform the Users' requirements into a working target system. Developer agents play roles including Analysts, Specifiers, Designers, Implementors, Testers, and Integrators.
4. *Maintainers*: Software systems are frequently modified to take care of changes in the requirements or discovery of errors in system development. Maintainers take care of such modifications, but may not have participated as the system's initial developers.

Notice that DIF considered only two categories of agents: (1) Super User (equivalent to the Manager and Client in the new framework), and (2) General User (equivalent to the End User, Maintainer, and Developers in the new framework).

The advantage of this categorization is that it helps us conceptualize the functions of agents in the process: each function viewed as a task associated with one kind of agent. This does not preclude the existence of interactions between agents, which become necessary when agents have intertwined tasks.

The next step is to identify the software process tasks of agents and to see how an I-SHYS can assist them. For this purpose we define:

- *Meta-tasks* which define the nature, configuration, and possible orderings of other tasks in the software process. Much of this is based on our study of the software process from an organizational perspective [Sca84].
- *Product tasks* that agents in the software process perform, borrowing from our empirical analyses of software engineering [Sca81,BS87] and from definitions prevalent in the software engineering literature (e.g., see [Boe81]).
- *Actions* that need to be performed in order to fulfill the commitments of tasks.
- *Primitive actions* which can be performed on a hypertext system.

The distinction between an action and a task is that a task represents the action of an agent which entails the fulfillment of some commitment [McD85]. Hence, creation of a sorting program is an action; but creation of a sorting program by an agent, A for sorting the files on a tape, T is a task.

The following sections elaborate this categorization.

3.1 Meta-Tasks

Meta-tasks are all performed by an agent assuming the role of a 'Manager'. We reiterate that an agent does not necessarily have a one-to-one mapping with the individuals in the process. The following meta-tasks have been identified [Sca84, p. 46]:

- Planning(PM⁵): Detailing the tasks that need to be performed in the software process.
- Organizing(PM): Allocating resources to the agents.
- Staffing(PM): Assigning agents to tasks.
- Directing(PM): Giving help to other agents in terms of what decisions to make in situations of uncertain or incomplete information.
- Coordinating(PM): Getting groups of people to work collaboratively.
- Scheduling(PM): Assigning time constraints on tasks.
- Validating(QAM): Confirming that the running system meets the requirements of the Users.
- Verifying(QAM): Confirming the consistency, completeness, and integrity of evolving software descriptions.

Each meta-task has a document or processable description (e.g., plans, schedules, work breakdown structure) associated with it. Accordingly, each meta-task needs to encode a different source of knowledge about the software process, and their relationship to other meta-tasks and product tasks in order to be capable of providing active participation.

3.2 Product Tasks

Product related tasks are carried out by agents assuming the role of either a Developer, a Maintainer, or a User. Users pose requirements for the system and use the system. Maintainers change the system based on emerging requirements of Users. Developers build the system using requirements posed by the Users and guidelines suggested by PMs. Several tasks can be performed in this regard:

1. Requirements Definition (Clients)
2. Requirements Analysis (Analysts)
3. Functional Specifications (Specifier)
4. Architectural Design (Designer)
5. Detailed Design (Designer)
6. Implementation (Implementor)
7. System Integration (Integrator)
8. System Delivery (Integrator)

⁵ Agent category responsible for the task

9. System Use (End User)
10. Fixing bugs in the system (Maintainer)
11. Creating revisions of the system (Maintainer)
12. Enhancing the system (Maintainer)
13. Creating new versions of the system (Maintainer)
14. System bug discoveries and reporting (End User)
15. Testing (Tester)
16. Requesting Enhancements on the system (End User).

The definition of these tasks can be found in any book on software engineering (e.g., see [Boe81]). As before, each product task has an associated document or software description that is produced upon its completion. In turn, forms with computational methods attached are needed in order to help elicit the pertinent software information in order to evaluate consistency, completeness, and integrity of intra- and inter-tasks products.

The tasks viewed from this viewpoint result in a task diagram as shown in Figure 4. The figure shows the distribution of tasks of the three agents: Maintainers, Developers, and Users, with an example of a task of PM and that of a specifier elaborated. For a detailed account of the process related tasks, the reader is referred to [SBB*86]. The intertwining of tasks of multiple agents leads to requirements of interaction and collaboration, as discussed in section 3.5.

