Scripted Documents: A Hypermedia Path Mechanism

Polle T. Zellweger

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

ABSTRACT

The concept of a *path*, or ordered traversal of some links in a hypertext, has been a part of the hypertext notion from its early formation. Although paths can help to solve two major problems with hypertext systems, namely user disorientation and high cognitive overhead for users, their value has not been recognized. Paths can also provide the backbone for computations over a hypertext, an important issue for the future of hypertext. This paper constructs a framework for understanding path mechanisms for hypertext and explores the basic issues surrounding them. Given this framework, it reviews path mechanisms that have been provided by other hypertext systems. Finally, it describes the Scripted Documents system, which has been developed to test the potential of one powerful path mechanism.

1. INTRODUCTION

Hypertext is a valuable contribution to the information age, allowing readers to access related information through machine-supported links. However, current hypertext systems have several well-recognized problems. Two of the most significant are *user disorientation* in hypertext networks and the additional *cognitive overhead* needed to create, manage and choose among links [Conk87a, Conk87b]. Moreover, the browsing paradigm offered by most hypertext systems does not meet the needs of readers who are unfamiliar with the material being presented [Hamm88]. An important component of the information conveyed by an author to a reader in a traditional setting is the order in which the material appears. In most current hypertext systems, readers may fail to understand the material presented because they view it in the wrong order, or they may simply comprehend it less well.

The concept of a path, or ordered traversal of some links in a hypertext, can help solve these problems. Users are less likely to feel disoriented or lost when they are following a pre-defined path rather than browsing freely, and their cognitive overhead is reduced because the path either makes or narrows their choices. Paths allow authors to determine an appropriate order of presentation for a given audience. In addition, paths can provide the backbone for computations over a hypertext, one of Halasz's seven issues for the next generation of hypermedia systems [Hala88].

Although paths have been a part of the hypertext notion from its early formation [Bush45], few current hypertext systems provide paths, and in fact the path concept has not been examined systematically. As we shall see, paths need to become first-class citizens in hypertext systems. Systems should provide path mechanisms that make all kinds of paths easy to create, to find, and to follow.

Paths are not intended to supplant browsing as a hypertext navigation paradigm, but rather to augment it. Obviously paths can reduce the readers' flexibility as well. The

challenge is to provide paths that are expressive and/or are an integrated adjunct to other links within a hypertext network.

This paper constructs a framework for understanding path mechanisms for hypertext and explores the basic issues surrounding them. Given this framework, it reviews path mechanisms that have been provided by other hypertext systems. The final section describes the Scripted Documents system, which has been developed to test the potential of one powerful path mechanism.

2. A FRAMEWORK FOR EFFECTIVE PATH MECHANISMS

There is more to providing paths than merely allowing authors to specify sequences of nodes or links. This section presents a framework for understanding and evaluating path mechanisms.

An effective path mechanism must satisfy three major requirements. It must bring expressive power to authors, help authors create and modify paths, and help readers find paths and follow them in flexible ways. Furthermore, it should provide for efficient storage and execution.

2.1 Expressive power of paths

By a path, we mean a presentation of successive entries, ordered such that most or all of the decisions about the order of presentation are made by the author in advance, rather than by the reader during path playback. The expressive power of a path is determined by two things: the sequencing model that the path uses to order entries and the characteristics of the entries that can appear along the path. We discuss each in turn.

We define several different sequencing models, ranging from simple sequential paths to more complex paths that are essentially programs.

A *sequential path* is an ordered sequence of entries. Sequential paths can be used to present an ordered progression or merely to collect an unordered set of entries. Most existing implementations allow an entry to appear repeatedly in a path.

A *branching path* contains branches from which the reader must manually choose, and thus acts as a smaller subnetwork of the hypertext network. Although branches provide options to allow readers to customize a path for their interests, they also increase the cognitive overhead required to follow the path.

A *conditional path* contains branches at which the system evaluates author-specified tests to determine which direction to go next. For example, conditional paths can be used to tailor presentations to a given class of readers, to adapt to particular hardware constraints such as lack of color or audio output devices, or to provide information appropriate to the current time. A path may also be able to examine its previous history. A conditional path that asks the reader questions can provide control similar to branching paths.

