Triton, a Hypertext System for the Macintosh

by

Gary B. Terusaki
CS 501 - Master's Project
Oregon State University

Major Professor - Dr. Bill Bregar

March, 1992

Abstract

Until now, most hypertext systems have been implemented on large scale computers. With improvements in microprocessors and development of graphical user interfaces, personal computers can run systems that previously needed the power of a mainframe. The low costs and widespread use of PCs will enable many people to use hypertext technology. Triton is a hypertext system that runs on the Apple Macintosh computer. It incorporates ideas from the Neptune hypertext system of Tektronix and from the Dexter hypertext model. This paper presents the Dexter model, a survey of hypertext systems, the implementation of Triton, and a list of features for future hypertext systems.

Acknowledgements

I would like to thank Dr. Bill Bregar of Tektronix for his help and guidance during this project. I would also like to thank Mayer Schwartz of Tektronix for his advice and Bruce Cohen of Tektronix for providing me with the Neptune source code. I am also deeply indebted to Malek Daaboul of Sequent Computer Systems for his support and encouragement during the course of my study.

Contents

Dedication

To my parents.

1. Introduction

This paper documents the development of Triton, a hypertext system for the Macintosh computer. Before discussing Triton, a brief discussion of hypertext and its history is in order.

The concept of hypertext has existed since at least 1945. That year Vannevar Bush, President Franklin Roosevelt's science adviser, wrote about a theoretical machine called memex [3]. Memex was to be a repository of stored knowledge with the ability to cross-reference related items. Its storage media consisted of books, papers, photographs, and microfilm. Navigation aids would allow the user to display pieces of information and retrieve any items that were cross-referenced. Memex, as envisioned by Bush, was a mechanical device that would access information in a quick manner. However, it was not until the availability of electronic computers that his ideas became feasible to implement.

With the availability of large scale mainframe computers, researchers developed a number of hypertext systems. While early systems such as Intermedia, NLS, and Xanadu allow storage and retrieval of text, each has a different purpose. Intermedia [5], which started as a research project at Brown University, is a general writing hypertext system. Douglas Englebart first proposed NLS [5] as a tool to help people expand their thinking processes with a computer. McDonnell Douglas now markets NLS as a tool that supports electronic mail and software engineering artifacts. Xanadu [5] is a library system for general literature that was created by Ted Nelson. Autodesk is currently developing a commercial implementation of Xanadu for Sun workstations [8].

Because of the diversity of features and the different areas these systems addressed, one of the questions that early researchers had to deal with was, "What is hypertext?" They generally agreed that hypertext systems met the following criteria [5, 26]:

1. Hypertext systems are implemented on computers.

2. Information is stored in files on magnetic disk.

3. Documents can have *link indicators* embedded in their text indicating that another related document is available for inspection.

4. The system has commands that allow a user to browse various items in an arbitrary manner. This feature is sometimes called *non-linear text*.

5. A *browser window* displays a picture of how the documents are linked together if the hardware supports a graphics terminal.

Figure 1-1 shows an example of such a system. There are two documents and a browser window. The document labelled "Quicksort Algorithm" has a highlighted link, "LINK 18," that points to an entry in the document labelled "References." In the browser window, a line between the boxes "Quicksort Algorithm" and "References" represents this link. By using navigational tools in the system, a user can select a link and request that the related document be displayed on the screen.

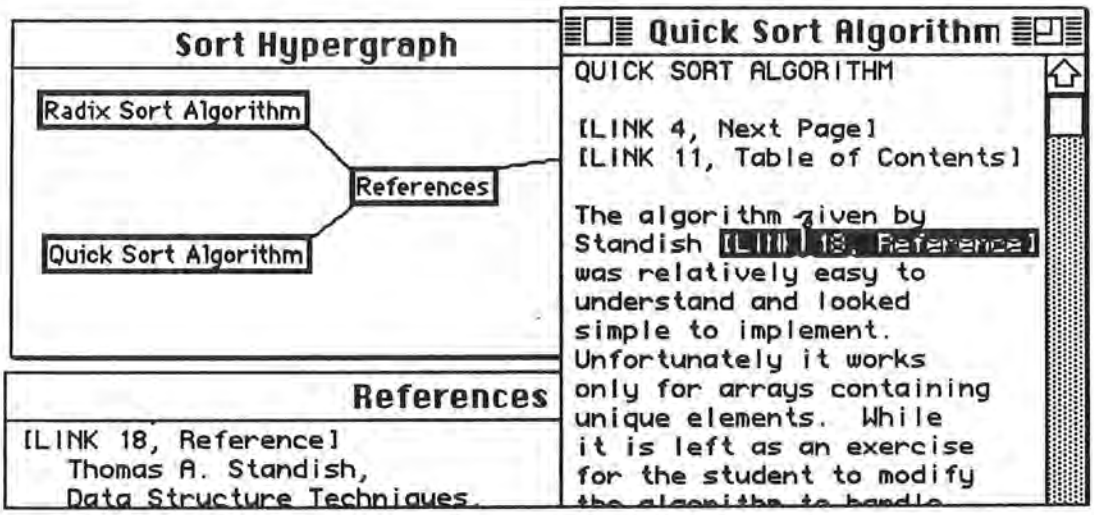

Figure 1-1. Example of a hypertext system.

Modern technology has spawned devices that were unavailable or unthought of at the time of these first systems. Advances in microprocessor technology, high resolution color graphical displays, and software engineering have enabled hypertext systems to be

developed for smaller computers. In addition, technologies that were previously thought of as being outside the scope of computers are contributing to the development of hypertext. Contemporary hypertext systems are starting to incorporate sound, animation, and video [22, 26]. The term *hypermedia* has been created to describe these technologies. Some writers make a distinction between hypermedia and hypertext. In this paper the scope of the term hypertext includes these new technologies.

The Triton hypertext system runs on the Macintosh computer and is based on the Neptune system developed at Tektronix [6]. Neptune creates and maintains documentation for software engineering projects. Its data base mangement functions run on a VAX computer. The user manipulates documents from a workstation that runs a user interface written in Smalltalk. Neptune requires a large investment in hardware and sophisticated operating systems. One of the goals of the Triton project was to see if hypertext systems developed for a mainframe environment could be ported to a small computer.

The example in figure 1-1 was created using Triton. Triton uses the standard Macintosh user interface to display nodes and browsers in regular Macintosh windows. These windows are similar to the displays produced by Smalltalk. After selecting link markers with the mouse, the user can use menu commands to navigate to other nodes.

While Triton satisifies the five basic criteria for early hypertext systems, advances in technology and user sophistication continue to elevate standards. The Dexter hypertext model [14] was created as a standard for future hypertext systems. This paper examines the Dexter model and uses it to compare Triton with several existing hypertext systems. Other sections of this paper discuss the features and implementation of Triton in detail. This paper concludes with a list of considerations for future hypertext systems. Documentation for running Triton is included in Appendix A.

2. The Dexter Hypertext Model

In October 1988 a number of hypertext researchers gathered at the Dexter Inn in New Hampshire for the first of several workshops. The result of their efforts is the Dexter hypertext model [14]. The main purpose of this model is to define a set of features that hypertext systems should have and to provide a standard of interoperability and data exchange between two different hypertext systems. The model also defines a standard lexicon of terms for the components and features of these systems. By using the Dexter model as a standard, an attempt can be made to see if a system generally conforms to what can be called a hypertext system. The model can also be used to compare and contrast different hypertext systems.

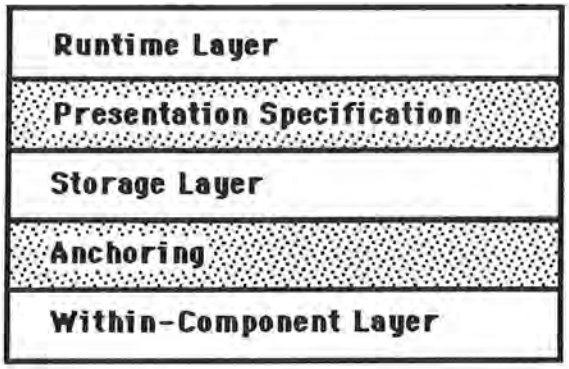

Figure 2-1. Architecture of the Dexter model.

Figure 2-1 shows a diagram of the architecture of the Dexter model. There are three layers in the model: the *storage layer*, the *within-component layer*, and the *runtime layer*. The storage layer manages all data structures that contain hypertext information. The runtime layer provides the user interface. The within-component layer performs the actual function of the application (i.e., financial analysis, engineering design, etc.). There are two interfaces in the model. The *presentation specification interface* uses data from the storage layer to instruct the runtime layer on how information is to be displayed. The

anchoring interface maps data structures from the within-component layer onto the storage layer.

The next sections cover the Dexter architecture in greater detail. This is followed by the application of the model as a standard. The last section discusses some weaknesses in the model.

2.1 The Storage Layer of the Dexter Model

The Dexter model places heavy emphasis on the *storage layer*. This is where actual data is stored. At this level, all data is stored in a *component* (figure 2-2). Each component has a *base component* that may be an *atom*, *link*, or *composite*. An atom can be thought of as the equivalent of the data in a typical hypertext node or the information contained in a HyperCard card. However, the storage layer does not know anything about the internal data structure of the atom or how it is to be displayed. All that is known is that the atom represents some piece of data.

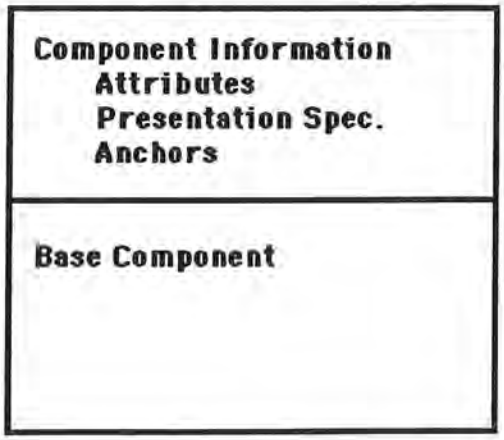

Figure 2-2.