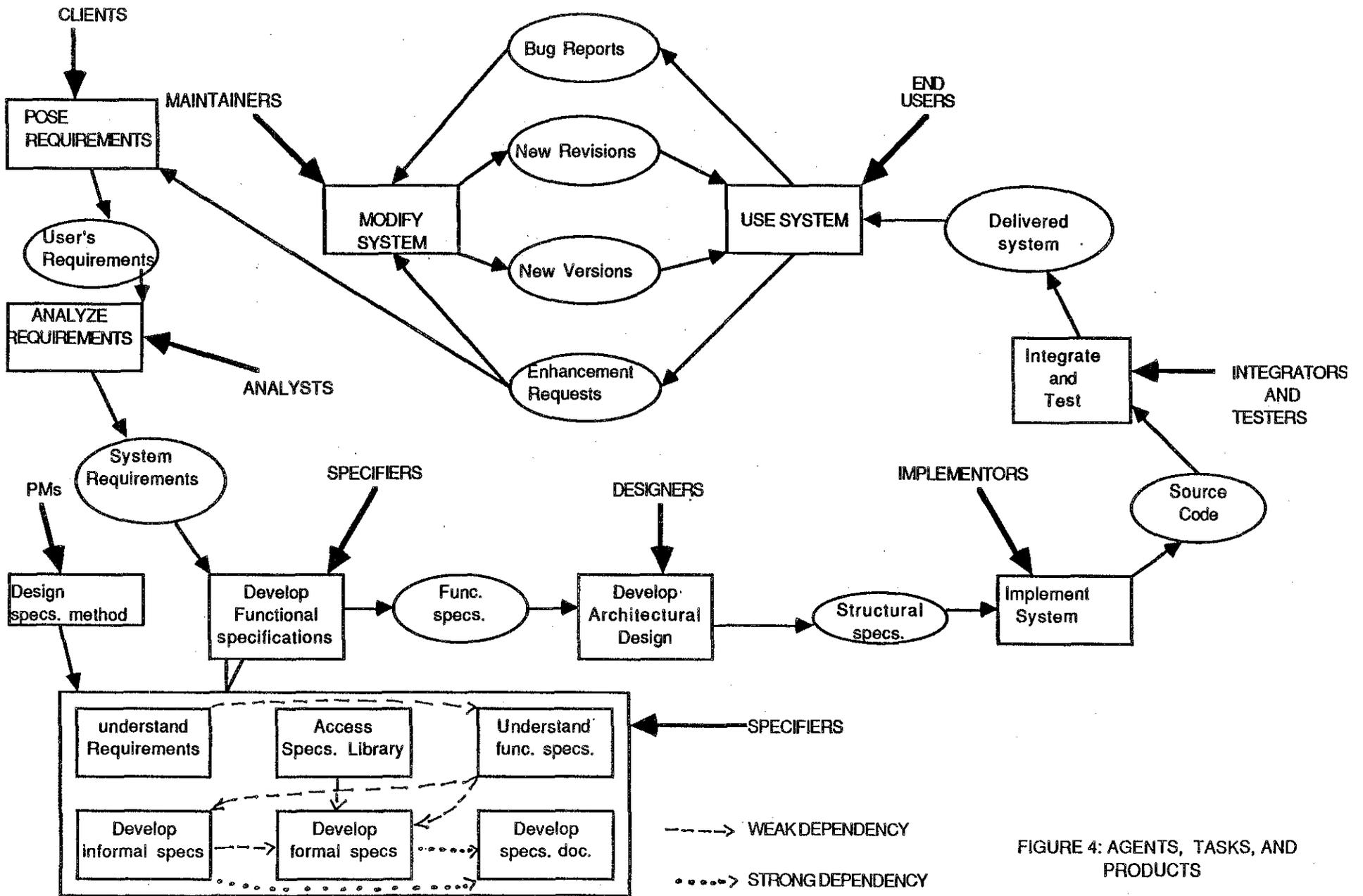


FIGURE 4: AGENTS, TASKS, AND PRODUCTS

3.3 Actions

Actions are obtained from the definition of tasks by detailing the steps that need to be done by the agent in order to effectively carry out the task.