Extending the simple idea of conditionals to a more complete programming paradigm can provide more expressive power. Several significant types of paths appear along this progression:

- *procedural paths*, which allow a path to appear as an entry of another path, and thus ease a script author's job by supporting modularity and re-use;
- *programmable paths*, which are conditional paths that can store values in variables to form indexed loops or record information for later use;
- *variable paths*, which are conditional paths that contain variable links, in which the next entry to be presented is located dynamically, perhaps

computed by an earlier entry or perhaps involving a search through the hypertext; and

- *parallel paths*, which are paths that can execute simultaneously and may be able to fork, join, or synchronize with activities on another branch.

We call the items that are linked together along a path *entries*. Their purpose is to provide the content, while the path provides the sequencing.

In its simplest form, an entry is (or refers to) a node in the underlying hypertext, and the path provides a way to display a sequence of nodes.[1] However, we note that hypertext systems have offered users a wide range of addressing granularity for links, from points or regions within a node to collections of nodes [Conk87a, Trig88]. A path mechanism may offer the user the same entry granularity as the underlying hypertext system, or it may offer a different one (typically coarser). Notice that although link endpoints have asymmetric granularity in most current hypertext systems (e.g., point source and node destination), each path entry acts as both a link source and a link destination.

An entry may thus be a separate object that shares some of the features of a node or link in the underlying hypertext. It may refer to a point or region within a node, a node, or a collection of nodes. It may also have additional fields that modify or add to the behavior of the underlying node(s).

We call entries that can perform actions *active entries*. For example, an active entry might animate a picture, play back voice, perform a computation, or construct part of an output document. Combining active entries with paths is particularly appropriate, because the paths can provide the ordering for the actions performed at each entry, thus creating sequences of actions that flow across the hypertext network. A path mechanism that incorporates active entries should allow a flexible association between entry locations and entry actions, so that a path (or possibly different paths) can visit the same location repeatedly and perform different actions.

2.2 Creating and editing paths

Although paths can be created manually using only conditional links, this process is quite tedious and error-prone. A path mechanism should include a path editor that allows authors to create and edit paths from path-level views. They should be able to link existing material into paths quickly and easily. They should also be able to edit paths by rearranging, adding or deleting entries. These editing operations may take different forms depending on the physical representation of the path.

Paths can be represented as sequences, as directed graphs, or as "programs" (linear text), as shown in Figure 1. Sequences and programs have no explicit links. Instead, implicit links are formed respectively by their adjacency relationships and interpretation rules.

2.3 Visualization and navigation support

Visualization tools help authors maintain paths and help readers find them.

Because paths form a meta-level above the underlying material, they must be kept consistent when authors edit that material. The responsibility for this maintenance rests jointly on the author and the path mechanism, but the path mechanism must at least alert the author when paths cross an item that is being edited. For example, if an author

---

[1] A path could also consist of a sequence of links. However, this imposes a much stricter consistency requirement on the author or the path editor: the destination of entry i must equal the source of entry $i+1$.

deletes material from the underlying hypertext network, any corresponding entry should also be removed from all paths in which it participates. Ideally, the author would be alerted appropriately *before* the material is deleted. If a corresponding entry has actions, the author should also be informed that the computational semantics of the path may be in jeopardy. Edits to the underlying material may also cause the opposite problem, in which an edited entry remains a member of a path to which it is no longer relevant. Moving or copying underlying material may also cause difficulties for some path mechanisms.[2]

Readers must be able to find relevant paths easily, just as they must be able to view and select among hypertext links. Ideally, this means that readers would be shown that paths start from or traverse the current location. However, local indicators are not sufficient, because readers may never stumble on places that are members of relevant paths. Thus there must also be global ways to locate relevant paths. Both local and global methods must be robust and easy to use, because their intended audience is the reader who is unfamiliar with the material and/or the hypertext network.

2.4 Playback control

A path mechanism must also allow readers to play back or display a path.[3] We describe several possible methods that provide readers with varying playback control. A system may provide more than one method.

A system with *single-stepping control* allows the reader to start a path and then issue a "next" command repeatedly to traverse it in order.

A system with *automatic control* allows the reader to start a path and have it continue automatically to successive entries. Such systems must allow authors to specify timing to indicate when the next link should be traversed. In addition, readers must be able to stop a path that is executing and possibly resume later. An ideal system would further allow the reader to adjust the playback speed to his or her liking.