Structure of a Component.

A base component can also be a *link* (figure 2-3). A link is a series of two or more *specifiers*. Each specifier represents an endpoint of the link. At the most basic level, a

specifier would point to an atom and a displacement within the atom. A complete discussion of specifiers is deferred until the anchor interface.

Most hypertext systems have only binary links; they are represented in the model by a link that has only two specifiers. The Dexter model allows a link to reference an arbitrary number of components. For example, a word defined in a glossary could be pointed to by all occurrences of that word in a book.

Finally, a base component may be a *composite*. A composite consists of an arbitrary number of links, atoms, or other composites. No composite may either directly or indirectly contain itself. A node that represents a rolodex file can be modelled by a composite. If the node is represented as a Macintosh icon, opening the node can display a window containing the individual rolodex cards. Such cards form a group of related information, but they do not need links to show how they are related.

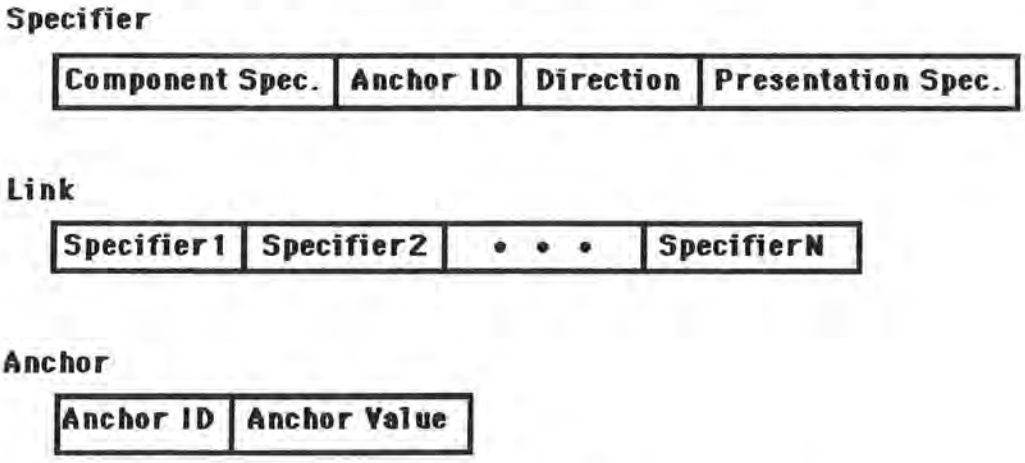

Figure 2-3. Data Structures in the Storage Layer.

Each component has *component information*. This consists of *attributes*, *presentation specifications*, and *anchors*. The storage level considers these items to be atomic and has no knowledge of their internal structure. A discussion of the presentation specification and the anchors is deferred to their interface sections.

The *attributes* section of the component information represents an arbitrary number of attribute-value pairs. Each component can have an unlimited number of attribute-value pairs that contain either system or user information about the component. For example, in a hypertext system that contains a number of books, typical attribute-value pairs for a particular component might include: AUTHOR="Goodman, Danny", TYPE="NON-FICTION", SUBJECT="COMPUTERS", TITLE="The Complete HyperCard Handbook."

In the Dexter model, all components have exactly one attribute called a *unique identifier* (UID). The storage layer is responsible for insuring that each UID has exactly one value that is unique across all components. The model uses the UID to access a specific component.

The storage layer has a set of functions for maintaining data that is similar to the functions provided by most database management systems: create component, delete component, and modify component. An additional set of functions allows adding, deleting and altering attribute-value pairs. There are also two functions for accessing components, a *resolver* and an *accessor*. The resolver function takes a set of *component specifications* as an argument and returns a UID if a matching component can be found. Component specifications are similar to the arguments of a database query. If the given set of component specifications cannot be resolved to a particular UID, the resolver function will not return a value. Once a UID has been returned, it can be given to the accessor function to gain access to the component.

2.2 The Within-Component Layer and the Anchor Interface

The *within-component layer* consists of the application software and a description of all components in the system. This description is the equivalent of a file definition or a schema. Because of the many data types available such as graphics, sound, and

animation, the Dexter model does not describe this layer in great detail. The model allows other developers to add existing standards to the model or to create new standards for future technologies.

The *anchor interface* connects the storage layer to the within-component layer. Another look at figure 1-1 shows the graphical representation of links in the browser window. The two text windows show the link indicators embedded within the text. The link indicators notify the user that this part of the text is related to some other item. In a similar manner, the anchor interface is used to map an anchor in the storage layer to a piece of data in the within-component layer. The anchor interface shows where the endpoints of a link are located.

An *anchor* designates a specific point within a base component. Each component may have an arbitrary number of anchors. The anchor interface divides each anchor into two parts, the *anchor id* and the *anchor value*. Anchor ids identify the individual anchors and are unique within each component. The anchor value represents the place within the base component that would be the endpoint of a link. The anchor value may be the number of words into a paragraph, a table index, the number of radians into a pie chart, or any other value that is appropriate. The anchor interface is responsible for making updates to the anchor value as changes are made to the data.

The storage layer defines a link as a set of two or more *specifiers*. Each specifier is an endpoint of the link and contains four parts: the *component specification*, the *anchor id*, the *direction* and the *presentation specification*. The component specification resolves to the UID of a particular component. Together with the anchor id, it identifies a specific anchor. The direction can indicate if the endpoint is either a source or destination, or if there is no direction. Discussion of the presentation specification is covered in the section on the runtime layer.

2.3 The Runtime Layer and the Presentation Specification Interface

The third layer is the *runtime layer*. This is the equivalent of the Macintosh user interface; it controls the way data is displayed and manipulated by the end user of the system. Because of the wide variety of hardware platforms available, the model does not attempt to describe a particular implementation of tools or features available to the user. Instead, this layer addresses the way data is to be presented to the user.

When a user requests that a node be displayed, the runtime layer will call the *present* function. The present function sends a retrieve request to the storage layer for the desired component. When the component is returned, the runtime layer will copy it and assign a unique *instantiation identifier* (IID) for the request. The *presentation specification interface* will examine the *presentation specifications* of the component and instruct the runtime layer on how the data in the instantiation is to be displayed to the user. For example, assume that a component contains statistical information. A window could have link references that will display the same data in either tabular or pictorial format.

The instantiation can be thought of as the equivalent of a database view. More than one instantiation of the same piece of data can exist at any one time; each instantiation has its own unique IID. The user can discard a given instantiation with the *unpresent* function. Updates to an instantiation can be applied to the storage layer by using the *realize* function.

2.4 Application of the Dexter Model as a Standard

To see if a system can be called a hypertext system as defined by the Dexter model, one can apply the process of *data reification* as described by Jones [16]. To use this process, the architecture of the system in question must be specified in Z (Zed). Z [21] is a language that is used to formally specify the Dexter model. The process requires two

steps. First, it must be possible to map the types and functions of the Dexter model to the target system. Next, it must be shown that there is at least one actual representation for each abstract value in the target.

According to the authors of the Dexter model, no work has yet started comparing the model to actual hypertext systems. A simpler way of insuring conformance to the model is to see if all functions in the Dexter model exist in the target system. This informal method will be used in this paper.

When the Dexter model was first created, its abundance of features was meant to serve as a guide for future systems. As it exists now, the model includes features that are not found in other existing hypertext systems such as multiple endpoint links and composites. Because of this, it can be said that none of the existing hypertext systems conform to the model and therefore do not qualify as hypertext systems. To remedy this, the Dexter team is currently developing a standard that will specify a minimal set of features that a hypertext system should include.

2.5 Weaknesses of the Dexter Model

Researchers have been debating about what features the next generation of hypertext systems should include. The following features are not a part of the Dexter model.

2.5.1 Advanced Search Facilities

A number of systems allow a user to search for a word within a node. *Content search* and *structural search* are advanced facilities that Halasz [13] has suggested for the next generation of hypertext systems. Content search allows a user to search for a word in all nodes of a hypertext document. Structural search allows a search for a string

depending upon the structure of the hypertext document. A structural search might request string "ABC" in all nodes that are referenced by a node that contains string "XYZ."

2.5.2 Dynamic Link Construction

Links in most hypertext documents form a static network. The author of the document inserts links into nodes in an order that he thinks is important. These links cannot be changed by the casual user of the system.

Dynamic links allow the user to determine the way nodes should be linked together. If a hypertext system contains medical information, a doctor may wish to access the abstracts of all articles on a particular disease that were written after a certain date. A system that has dynamic linking capability will allow the doctor to request that a temporary link for such information be created. Watters and Shepherd [24] have suggested a model that provides such facilities using a relational database.

2.5.3 Version Control

Version control is desirable in hypertext systems that are continually being updated by a group of users. Changes that are made to a hypertext document can be traced to a specific user and time stamped. The section of this paper on Neptune discusses version control in more detail.

3. Similar Hypertext Systems

This section discusses some of the hypertext systems that are similar to Triton. These include Neptune, Guide, and HyperCard. Neptune [6] is the system on which Triton is based. Guide [20] is a hypertext system that runs on both the IBM PC and the Macintosh. HyperCard [9, 10, 25] is the most widely known and used hypertext system for the Macintosh. For an in-depth discussion of several other hypertext systems, see Conklin [5].

3.1 The Neptune System

Neptune [6] is a hypertext system that was developed by Tektronix for keeping track of documents in a Computer Aided Design environment. It follows the layered architecture of the Dexter hypertext model very closely. Neptune's Hypertext Abstract Machine (HAM) implements the functions of the Dexter model's storage layer. HAM is written in C and runs on a UNIX mainframe as a client-server process. In addition to maintaining links and nodes in a hypertext document, HAM is a complete data base system that provides recovery from system failure and transaction synchronization for multiple users. Unlike the Dexter model, Neptune includes extensive support for version control. A user can edit several different versions of a link or document simultaneously.