As an example consider the Develop functional specifications task in Figure 4. This task can be broken down into several actions [Ben87]:

- Understand Users' requirement
- Understand functional specifications of the system
- Develop informal specifications of the system
- Access library of exemplar specifications
- Develop formal specifications of the System
- Develop a specifications document for the system.

Each action can be carried out by the Specifier agent. The dependency of actions (as shown in Figure 4) indicates that: (1) (Weak Dependency) If action A is dependent on action B, then action A cannot be effectively started unless there is a successful completion of action B. This means that A can possibly be started before the end of action B, but the results will not be as *effective* as when A is started after B is finished. For example, a specifier can start developing the formal specifications before looking at a library of exemplar specifications. (2) (Strong Dependency) If A is dependent on B, then A cannot be started unless B has finished. For example, a specifier cannot develop the specifications document unless the formal and the informal specifications have been written.

These tasks can be related to the BTs defined for the Functional Specifications Form (Table 1) as follows:

- Understand users' requirement leads to the informal narrative specification BT 2.0;
- Understand functional specifications and Develop informal specifications of the system lead to the Narrative Specifications BT 2.1;
- Access library of exemplar specifications and Develop formal specifications of the system lead to BTs 2.2 through 3.0; and
- Develop specifications document for the system leads to the Functional Specifications Form, with all its BTs integrated.

The support provided by DIF in integrating the results of multiple tasks is now clearer. But DIF deals with only the results of tasks, and not how tasks are performed. Hence it supports the tasks at a very coarse level of granularity, ignoring the actions that compose the task. In an I-SHYS our focus

is to develop definitions of tasks in terms of the actions that compose them, such that the actions are simple enough to be automatically supported. As an aside, we must caution the reader that we are not suggesting to reduce the creative aspects of the process. Rather, we are suggesting to reduce its non-creative aspects which encumber creative aspects. This is in tune with the philosophy being pursued by Delisle and Schwartz in their definition of an Idea Processor [DS87], by the Colab project at XEROX, PARC [FS86] and project Nick at MCC [BCE*87].

To see how this could be achieved, consider the action Understand Users's requirements, which is part of the task of Develop functional specifications. It can be broken into finer grained actions as follows:

- View the requirements document;
- Create notes of the key points of the requirements;
- If something is unclear, communicate with the Users to understand it better;
- If still unclear then discuss it with fellow Developers, PMs, or Users;
- Organize notes about the understanding.

This leads to the definition of Primitive Actions on a hypertext as follows.

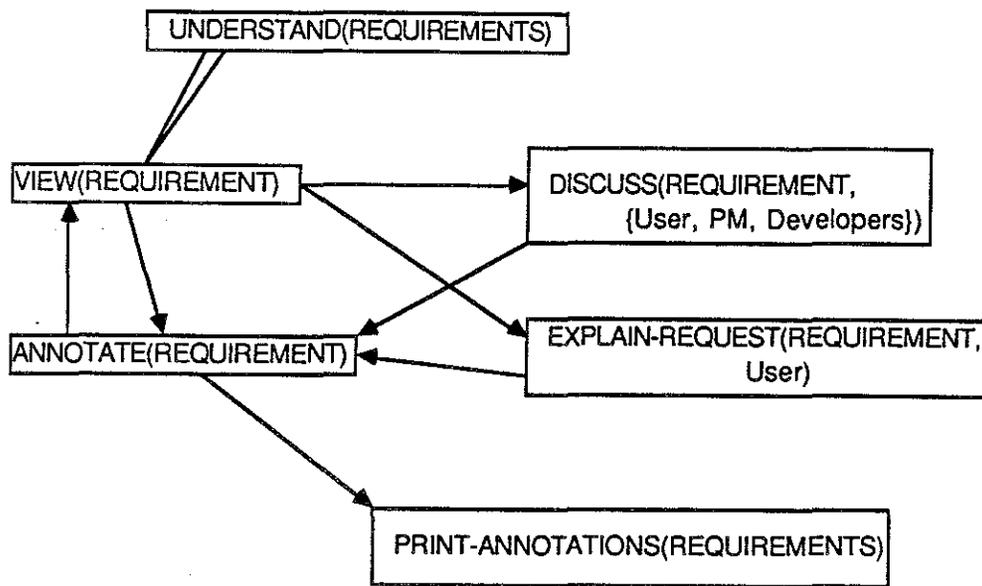
3.4 Primitive Actions

Consider the hypertext to consist of objects and relationships between the objects⁶[Gar87a]. (In DIF these were BTs and links between BTs.) From the example in the previous subsection, we can identify the following primitive actions:

- VIEW(REQUIREMENT) — 'look at' an information object containing a User's requirement.
- ANNOTATE(REQUIREMENT) — attach notes of 'understanding' to an object X.
- EXPLAIN-REQUEST(REQUIREMENT, U) — request an explanation of a requirement from a User agent.
- DISCUSS(REQUIREMENT, {A₁, A₂, ... A_n}) — discuss a requirement with other agents of the process.
- PRINT-ANNOTATIONS(REQUIREMENTS) — organize and print the annotations attached to the requirements.

The action Understand User's Requirements can be understood in terms of these primitive actions as the following diagram shows:

⁶The objects are the nodes in the hypertext graph, and the relationships are the edges.



Consider the diagram as a non-deterministic flowchart. The double lined arrows show the relation 'elaboration' [SFG85] which means that the action ordering at the head of the arrow can be considered as the elaboration of the action at the tail of the arrow.

The primitive actions developed in this manner can be generalized by replacing REQUIREMENT with an arbitrary object X. For example, we can have an action UNDERSTAND(X) which is composed of VIEW(X), DISCUSS(X, α), EXPLAIN-REQUEST(X, ζ), ANNOTATE(X), and PRINT-ANNOTATIONS(X), where α is a set of agents, and ζ is the agent responsible for the creation of X.

3.5 Discussion

There are several ways that this approach can result in knowledge about software process agents, tasks, and products being encoded in an I-SHYS. In this subsection we discuss these issues.

Establishing Context: The break up of tasks into actions and of actions into primitive actions results in the establishment of *context* [DS87] of the hypertext. For example, the Understand Users Requirements establishes the context in which one of the five primitive actions can be taking place. An I-SHYS can therefore automatically prune out the objects and relationships that are not pertinent to one of these actions. Since the action of Understand Users' Requirements is carried out over a period of time, this can reduce the information overload on the agent, as the agent does not have to worry about objects and relationships which are not relevant to his/her immediate action. In current practices context establishment would have to be done manually by the users of hypertext

and can lead to much semantic complexity [DS87].

Semantically Rich Commands: The analysis naturally leads to definitions of primitive actions which can be converted into semantically rich commands. For example, it is possible to provide commands such as DISCUSS(X,A) which informs the hypertext to start a discussion about object X with agent A. Coordination tools such as suggested by Winograd [Win87] and Sluzier and Cashman [SC84] can then be integrated with the hypertext system, to 'intelligently' support discussions between agents. Similarly there can be a command to PRINT-ANNOTATIONS(X) which instructs the hypertext system to organize and print the annotations attached to an object. The encoding of what kind of organization is required can be provided along the lines of *configurations* suggested in section 2.5.

Task Coordination: A break down of tasks into the actions that constitute them, allows the hypertext system to be informed about the agents and what tasks they are expected to perform. This allows the hypertext system to perform task coordination, by which it can trigger demons in case actions which were supposed to be done are not done, or if performing an action requires a response from an agent. For example, if agent A_1 performs the action EXPLAIN-REQUEST(REQUIREMENT, U_1), then agent U_1 is expected to respond with either an explanation, denial, or an alternative suggestion [Win87]. If U_1 does not respond within a certain time framework then a demon can be fired which suggests to A_1 to either: abandon the request, send a complain to a manager, or ask someone else. There are a whole set of such communication acts which arise in the course of a software process and which need automated support for their coordination [Ked83].