In addition, a system may provide *browsing control*, to permit a user to visit entries in a path in an arbitrary order by selecting them individually from a path-level view.[4] This provides a level of mixed initiative that can be useful as the expertise of a reader increases [Hamm88].

However, combining programmable paths or active entries with browsing control can be dangerous. In particular, the actions earlier in a path may be necessary pre-conditions for either the branching decisions or the actions at a later step of the path. For example, earlier actions may open a window that a later entry wishes to write in, or they may record a reader's preference that will be used later to direct the specific path. Devising flexible methods for ensuring that all pre-conditions have executed before a reader may jump to a new point in a path remains a topic for further research.[5] In addition,

---

[2]Many of these situations could damage corresponding links as well, especially in systems that allow some links to be invisible, such as Intermedia [Yank88].

[3]A path could be presented spatially rather than temporally, by concatenating the entries in the path to form a single linear view. Playback control is not really relevant in this case.

[4]We consider that a path mechanism that provides only browsing control is not really a path mechanism, because it provides no path interpreter.

[5]A generalization of Textnet's *prerequisite* and *must-follow* tags on links [Trig86] might prove useful here.

designing paths to permit the maximum reader flexibility is a significant challenge for path authors.

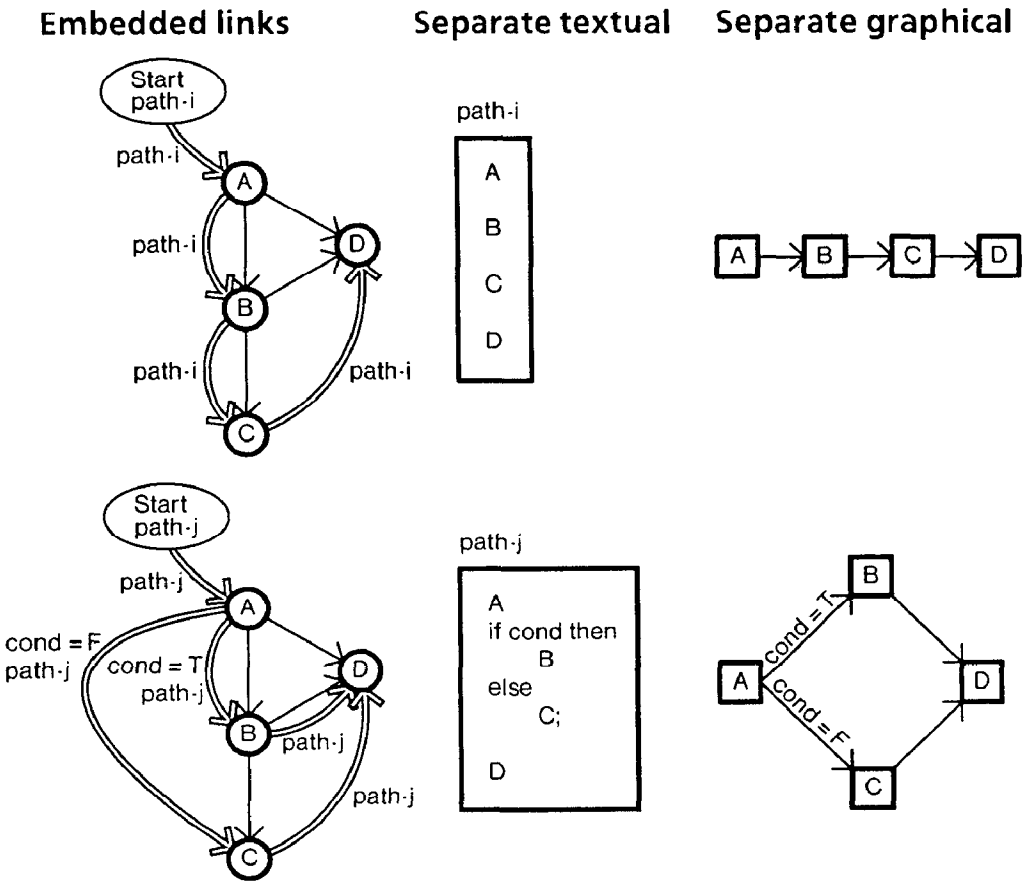

Figure 1. Alternative path representations. Three different representations of paths are shown for linear (top) and conditional (bottom) paths. On the left, circles represent hypertext nodes, while double-line arrows represent path links embedded in the hypertext network.