HAM provides two advanced query functions. *LinearizeGraph* takes a given node and constructs a linear subset of the graph by doing a depth first search on all outbound links of the node. A second function, *getGraphQuery*, returns nodes whose attribute-value pairs match queries supplied by the user.

The Neptune user interface runs separately from the HAM and is written in Smalltalk. A user can view a diagram of a hypertext document in a *graph browser*. A *node*

browser displays the contents of the individual nodes. The getGraphQuery function returns a set of nodes that can be viewed in a *document browser*. The window of the document browser contains several panes (figure 3-1). The smaller panes display a list of the nodes returned by getGraphQuery. A larger pane displays the current node selected from this list.

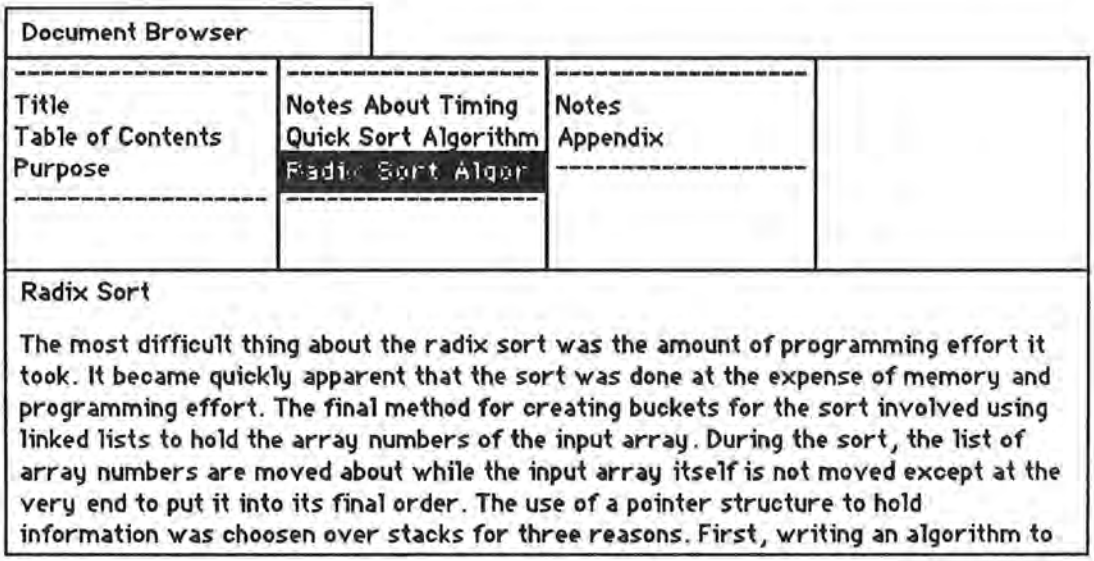

Figure 3-1. Example of a Neptune document browser.

Neptune's *application layer* provides tools that support software and electrical engineering applications. This layer is the equivalent of the Dexter within-component layer. These tools include text editors, project management software, and compilers.

3.2 Guide

Guide [20] is a product of OWL International. It supports text nodes of unlimited length and a browser window. The user can create graphical images with a supplied desk accessory or can import them into the document. There are three types of link markers: the *note*, the *reference*, and the *button*. When the cursor is moved over a text window, it

changes shape as it passes over the different type of link markers. The cursor changes into an asterisk, crosshair, or arrow when it is above a link marker indicating a note, button, or reference, respectively.

The difference among the three types of link markers is cosmetic and is done for the benefit of the user. A note is a small window that contains a brief piece of text, such as a definition (figure 3-2). It can also be a command that the system executes. A button generally points to another node. References chain nodes in front to back order so that a document may be read in a sequential manner.

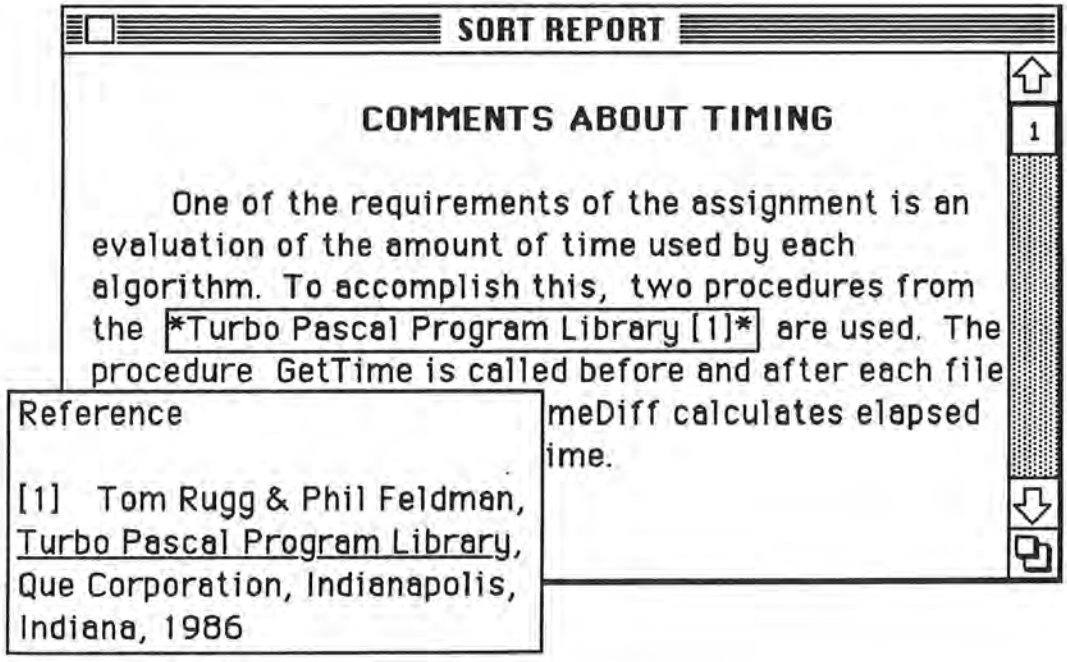

Figure 3-2. Example of a Guide note.

3.3 HyperCard

Since its introduction in 1987, HyperCard [9, 10, 25] has become the most popular hypertext system for the Macintosh. To promote the use of HyperCard, Apple Computer bundled the software with each new Macintosh it sold and provided the software free or at minimal cost through its dealers. *The Complete HyperCard Handbook* [10] provides extensive and well written documentation. This section examines the features of HyperCard and compares it to the Dexter hypertext model.

3.3.1 HyperCard Features

HyperCard stores data in one or more *stacks*. Each stack contains what would generally be one hypertext document. A stack consists of a collection of *cards*. The card is the basic unit of information; it is the equivalent of a node in other hypertext systems. Each card is a fixed size that covers the standard nine inch Macintosh screen. Variable size cards are supported in version two.

Cards can contain graphics, text, fields, and buttons. The user can create graphics and text using tools included from MacPaint, or he can import drawings into the stack with the "import paint" command. Cards are a fixed size and cannot be scrolled, however they can contain scrollable text fields. Fields defined on the card contain variable information. Buttons are used to execute small programs called *scripts* or to provide linking capability.

Figure 3-3 shows an example of a card. By clicking on the button labelled "Go to next Card," the user can navigate to another card. The button with the telephone executes a script that generates touchtone sounds. The bottom of the card contains the text "this text contains a button **here**." In this case, an invisible button has been defined around the word

here. By clicking on **here**, linking or script execution can be performed in a manner consistent with other hypertext systems.

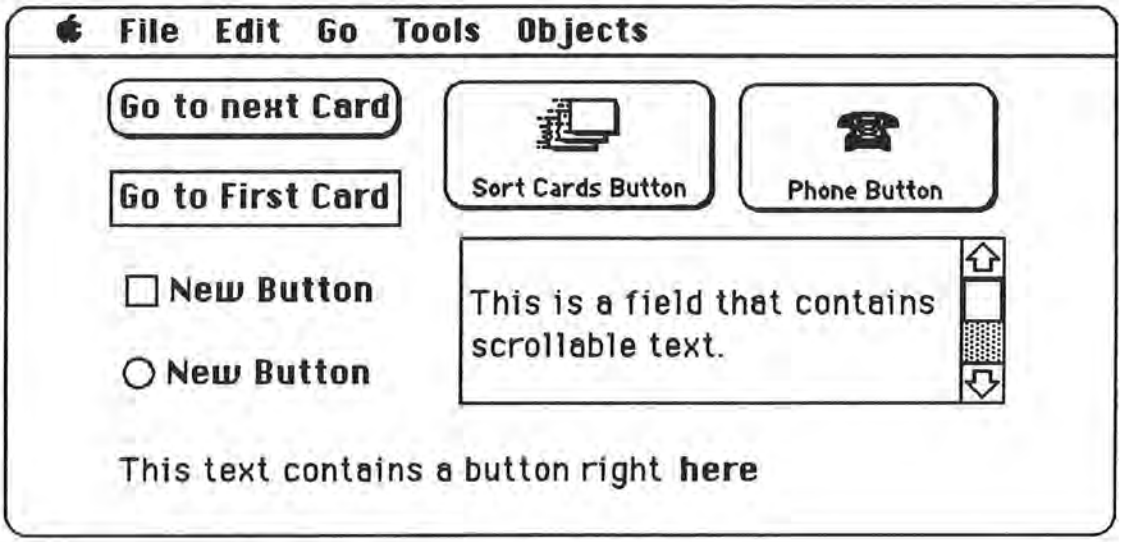

Figure 3-3. Sample buttons on a HyperCard card.

Hypercard's scripting language is called HyperTalk. Small scripts, English-like programs, can be attached to cards, fields or buttons. The scripts perform actions when they are activated by the user. HyperTalk includes arithmetic and text manipulation functions. It can query or alter values in the fields of the stack. In addition to HyperTalk, separate programs written in C or Pascal can be added to a HyperCard stack; these programs, called XCMDs, provide added functionality.