Interactions: As we saw in the previous examples, there are actions which explicitly require the intervention of other agents. For example, the action EXPLAIN-REQUEST(X,A) requests the explanation of object X by agent A. To support such interactions we have identified the following attributes of interactions which can possibly influence the nature of support required by them:

Agents: The categories and number of agents that interact in completing product or meta-tasks.

Communicated-object: descriptive object which is the focus of interaction.

Signal: medium or language used for the interaction. The various forms of signal of interaction that we consider are: (1) natural language, (2) figures and graphs, (3) formal language (which can be subjected to automated semantic analysis), and (4) semi-formal language (language which has a mixture of natural language, figures, graphs, and formal language).

Time: time period over which the interaction is carried out. We distinguish (1) duration of interactions as being either long or short, or (2) whether the interaction is synchronous or asynchronous.

Space: the geographical relationship of agents interacting. This can be either: (1) distributed, implying that the interacting agents are geographically dispersed and therefore cannot engage in face to face discussions; or (2) local, meaning that the interacting agents are in the same geographical location and therefore can engage in face to face discussion.

Each of the interaction attributes has an influence on the functionalities demanded of the hypertext system. Most of the functionalities can be trivially mapped from the value of the attribute. For example, a interaction carried out synchronously requires a computer mediated real time conferencing support. Similarly a interaction carried out asynchronously requires some structured mail support [MGL*87]. The advantage of encoding this information into an I-SHYS is that in a complex process such as the software process, the demands on the functionalities of the hypertext system can change depending on the stage at which the process is. If the hypertext system 'knows' about this, it can change itself to meet the requirements imposed by the stage of the process.

4 SUMMARY AND FUTURE WORK

In this paper we have described a Software Hypertext System DIF. We have then shown how notions of a Software Hypertext System can be extended to the concept of an Intelligent Software Hypertext System (I-SHYS), by studying the environment of I-SHYS from a tools, tasks, and interaction framework. Implementation of parts of I-SHYS are in progress using KnowledgeCraft on the TI Explorer LISP machine. We are also building a theoretical model of I-SHYS using an extension of the theory of Situation Calculus [McD85, Gar87b]

There is a limitation in the approach presented in this paper for the construction of I-SHYS. The limitation has been suggested by the results of empirical studies of the software process [Sca84, BS87]. An assumption made in systems such as I-SHYS is that the software process is a *closed system* wherein the tools used in the process, the tasks performed in the process, and the interaction patterns of the process, can be defined *a priori*. Empirical investigations of the process, however, indicate that the software process is an *open system* [Hew86] where the tasks and interactions are determined by complex relationships between the process, the agents in the process, the setting in which the process is carried out, and the product which is being developed. These concepts are illustrated by categorizing the tasks and interactions as:

- Primary Tasks versus Articulation Tasks [BS87, Gas86], and
- Routine Interaction versus Non-routine interaction [Gas86].

Primary tasks are the tasks prescribed by the policies of the process. Articulation tasks are a non-deterministic sequence of actions that must be performed in order to effectively carry out a primary task. Articulation tasks emerge when primary tasks descriptions are incomplete to handle

unexpected circumstances such as when breakdowns, foul-ups, or resource bottlenecks suddenly emerge. An example will make this clearer⁷:

Suppose that the primary task of an agent is to print a report detailing an understanding of the functional specifications of a system. Suppose that the agent has developed a complete understanding, and in order to print the report the use of a text-editor and a text-formatter are required. Also suppose that the agent is not familiar with any of the text editors available. Then in order to accomplish the primary task (that of printing a report) the agent has to accomplish the articulation task of understanding a text editor.

In parallel to the notions of primary tasks and articulation tasks, we distinguish between two kinds of interactions:

1. Routine interactions, and
2. Non-routine interaction.

A routine interaction occurs as part of the 'ideal' software process and not because of situations peculiar to particular settings. Programming, considered as an interaction between implementor, editor, and compiler is a routine interaction. Non-routine interactions emerge when primary software process tasks depart from expectation or plan. For example, negotiations for competing resources by multiple agents may be a non-routine interaction. Ideally a programmer should be able to follow Wirth's step-wise refinement paradigm and develop efficient 'literate programs' [Knu84]. But in real world situations, progress is not readily smooth and thus the programmer might have to try non-routine interactions (e.g., discuss issues with associates) to develop the final program.

Systems such as I-SHYS can provide support for Primary tasks and Routine interactions. For supporting Articulation tasks and Non-routine interaction, the approach suggested in this paper fails. As an alternative, one can explore the use of Case Based Reasoning. If we analyze articulation tasks and non-routine interaction, we find that these can be successfully accomplished by agents who have had similar experiences before. Case Based Reasoning provides a framework by which experiences of several agents can be encoded in a knowledge based system and the system can provide suggestions to the agents by examining the actions of agents who were faced with similar situations before. Encoding such knowledge requires a set of empirical studies which acquire knowledge about articulation tasks and non-routine interaction in various projects. It requires a framework in which to encode such knowledge such that similarity-reasoning can be performed on present situations to the situations in past cases. Our group has started efforts in this direction [Ben87, Jaz87].

⁷This example is the essence of a more detailed example presented in [BS87]

5 ACKNOWLEDGMENTS

The ideas in this paper have benefited from discussions with Salah Bendifallah and Abdulaziz Jazzar. Comments from Salah Bendifallah on earlier versions of the paper have improved the presentation. The work reported here has been supported through research grants or contracts with AT&T Information Systems, TRW Systems Engineering Design and Development, Hughes Radar Systems Group under contract number KSR576195-SN8, IBM through the Socrates project at USC, and MDA 903-81-C-0331 from DARPA to USC/ISI. In addition, the first author acknowledges the support provided by the USC graduate school through the All-University-Pre-Doctoral Merit Fellowship.

References

- [Bal86] R. Balzer. Living in the Next Generation Operating System. In *Proceedings of the 10th World Computer Conference*, IFIP Congress, Dublin, Ireland, September 1986.
- [BCE*87] M. Begeman, P. Cook, C. Ellis, M. Graf, G. Rein, and T. Smith. PROJECT NICK: Meetings Augmentation and Analysis. In *Computer Supported Cooperative Work Conference*, 1987.
- [BEH*87] T. Biggerstaff, C. Ellis, F. Halasz, C. Kellog, C. Richter, and D. Webster. *Information Management Challenges in the Software Design Process*. Technical Report STP-039-87, MCC, Software Technology Program, January 1987.
- [Ben87] S. Bendifallah. *Understanding Software Specifications Work: An Empirical Analysis*. PhD thesis, Computer Science Department, University of Southern California, December 1987. Forthcoming.
- [BGW82] R. Balzer, N. Goldman, and D. Wile. Operational specification as the basis for rapid prototyping. *ACM SIGSOFT Software Engineering Notes*, 7(5):3-16, 1982.
- [Boe81] B. Boehm. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.
- [BS87] S. Bendifallah and W. Scacchi. Understanding Software Maintenance Work: An Empirical Analysis. *IEEE Transactions on Software Engineering*, SE-13, March 1987.
- [Con87] Jeff Conklin. Hypertext: An Introduction and Survey. *Computer*, 20(9):17-41, September 1987. Also available as MCC Technical Report no. STP-356-86, Rev. 1.
- [DS87] N. Delisle and M. Schwartz. Contexts: a Partitioning Concept for Hypertexts. In *Computer Supported Cooperative Work Conference*, 1987.
- [Dwb] *Documenters Work Bench*. UNIX Documentation.