HyperCard was designed to be used by people with various levels of skill. The browsing and typing levels allow the casual user to examine and enter data into a stack. The painting and authoring levels allow the creation of a stack. The scripting level allows the stack author to use the functions of HyperTalk. Linking between cards does not require a knowledge of HyperTalk.

Buttons are only one of the navigation aids that HyperCard provides. The Go menu contains several navigation commands. The "Recent" command displays a visual history of

the last forty-seven cards that have been viewed (figure 3-4). The user can jump to any of these cards by clicking on its replica.

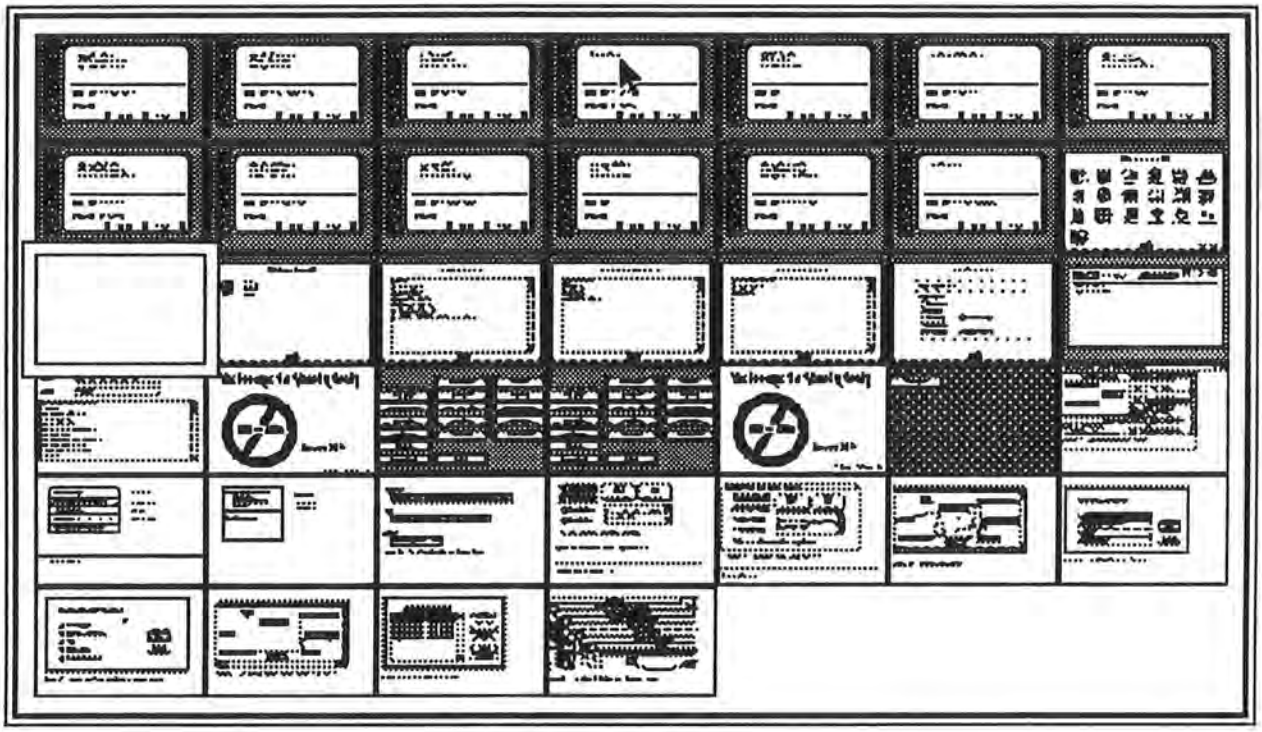

Figure 3-4. Display of last recently viewed cards.

3.3.2 Comparing HyperCard Against the Dexter Hypertext Model

Since the program source for HyperCard is the property of Apple Computer, it is generally not known if HyperCard conforms to the layered architecture of the Dexter model. HyperCard links are equivalent to single directional links in the Dexter model. It is possible to attach attribute-value pairs to cards. This can be done by defining a field on a card for the attribute and using the data stored in the field for the value.

One of the shortcomings of HyperCard is that it does not provide a browser window of all cards in a stack. Another problem is that it can not insert link markers into

text such as other systems like Triton. Creation of hypertext-like links in the text of cards can be done, but it is tedious. Some authors have said that these shortcomings show that HyperCard does not meet the criteria for a hypertext system. However, these features are not a part of the Dexter model. Using a minimal standard, HyperCard does conform to the Dexter standard.

3.4 Reg-in-a-Box

Reg-in-a-Box [8] is a hypertext application developed by the United States Environmental Protection Agency to distribute Federal regulations on underground storage tanks. The EPA created a version for the IBM PC using KnowledgePro and for the Macintosh using HyperCard. EPA's goal is to reduce environmental damage and its associated costs to business by making the regulations as readily available as possible. To do this, EPA gives the application away as freeware so that it will reach the greatest number of people.

The HyperCard version features a picture of a typical storage tank. Invisible buttons on the picture provide links to other cards. When a user clicks on the tank, the button displays a card with an exact copy of the appropriate Federal statues for the selected part. Another button in the statues links to a plain English explanation of the regulation.

William Fosket, the project manager for Reg-in-a-Box, had two complaints about the system [8]. The first is the need for tools to validate the targets of the links. During checking of the many links created, Fosket found that it was easy to point a link at the wrong destination. The second complaint was the need for links with multiple endpoints (i.e. the multiple link marker implementation as provided by Dexter). For example, a valve may have numerous regulations. In HyperCard, an invisible button covering the picture of the valve will only allow one destination.

3.5 MAGNA

ABC News developed MAGNA, the Macintosh ABC General News Almanac [17, 22], to help cover the 1988 presidential election. It is an extensive collection of HyperCard stacks that was used by newscasters and ABC support personnel. One card might contain information on a political candidate and his views. Another card could contain information on the delegates of a state. Pictures on another card could allow a newscaster to know the exact location of a state delegation on the convention floor. During the nominating process, the staff continually updated the stack with delegate counts. A HyperTalk script computed a count of total delegates for each candidate.

The creation of MAGNA was the ideal of David Bohrman, a Senior Producer of ABC News. After the election Bohrman continued experimentation with HyperCard. With the help of an ABC engineer, he was able to connect a laser disk to a Macintosh. Eventually, what started out as a special interest project of Bohrman's was spun off into ABC News Interactive. Their first product, "88 Election," contains HyperCard stacks, a video disk, and supplementary hardcopy material. Since that time, they have released products on AIDS, the San Francisco earthquake of 1989, and the "I Had a Dream" speech of Dr. Martin Luther King. With its extensive collection of film, ABC New Interactive is continuing to release new products that are aimed at the educational market.

4. The Triton Hypertext System

Triton started as an attempt to port the software engineering capabilities of the Neptune system over to the data processing facilities of Tektronix. Although the majority of business data processing at Tek runs on large scale IBM and VAX mainframes rather than workstations, there are many similarities between software engineering and creating effective business systems. Both disciplines require the creation and storage of many different documents such as proposals, data flow diagrams, software bug reports, specification sheets, and end user manuals. Large numbers of people work on such projects, and they need some method to keep track of all the documents that are created or updated.

Unfortunately, installing Neptune by simply installing a workstation is not cost effective. The workstations that run Neptune are high end products that cost many times more than current terminals or personal computers. The cost for the number of workstations required to support the data processing staff, expenditures for learning time, machine maintenance, and floor space is also considerable.

The use of the IBM mainframe to run Neptune is also unfeasible. While abundant computing power is available, the mainframes do not run UNIX. Also, availability of graphics terminals is extremely limited because they are expensive and their capabilities are usually not required in a business environment.

The Macintosh is a practical platform for several reasons. The graphical user interface provides displays similar to the ones on the workstations. Neptune is written in C, and several C compilers are available for the Mac. Low end Macintoshes are inexpensive and are common in the DP department. Many employees already use the Macintosh for word processing, spreadsheets, and presentation graphics. Because of the standard way that Macintosh applications work, users will regard Triton as just another program. They will not have to learn a new system on new hardware.

There were two attempts to implement Triton. The first attempt called for porting Neptune from the VAX to the Macintosh and for writing the user interface in C. The porting was abandoned after one month because of technical difficulties encountered. The Neptune source code was too large and too complex. In the second attempt, all Neptune code was discarded. The design of Triton focused on the user interface and on desirable hypertext features.

The next section discusses the features in the final implementation of Triton. The remaining sections discuss technical details of the implementation and a list of problems encountered in the project. Finally, Triton is compared to the standards of the Dexter model.

4.1 Features of the Triton Implementation

This section uses "Sort Hypergraph," a sample hypertext document, to illustrate features of the final Triton implementation. A copy of "Sort Hypergraph" is on the Triton distribution disk. As used in this paper, the term *hypergraph* is defined to be a hypertext document created by Triton. At the implementation level, a hypergraph is a folder that contains a set of Macintosh documents. These documents include control information and node data. The hypertext author and casual user should view the hypergraph as an atomic entity. They must not attempt to individually manipulate the components of the hypergraph. The first part of this section discusses Triton features that a hypertext author will use.

After a user launches Triton, he is prompted to enter his initials. The initials will be part of a timestamp that is used for version control.

The creation of "Sort Hypergraph" starts by selecting "New Graph" from the file menu. Triton prompts the user for the name of the new hypergraph. After the name is entered, an empty browser window appears on the desktop. Next, the user selects "Add

Node..." from the node menu. Triton opens a window for the node and draws a box that represents the node in the browser window. Although the current version of Triton supports only text nodes, no restrictions are placed on the data that can be used. Future versions of Triton may have nodes that are managed by graphics editors or project management software. Triton text nodes are of arbitrary length. The windows are resizable and scrollable.