- [FS86] G. Foster and M. Stefik. Cognoter, theory and practice of a collaborative tool. In *Proceedings of the Computer Supported Cooperative Work Conference*, pages 7-15, 1986.
- [Gar87a] P. Garg. Abstraction Mechanisms in Hypertext. 1987. To be presented at *Hypertext '87*.
- [Gar87b] P. Garg. Theoretical foundations for Intelligent Software Hypertext Systems. 1987. Computer Science Department, USC, In preparation.
- [Gas86] L. Gasser. The integration of computing and routine work. *ACM Transactions on Office Information Systems*, 4(3):205-225, July 1986.
- [GLB*83] C. Green, D. Luckam, R. Balzer, T. Cheatham, and C. Rich. *Report on a Knowledge Based Software Assistant*. Technical Report KES.U.83.2, Kestrel Institute, June 1983.
- [GS87] P. Garg and W. Scacchi. Software Hypertext Environments for Configured Software Descriptions. 1987. Submitted for publication, 1987.
- [GS88] P. Garg and W. Scacchi. A Hypertext System for Software Life Cycle Documents. January 1988. To Appear in *Proceedings of the 21st Hawaii International Conference on System Sciences*.
- [Hen86] P. Henderson, editor. *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*. ACM, SIGPLAN notices, vol. 22, no. 1, Palo Alto, California, Dec. 9-11, 1986.
- [Hew86] C. Hewitt. Offices are open systems. *ACM Transactions on Office Information Systems*, 4(3):271-287, July 1986.
- [Jaz87] A. Jazzar. *A Model for Managing Software Documents*. PhD thesis, University of Southern California, December 1987. Computer Science Department, Forthcoming.
- [Ked83] B. I. Kedzierski. *Knowledge-Based Communication and Management Support in a System Development Environment*. PhD thesis, University of Southwestern Louisiana, November 1983. Available as Kestrel Institute Technical Report KES.U.83.3 Kestrel Institute Palo Alto California.
- [Knu84] D. E. Knuth. Literate Programming. *The Computer Journal*, 27(2):97-111, 1984.
- [McD85] D. McDermott. Reasoning about plans. In J. R. Hobbs and R. C. Moore, editors, *Formal Theories of the Common Sense World*, pages 269-317, Ablex Publishing Corporation, Norwood, New Jersey, 1985.
- [MGL*87] T. W. Malone, K. R. Grant, K. Lai, R. Rao, and D. Rosenblitt. Semi-structured messages are surprisingly useful for computer-supported coordination. In *Computer Supported Cooperative Work Conference*, 1987.

- [NS87a] K. Naryanaswamy and W. Scacchi. Database Foundation to support Software Systems Evolution. *The Journal of Systems and Software*, 7(1):37-49, March 1987.
- [NS87b] K. Naryanaswamy and W. Scacchi. Maintaining Configurations of Evolving Software Systems. *IEEE Transactions on Software Engineering*, SE-13(3):324-334, March 1987.
- [Que] *Ingres Reference Manual*. Unix 4.2BSD Documentation.
- [Sal86] G. Salton. Another look at Automatic Text-Retrieval Systems. *Communications of the ACM*, 29(7):648-656, July 1986.
- [SBB*86] W. Scacchi, S. Bendifallah, A. Bloch, S. Choi, P. Garg, J. Skeer, and M. J. Turner. Modelling the Software Process: A Knowledge Based Approach. 1986. System Factory Unpublished Manuscript.
- [SC84] S. Sluzier and P. M. Cashman. XCP: An Experimental Tool for Supporting Office Procedures. In *IEEE 1984 Proceedings of the First International Conference on Office Automation*, IEEE Computer Society, Silver Spring MD, 1984.
- [Sca81] W. Scacchi. *The Process of Innovation in Computing: A Study of the Social Dynamics of Computing*. PhD thesis, Department of Computer Science, University of California, Irvine, 1981.
- [Sca84] W. Scacchi. Managing software engineering projects: a social analysis. *IEEE Trans. Software Eng.*, SE-10(1):49-59, January 1984.
- [Sca85] W. Scacchi. Software specification engineering: an approach to the construction of evolving system descriptions. 1985. Research Report USC/Information Sciences Institute Marina Del Rey CA in preparation.
- [Sca86] W. Scacchi. A Software Engineering Environment for the System Factory Project. In *Proc. 19th Hawaii Intern. Conf. System Sciences*, 1986.
- [SFG85] A. Sathi, M. Fox, and M. Greenberg. Theory of activity representation in project management. *IEEE PAMI*, September 1985. Special issue on principles of knowledge based systems.
- [Sta84] R. Stallman. EMACS: The Extensible, Customizable, Self-Documenting Display Editor. In D. R. Barstow, H. E. Shrobe, and E. Sandelwall, editors, *Interactive Programming Environments*, pages 300-325, McGraw-Hill Book Company, 1984.
- [Tic82] W. Tichy. Design, Implementation, and Evaluation of a Revision Control System. In *6th International Conference on Software Engineering*, pages 58-67, Tokyo, Japan, 1982.

- [Tri83] R. H. Trigg. *A Network-Based Approach to Text Handling for the Online Scientific Community*. PhD thesis, Maryland Artificial Intelligence Group, University of Maryland, November 1983.
- [Win87] T. Winograd. A language/action perspective on the design of cooperative work. In *Computer Supported Cooperative Work Conference*, 1987.

AUTHOR'S

INDEX

A U T H O R ' S I N D E X

Akscyn, Robert	<i>Knowledge Systems Incorporated</i>	1
Anderson, Kenneth T	<i>Brown University</i>	67
Bader, Gall	<i>Brown University</i>	67
Beeman, William O.	<i>Brown University</i>	67
Begeman, Michael L.	<i>MCC</i>	247
Bell, Brigham	<i>The University of Colorado</i>	215
Bigelow, James	<i>Tektronix, Incorporated</i>	397
Bolter, Jay David	<i>University of North Carolina</i>	41
Brown, P.J.	<i>Workstations Limited and the University of Canterbury</i>	33
Campbell, Brad	<i>Tektronix, Incorporated</i>	21
Charney, Davida	<i>The Pennsylvania State University</i>	109
Chimera, Rick	<i>Carnegie Mellon University</i>	121
Collier, George H.	<i>Bell Communications Research</i>	269
Conklin, Jeff	<i>MCC</i>	247
Crane, Gregory	<i>Harvard University</i>	51
Ferguson, Gordon J.	<i>University of North Carolina</i>	195
Fox, Barbara	<i>The University of Colorado</i>	215
Frisse, M.D., Mark E	<i>Washington University School of Medicine</i>	57
Garg, Pankaj K.	<i>University of Southern California</i>	375,409
Gillespie, Terilyn	<i>Carnegie Mellon University</i>	121
Gomez, Louis	<i>Bell Communications Research</i>	175
Goodman, Joseph M.	<i>Tektronix, Incorporated</i>	21
Halasz, Frank G	<i>MCC</i>	345
Hammwöhner, Rainer	<i>The University of Constance</i>	155
Irish, Peggy M.	<i>Xerox Palo Alto Research Center</i>	89
Jones III, Henry W.	<i>Morris, Manning, and Martin</i>	367

Joyce, Michael	<i>University of North Carolina</i>	41
Kaufer, David	<i>Carnegie Mellon University</i>	121
King, Roger	<i>The University of Colorado</i>	215
Landauer, Thomas K.	<i>Bell Communications Research</i>	175
Landow, George P.	<i>Brown University</i>	331
Larkin, Jame	<i>Brown University</i>	67
Lewis, Clayton	<i>The University of Colorado</i>	215
Marshall, Catherine C.	<i>Xerox Special Information Systems</i>	253
McClard, Anne P.	<i>Brown University</i>	67
McCracken, Donald	<i>Knowledge Systems Incorporated</i>	1
McQuillan, Patrick	<i>Brown University</i>	67
Neuwirth, Christine	<i>Carnegie Mellon University</i>	121
Oren, Tim	<i>Apple Computer, Incorporated</i>	291
Raskin, Jef	<i>Information Appliance</i>	325
Raymond, Darrell, R.	<i>University of Waterloo</i>	143
Remde, Joel R.	<i>Bell Communications Research</i>	175
Riley, Victor	<i>Tektronix, Incorporated</i>	397
Scacchi, Walt	<i>University of Southern California</i>	409
Shields, Mark	<i>Brown University</i>	67
Shneiderman, Ben	<i>The University of Maryland</i>	189
Smith, John B.	<i>University of North Carolina</i>	195
Smolensky, Paul	<i>The University of Colorado</i>	215
Thiel, Ulrich	<i>The University of Constance</i>	155
Tompa, Frank Wm.	<i>University of Waterloo</i>	143
Trigg, Randall H.	<i>Xerox Palo Alto Research Center</i>	89
Walker, Janet H.	<i>Symbolics Incorporated</i>	307
Weiss, Stephen F.	<i>University of North Carolina</i>	195
Yoder, Elise	<i>Knowledge Systems Incorporated</i>	1