The data in "Sort Hypergraph" was originally a computer science paper [23] created with MacWrite. Cut and paste operations transfer the text from MacWrite to the Triton nodes. The user can rearrange boxes in the browser window by clicking and dragging them. The original MacWrite document has many references, For example, the table of contents points to the various sections in the paper. There are references in the text to the bibliography. Each chapter logically follows the other, and chapters refer to tables in the appendix. These relationships provide a natural set of links. The hypertext author uses commands from the link menu to create hypertext links in a Triton hypergraph.

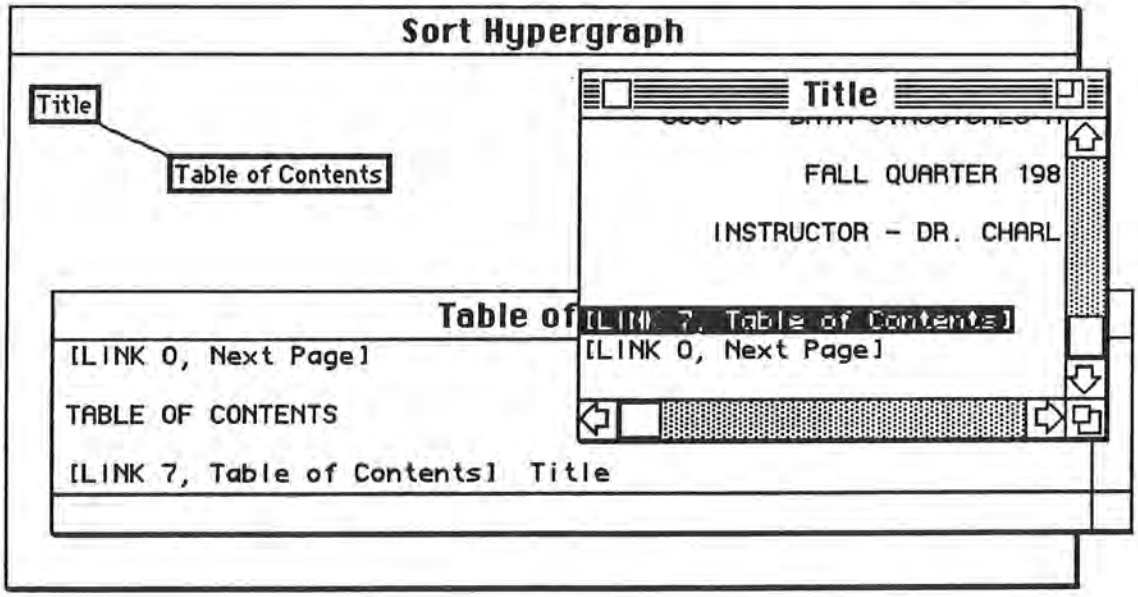

Figure 4-1. An Example of a Triton Link

To create a link, the author clicks on the piece of text where the link is desired and

selects "Define From Link" from the link menu. The author then specifies the endpoint of the link. This may be in the same or a different node. The author clicks in the destination and selects "Define To Link." Triton inserts a link indicator at both endpoints in the text. A line in the browser window shows the links between nodes (figure 4-1).

Triton supports binary links with up to four attributes. The user specifies the names of the attributes and which attributes will be assigned to a link by using the "Set Link Type" command from the link menu. Like Neptune, Triton always maintains link consistency. A link can be deleted either by selecting "Delete Link" for a particular link or implicitly by selecting "Delete Node" for a node containing one or more links. In either case, link references are deleted from all remaining nodes. "Dangling links" are not permitted. If text is added or deleted from a node, Triton will automatically update the location of link markers.

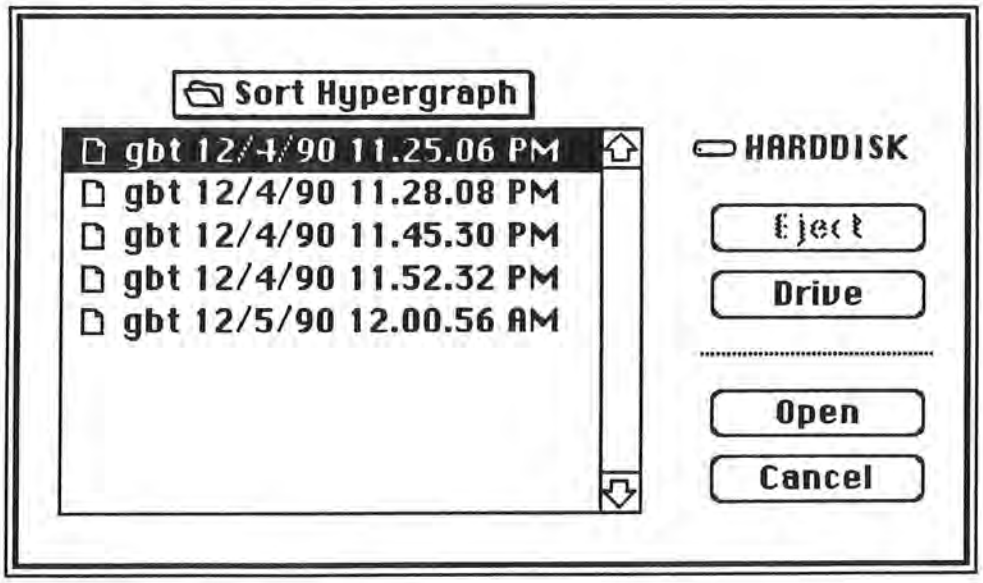

Figure 4-2. Version Control in Triton

When the author saves the hypergraph, Triton creates a new version. Later, when the author wants to update the hypergraph, Triton will display a list of all versions and will ask for the version to be used (figure 4-2).

The rest of this section discusses features used by the hypertext reader. After a user opens a hypergraph, the browser window and any author designated nodes appear on the desktop. The windows follow standard Macintosh conventions. They can be scrolled, resized, activated, and closed. Triton supplements this by allowing windows to be activated from the browser window or by special menu commands. For example, a user can select the "Purpose" box in the browser window. He can then activate the "Purpose" window with the "Open Node" command. Double clicking on the box will also activate the window.

Since the browser window is used for navigation, it should be easily accessed. The "Show Graph Window" command in the browse menu activates the browser. The "Restack Windows" command rearranges all windows on the desktop.

Triton provides two ways to traverse a link. After a user clicks inside a link marker, he can select "Go to Link" from the link menu. Triton will activate the target window and scroll to the location of the target link. Triton keeps track of the last one hundred link traversals. The user can return to the starting point of the last traversal by selecting "Return From Link."

The user can use link attributes to assist in viewing the browser. Figure 4-3 shows all links defined in "Sort Hypergraph." A common complaint of hypertext systems is that numerous links can create a feeling of disorientation because of the amount of clutter in a browser window [19]. If a set of link attributes is defined (figure 4-4), the user can select "Display Link Type" from the link menu to restrict the types of links that are displayed (figure 4-5).

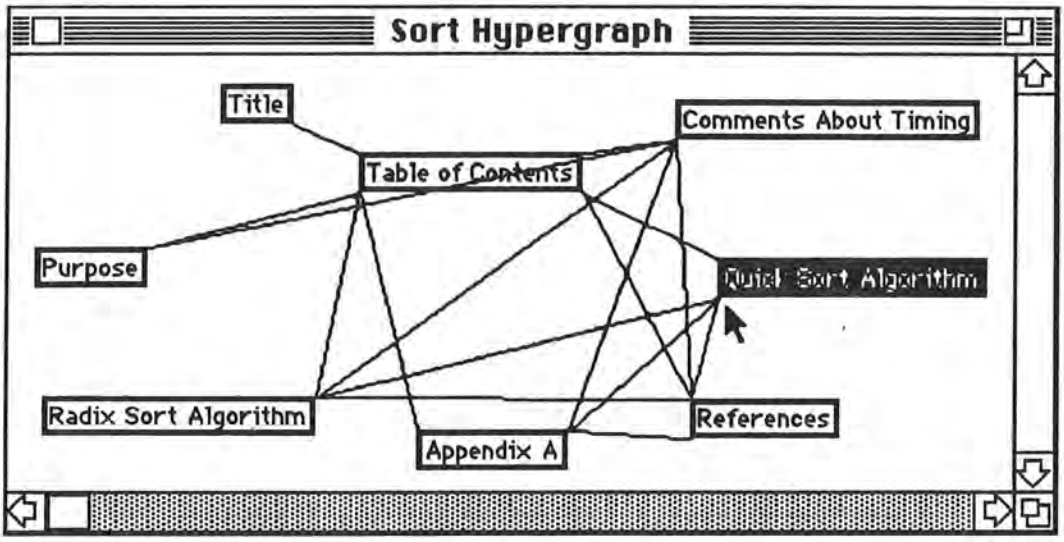

Figure 4-3. All of the Links in "Sort Hypergraph"

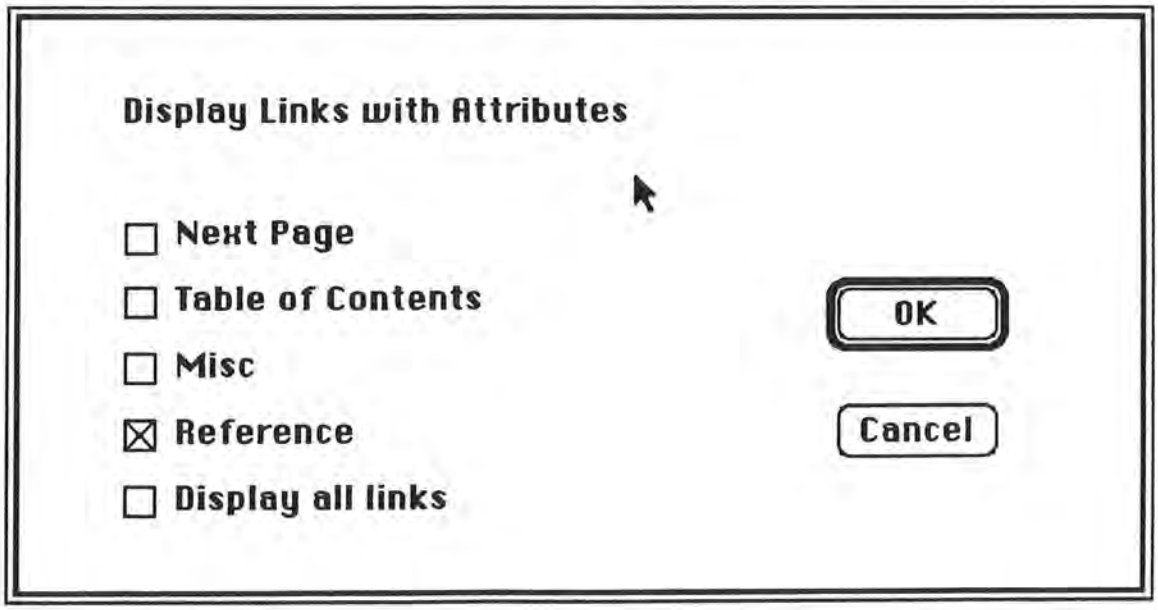

Figure 4-4. Restricting Links Displayed in the Browser Window

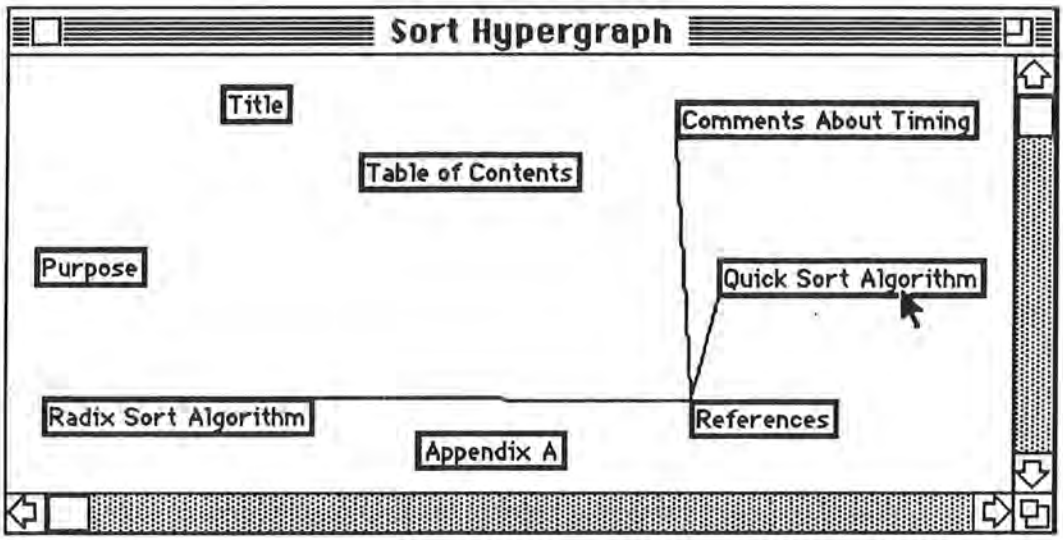

Figure 4-5. Displaying Links with the "Reference" Attribute

4.2 Technical Details of the Triton Implementation

Triton is a standard Macintosh application. With the exception of the storage of hypergraphs and the definition of links, Triton obeys all Macintosh user interface guidelines [15]. The Triton source consists of eight files written in C. Each file contains code to handle a specific area, such as window management or menu command processing. There are approximately 8,000 lines of code. The application takes up 32K of disk space and consists of the Triton code and routines from the MacTraps library; it contains no other code or packages.

Triton stores a hypergraph as a set of Macintosh documents within a folder. A given version of a hypergraph consists of a control document and a document for each node in the hypergraph. The folder contains all versions of the hypergraph.

The document for each Macintosh node contains only data. The control document contains all information for a single version of a hypergraph. This includes a list of all

nodes and links in the graph, free lists for unallocated links and nodes, and the names of the link attributes.

The node information entries are kept in a single array in the control document. Each entry specifies the Macintosh document name for the node it represents and contains chain entries. A forward and backward chain links the nodes in the hypergraph. A node may be either on the list of free nodes or on the list of allocated nodes. These lists are anchored in the control document. A similar array contains all links in the hypergraph. When Triton needs to create a new node or link, a subroutine returns a free entry and does all chaining.

Use of arrays allows the data for all nodes and links to be grouped together in one document. This is efficient when writing the control record out to disk. Also, since storage is already allocated, Triton can avoid the overhead of **malloc** or **GetNewHandle**. By using index numbers rather than real addresses, Triton avoids relocation problems when the hypergraph is written to and from disk.

Each node points to a list of its links. When a user clicks in a node, the Macintosh **TEClick** routine returns the location of the insertion point. Triton scans the list of links to determine if the click occurred within a link marker. Although the link array contains all links in the hypertext document, Triton scans only the links belonging to the node. Triton uses a linear list of the link entries because it is simple. Triton is a prototype and is not expected to handle large documents with hundreds of links. For large systems, Triton would need a more sophisticated method so that response time is acceptable to the user. Hash tables are fast but can waste disk space if the links are sparse. Since Triton already knows which node is being searched and the displacement of the link into the node, a better method would use some kind of index system.

4.3 Problems Encountered During the Project

Three problems were encountered during the project. The first was with the way Triton stores hypergraphs. A word processing program will leave one document on a disk that can later be modified or dragged to the trash. Triton on the other hand, creates a folder with the name of the hypergraph. Each node and control document is stored as a separate document within the folder. Discarding any document within the folder will destroy the hypergraph. Triton's way of storing a hypergraph, which is logically one document, is contrary to the Macintosh user interface guidelines. Since the objectives of the Triton project no longer focused on the HAM, this violation was ignored to simplify the project.

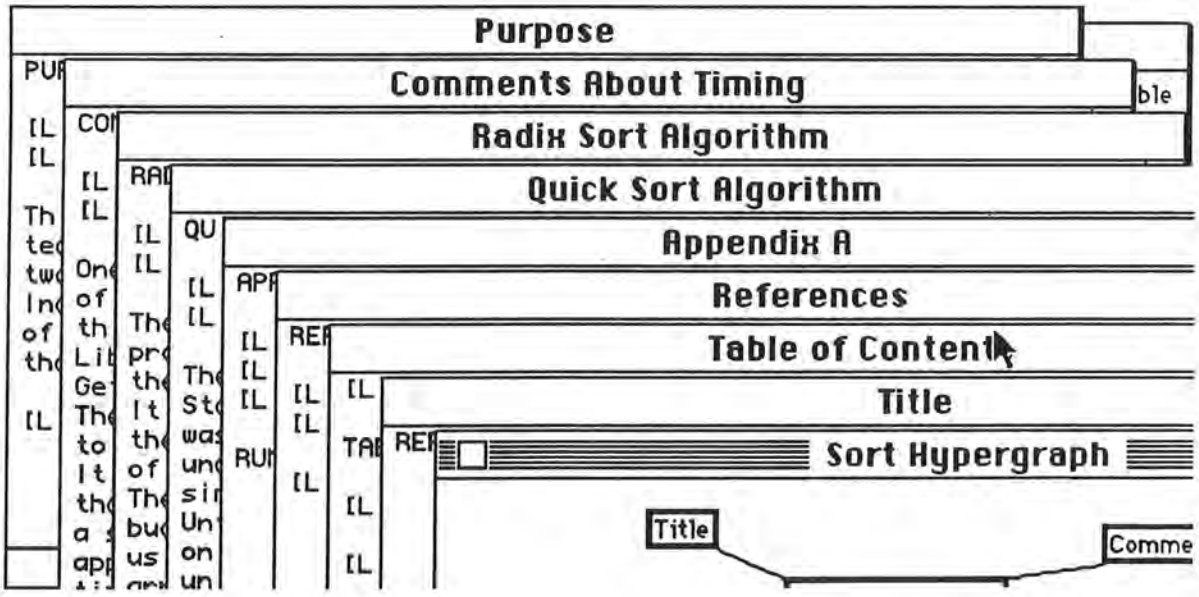

Figure 4-6. Crowding of Windows in a Macintosh Screen.

Triton's method of defining links created another problem. Once a define link operation is started, Triton will not allow the user to do anything until the end of the link is defined. The modeless program philosophy of the Macintosh dictates that the second part of the link definition should not prohibit any other operations. This violation was

necessary to insure that anchors were consistently maintained in the text of each window.

The final problem was caused by the small size of the standard Macintosh screen. "Sort Hypergraph" quickly revealed the problem of having many hypertext nodes crammed within the Mac's nine inch screen (figure 4-6). The "Show Graph Window" and "Restack Windows" options in the browse menu were added as afterthoughts. Since the browser window can be used as a navigation aid, it is convenient to make the graph window easily available. The "Restack Windows" option is standard in other products such as Microsoft Word and Excel. It was found to be a necessity because many more windows are usually opened under Triton.

4.4 Triton's Adherence to the Dexter Model

This section discusses how well Triton conforms to the standards of the Dexter hypertext model [14]. The Triton project places its main emphasis on the user interface and the application layer. While the Dexter model concentrates on the storage layer, there are several features that can be compared.

Triton does not follow the Dexter model's layered architecture. The functions for the application layer, storage layer, and user interface are divided among Triton's eight source files. In a typical Macintosh application, modules are grouped together based on the data types they handle. For example, a "window unit" contains all window operations. Since Triton is a single program that does not require client-server functions, it is simpler to structure Triton like a traditional Macintosh application.

Triton's file system is similar to the Dexter model's storage layer. The file system does not need to know anything about the data except the version to be used. There are no restrictions on the data contained in nodes.

Triton links are very similar to Dexter links. Although Triton supports only binary links, they are equivalent to a Dexter link that has two specifiers. Triton links can have link

attributes associated with them. Triton links contain anchors that specify both a node and a displacement as Dexter links do.

While the storage layer features of Triton are very simple, Triton does have a minimal set of the Dexter features.

5. Future Improvements to Triton and Other Hypertext Systems

This section discusses things that were learned during the development of Triton and how they might be used to improve existing and future hypertext systems. These areas include the use of the layered architecture of the Dexter model, client-server database systems, open software architecture, and tools for the hypertext document developer.

5.1 Dexter Architecture

The use of the layered architecture of the Dexter model can provide flexibility in the design of hypertext systems. Neptune's developers have stated that they are usually referring to the HAM when they talk about Neptune. Since the HAM, like the Dexter storage layer, knows nothing about the data it's processing, this implies that Neptune could be used for other applications. This can be done by replacing the within-component layer and modifying the user interface. For example, Neptune's strong version support would be very useful in a medical records system. Such systems require that the content of any change, the author, and time of change be identifiable.

The separation of the user interface layer from the within-component layer provides other advantages. By including things such as non-hardware specific tools, computational engines, and artificial intelligence capability in the within-component layer, we can port this layer to other platforms and to other types of applications without the need for extensive modification. Also, the user interface layer can handle the specific hardware without any need for a knowledge of the application.

5.2 Client-Server Database Systems

The attempt to port the Neptune HAM over to the Macintosh consumed much effort and was unsuccessful. To eliminate a major part of the development of a hypertext system, several researchers have proposed using relational databases to provide the functionality of the Dexter storage layer [24]. The Neptune HAM is a database management system that uses client-server communications. In addition to reducing development costs, keeping HAM on a mainframe has the advantage of providing computing power and multi-user synchronization not available on a Macintosh.

Apple Computer has realized the importance of the client-server architecture and is creating a number of products to exploit this facility. A recent videotape called "Apple Seminar for One" [1] demonstrates some of these products. A Macintosh is connected to both a VAX and an IBM mainframe. A user formulates plain language database queries in a Macintosh dialog box. The Macintosh translates the request into the appropriate query language for the target processor. Information returned from the mainframe appears in a Macintosh window. Such information can be selected and copied to other applications on the Macintosh or to other mainframe servers.

5.3 Open Architecture

One of the requirements for future hypertext systems is that they have an open architecture [18]. Allowing either MacWrite or Microsoft Word to process a Triton node while maintaining link integrity would be an example of such an architecture. Although most software companies are going to be unwilling to provide this capability for other vendors, open architecture is available to some extent on the Macintosh. A text file on a PC contains both text and control information that must be interpreted by the word processing program. A Macintosh document stores data differently. A document is stored in two parts,

the data fork and the resource fork. The data fork contains the text data. The resource fork contains an arbitrary number of resource entries. Entries in the resource fork tell the Macintosh things about the document environment, such as which software application created the document. It is possible to add information such as link anchors to a MacWrite document by adding additional resource entries to the resource fork. The document can be edited with MacWrite or used with the hypertext system. Additional research is needed to see if hypertext control information can be updated after the document is edited.

5.4 Tools for the Hypertext Document Author

While some studies have been done on the user interface and how it affects the user of a hypertext system [5, 19], little has been done for the author of hypertext documents. There is nothing in the current literature about a set of standard tools for the author.

Three examples of problems caused by a lack of such tools have been discussed in this paper. The first is HyperCard's tedious and time consuming process for creation of hypertext like links. In the second example, the authors of Reg-in-a-Box have discussed the need for a tool to verify that links are properly targeted. The final example is the creation of the sample document for the Triton project. "Sort Hypergraph" has a large number of nodes that had to be constantly resized and moved about while constructing the document. The hypertext author needs a set of tools to control the clutter on the electronic desktop.

5.5 Conclusions

At first, the idea of porting Neptune from the VAX to the Macintosh was appealing because of the benefits that would be provided to the end user. One of the things that became apparent was that the design of a hypertext system must consider the number of

people that require concurrent access to a hypertext document and the type of access they need (i.e. read or update). The use of large mainframes that provide centralized control will be a necessity for large complex hypertext documents that are maintained by many people.

The Dexter hypertext model is a worthwhile tool. It can assist future hypertext system developers in two important ways. First, the model can provide a checklist of features that users will expect from a hypertext system. Second, the Dexter layered architecture can assist the developer in designing a system for a specific hardware platform and a specific user. For example, IBM and Apple Computer are currently designing hypertext systems for the educational market. While the authors of a particular product may require concurrent access for product development, the end user could be a single student. The student may be using a personal computer with the hypertext document on a CD-ROM. He will not require connection to a mainframe or the ability to update the document. By using the Dexter layered architecture, appropriate features can be designed into a hypertext system that will support both the author and the user on the appropriate platform.

Although there are many advantages to using the Dexter model, Triton does not use its layered architecture. Since Triton does not need client-server services and is not written in an object oriented language, it was easier to program Triton using the traditional Macintosh event-loop technique. As object oriented languages develop, it will be easier to implement the Dexter architecture on the Macintosh.

Bibliography

[1] Adams, Douglas. "Pathways and Relationships", MacUser, December 1987, pp 161-164.

[2] Apple Seminar for One, Video tape, Apple Computer, 1990

[3] Bush, Vannevar. "As We May Think", Atlantic Monthly, July 1945, pp 101-105.

[4] Carlson, David A. and Ram, Sudha. "HyperIntelligence: The Next Frontier", Communications of the ACM, XXXIII (March 1990), pp 311-321.

[5] Conklin, Jeff. "Hypertext: An Introduction and Survey", Computer, September 1987, pp 17-41.

[6] Delisle, Norman and Schwartz, Mayer. "Neptune: a Hypertext System for CAD Applications", Computer Research Laboratory, Tektronix, Inc., Beaverton, OR, 1986.

[7] Elmasri, Ramez and Navathe, Shamkant B. Fundamentals of Database Systems, Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc., 1989.

[8] Foskett, William H. "Reg-In-A-Box: a Hypertext Solution", AI Expert, February 1990, pp 38-45.

[9] Goodman, Danny. "The Two Faces of HyperCard", Macworld, October 1987, pp 123-129.

[10] Goodman, Danny. The Complete HyperCard Handbook, New York: Bantam Books, September 1987.

[11] Goodman, Danny. Danny Goodman's HyperCard Developer's Guide, New York: Bantam Books, July 1988.

[12] Glushko, Robert J. "Visions of Grandeur?", UNIX Review, VIII (February 1990), pp 70-80.

[13] Halasz, Frank G. "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems", Communications of the ACM, XXXI (July 1988), pp 836-852.

[14] Halasz, Frank and Schwartz, Mayer. "The Dexter Hypertext Reference Model", Submitted to the NIST Hypertext Standardization Workshop, Gaithersburg, MD, January 16-18, 1990.

[15] Inside Macintosh Volume I, New York: Addison-Wesley Publishing Co.1985, pp27-70.

[16] Jones, C. B. Systematic Software Development Using VDM. Hertforshire, England: Prentice-Hall International. 1986

[17] Levy, Steven. "How ABC Elected HyperCard", Macworld, June 1989, pp 47-52.

[18] Meyrowitz, Norman. "The Link to Tommorrow", UNIX Review, VIII (February 1990), pp 58-67.

[19] Nielsen, Jakob. "The Art of Navigating Through Hypertext", Communications of the ACM, XXXIII (March 1990), pp 296-310.

[20] Rasmus, Daniel W. "Hypermedia, Guide 2.0", MacUser, April 1990, pp 70-72.

[21] Spivey, J. M. The Z Notation, Hertforshire, England: Prentice-Hall International. 1989

[22] Stefanac, Suzanne and Weiman, Liza. "Multimedia, is it Real?", Macworld, April 1990, pp 116-123.

[23] Terusaki, Gary B. "Report on Solution of Sorting Algorithms", Computer Science Paper, October 1986.

[24] Watters, Carolyn and Shepherd, Michael. "A Transient Hypergraph-Based Model for Data Access", ACM Transactions on Information Systems, VIII (April 1990), pp 77-102.

[25] Williams, Greg. "HyperCard 2.0 - a good tool gets better", Apple Direct, May 1990.

[26] Winston, Alan. "The Age of Hypertext", UNIX World, November 1990, pp 86-90.

Appendix A - Instructions for Running Triton

Triton is a hypertext system that supports text nodes, a browser window, and linking capability. Triton will run on any Macintosh Plus or above. Permission is granted to copy and use Triton provided that the information in the "About Triton" dialog in the Apple menu is neither altered nor removed. Triton is distributed free and without any warranties. Use it at your own risk.

These instructions are divided into three sections: installation, tutorial, and menu reference. It is assumed that the user is familiar with operation of a Macintosh computer and has experience running several applications. It is also assumed that the user has some experience with hypertext systems.

A1. Installation

Copy Triton to the disk or folder where it is to be installed. Triton is now ready to run.

A2. Tutorial

Use this tutorial with the Triton software. You will become familiar with the capabilities of Triton by creating and navigating through a small hypertext document. Throughout this tutorial the terms *hypertext document* and *graph* are used interchangeably.

1. Launch Triton by double clicking on its icon. A dialog will appear asking you to enter your initials. This is used for version control when the hypertext document is saved.

2. Select "New Graph..." from the file menu. A dialog will ask you to name the new graph. Specify a name of *myGraph*. A new FOLDER with this name will be created on your disk.

   WARNING - A Triton hypertext document is saved as a group of Macintosh documents in a folder. Do not use the Finder to move documents out of this folder.

   An empty window with the name "myGraph" is displayed on the screen. This is the browser window. It will contain a drawing of all nodes in the graph.

3. Select "Add Node..." from the node menu. You will be asked to name the node. Call it *Able*. A text window called "Able" appears. Now type the following lines into this window:

   This is line 1 of Able.
   This is line 2 of Able.

4. Create a new node called *Baker* by repeating step 3.

5. Create a new node called *Charlie* by repeating step 3.

6. All the windows have zoom, close, and size boxes; they behave like standard Macintosh windows. Arrange the windows so they look similar to figure A-1.

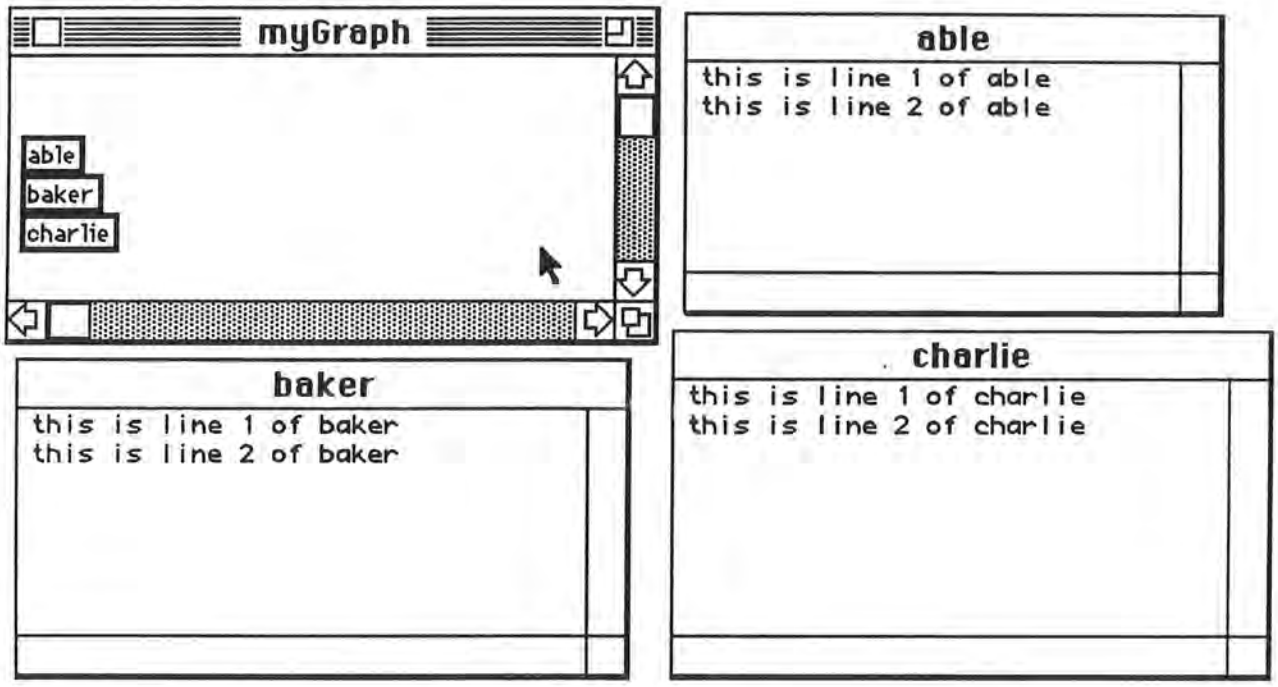

Figure A-1. Window layout after finishing step 6.

7. Click on "myGraph," the browser window.

8. Each of the nodes: Able, Baker, and Charlie are represented by small boxes in the browser window. Reposition the small box labelled Baker by clicking on it and dragging it approximately an inch to the right. Release the mouse button. In a similar manner, move the box labelled Able about half an inch to the right.

9. Select "Set Link Type" from the link menu. You may define four different link attributes for your graph. Any or all of the four attributes may be assigned to a single link.

   Replace the field "Attrib1" with *myType1*. Replace the field "Attrib2" with *myType2*. Make sure that only the box labelled myType1 has a check mark. Click on the OK button.

10. Click on the Able window. Set the text insertion point before the first line. Select "Define From Link" from the link menu. You will get a warning stating that the define link operation must be completed or cancelled before anything else can be done.

    Click on the Baker window. Set the text insertion point before the first line and then select "Define To Link" from the link menu.

    Link markers will be inserted in the Able and Baker windows. The links are numbered and contain the names of the attributes attached to the link. The browser window is updated with a line between the Able and Baker boxes indicating that a link has been established.

11. Select "Set Link Type" from the link menu again. Remove the check mark from the myType1 box. Place a check mark in the myType2 box.

12. Set the insertion point at the end of the text in the Charlie text window. Select "Define From Link" from the link menu.

    Set the insertion point at the end of the text in the Able window. Select "Define To Link" from the link menu. Another link is drawn in the browser window between Able and Charlie.

13. Click on the close box of the Baker and Charlie windows. Only the Able window and the browser window should be on the screen.

14. Select "Display Link Type" from the link menu. Make sure only the myType1 box is checked. Click on OK. One of the links disappears from the browser window. The remaining link has an attribute of myType1 associated with it.

    Select "Display Link Type" from the link menu and check the box labelled "Display All Links." Click on OK. Both links should now be visible in the browser window.

15. Click on the Able window. Place the insertion point at the end of the first line. Type several characters. Hit the backspace key as many times as possible. When you try to backspace over the link indicator, Triton will give you a warning. Deletion of links requires special operations from the link menu or browser window.

16. Click on the first link in the Able window. The entire link marker is highlighted. Now select "Go to Link" from the browse menu. This link points to the Baker window, which will now be made visible.

17. Click on the browser window. Double click on the small Charlie box in the browser window. Triton will now make Charlie the active window. All windows in the graph should now be visible.

18. Click on the browser window. Select the small Able box by clicking on it; it should be highlighted. Select "Delete Node" from the node menu. The Able node has been discarded. It is removed from the screen and erased from the browser window. The link indicators in the other nodes are removed so that link consistency is maintained.

This is the end of the tutorial.

A3. Triton Menu Commands

This section lists the commands in the Triton menus.

FILE MENU

New Graph - Prompts the user for the creation of a new hypertext document. A folder will be created using this name. All hypertext documents are composed of several Macintosh documents that are stored in this folder. DO NOT use the Finder to move any documents out of this folder.

Open Graph - Opens a previous saved hypertext document. Triton saves previous versions of a document and allows the user to access any of the past versions. See figure A-2.

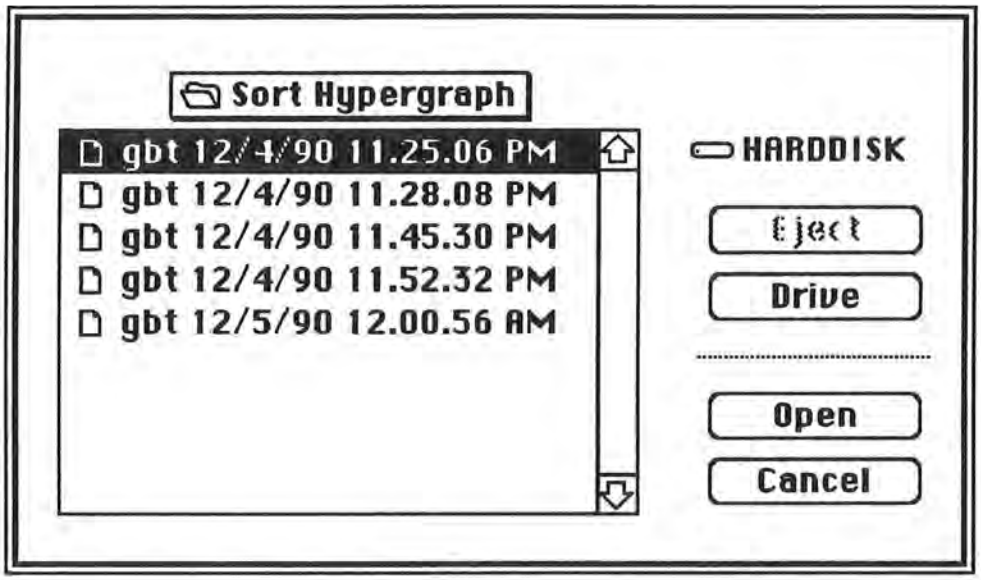

Figure A-2. Open command in FILE menu.

Close Graph - Closes the current hypertext document. Only one hypertext document may be open at any one time.

Save Graph - Saves the current hypertext document.

Save As - Saves the current hypertext document under a new name.

Page Setup - Prompts the user for preferences to be used in printing any of the nodes.

Print - Prints the currently active window.

Quit - Quits Triton.

EDIT MENU

This menu contains the standard items for Cut, Copy, Paste and Clear. The menu works only with text. If you are using Multifinder you can import text into Triton from your word processing program.

NODE MENU

Add Node - Requests a new node name from the user and adds the node to the hypertext document.

Delete Node - Removes the current node from the hypertext document. Any links in this node will be removed from all other nodes.

Open Node - When a node is selected in the browser window, this command will open the node and make it the current window.

Close Node - Hides the current node window. This can also be accomplished by clicking on the close box of the node.

Get Node Info - Displays information about the current node and allows the node name to be changed.

LINK MENU

Define From Link - This option is enabled when the insertion point is present in the current text node. This option starts the definition of a link. The operation must be completed by going to the destination of the link, placing the insertion point at the appropriate location, and selecting "Define To Link" from the link menu.

Define To Link - Completes the definition of a link started by "Define From Link."

Cancel Define Link - When a link definition has been started by "Define From Link," it can be cancelled with this option. The link marker for this operation will be removed from the starting node.

Delete Link - When a link marker is highlighted in a node, this command will remove the link and erase the link markers in both the source and destination nodes.

Set Link Type - Triton allows a link to have up to four attributes. This command is used to 1) set the names of the attributes and 2) allow any combination of the four attributes to be assigned to a link in the next "Define To Link" operation.

To set the names of the attributes, type their names in the fields of the dialog. To set the attributes that will be assigned in the next "Define To Link" operation, click the check boxes for the attributes to be assigned.

Display Link Type - This item controls which links are seen in the browser window. Only those links that have the selected attributes will be displayed.

BROWSE MENU

Go to Link - After selecting a link marker in a text node, this command will open the destination node and scroll to the marker's endpoint.

Return from Link - Triton keeps track of the last 100 links that were traversed. This item will return the user to the origin of the last link traversed.

Show Graph Window - Brings the browser window to the front and makes it the current window.

Restack Windows - Resizes and stacks all windows in the hypertext document.