2.5 Path storage

Issues of where hypertext systems should store their links are currently being debated within the hypertext community. Paths present similar issues. A path mechanism can store its path links with other hypertext links, or it can store its path links separately. In either case, the information stored must support path playback and path visualization. Figure 1 displays two paths expressed in three ways: as embedded links, as a separate textual path, and as a separate graphical path.

Path links that are stored with other hypertext links must be distinguishable from them to permit playback. Furthermore, as the sequencing model increases in complexity, more information must be stored on each link to determine the next entry during playback. Although embedded links make it easy to discover what paths cross a given location, they do not directly support finding the start of a path.

When paths are stored separately, the path specification either names or refers to entries. These "forward links" are required to support path playback. In addition, to support path visualization, it is important to provide "back links" from the entries to the path

specification. The back links may be actual links, or they may be indices into a path database. If no back links are provided, readers cannot readily discover existing paths.

3. DESCRIPTION OF OTHER SYSTEMS WITH PATHS

Using the framework developed in the previous section, we now examine other systems that have provided paths.

Memex trails. Vannevar Bush argued convincingly that his memex should support *trails*, or sequences of links that scientists would construct and consult while working [Bush45]. Any location could appear in multiple trails; each trail acted in essence as a new book.

Textnet paths. Textnet paths are simply ordered lists of nodes; these are separately-stored paths that a user can modify by adding or deleting nodes from the list [Trig86]. Thus Textnet provides sequential paths whose entries are entire nodes. The contents of a path are shown to the user as a single linear concatenation of text from each node, with small markers at each node boundary, rather than as a temporally ordered sequence of visits to the nodes. Textnet paths are used to generate multiple different linear documents from a single hypertext network, especially as formatted hardcopy, although it is unclear how readers learn of the existence of alternate paths.

HyperCard scripts. HyperCard's HyperTalk language allows an author to construct programmable or even variable paths, for example as a button script for some card [Good87]. However, although HyperCard does provide the necessary raw functionality, it provides no tools for creating, maintaining, finding, or playing back paths as distinguished objects. An implementor would have to provide these separately. According to Trigg, Elli Mylonos has extended HyperCard with a path card that can store and play back a linear sequence of cards [Trig88].

Guided tours in NoteCards. NoteCards is an authoring or idea management system [Hala87]. Its path mechanism, called *guided tours*, was developed as a way of communicating the organization of a NoteCards filebox to other users [Trig88]. Guided tours provide branching paths whose entries are a special kind of node, namely spatially-organized collections of cards called *tabletops*. Tours store their path links as ordinary links of type *GuidedTour* among tabletop nodes. Each tabletop node can display its GuidedTour links on request, but individual nodes cannot tell if they are part of a tabletop node. Thus navigational support for maintaining tours is weak in this respect. Authors create and edit a graphical view of a guided tour. In fact, a tour need not have a single starting point. Readers have both single-stepping and browsing control over the resulting graph, but no automatic control. The reader is given visual history feedback in the form of highlighting on links and nodes that have been traversed in this session.

Hammond & Allinson's guided tours. Hammond and Allinson used a "travel holiday" metaphor to study a variety of navigation methods in educational materials [Hamm88]. For users unfamiliar with the material being taught, their experiments demonstrated the utility of conditional paths (also called *guided tours*) over maps, keyword indices and browsing . Each multimedia node (called a *frame*) can include text, graphics, sound, time-delayed items and animation. Their procedural, conditional paths permit sub-tours (called *excursions*) and branches based on reader response or prior activity. A user has only single-stepping control through a tour, although he or she can suspend the tour at any time and return to it later. Tours always return the reader to the starting point. Their local maps, which show the connectivity of some subpart of the network, are a completely separate mechanism. These maps are quite similar to GuidedTour cards in NoteCards, except that they provide only browsing control (i.e., no path interpreter).

Petri nets in αTrellis. Stotts and Furuta model a hypertext as a Petri net that defines all possible transitions through the hypertext [Stot88]. The strengths of this unconventional hypertext representation are that it can express prerequisite relationships between entries

well, and it can formally synchronize the simultaneous viewing of multiple locations. While it is true that a Petri net is a generalization of hypertext link structures, a Petri net is less powerful as a path mechanism than a programmable path. Furthermore, the ability of authors to learn to construct such hypertexts (which depends largely on what Petri net construction tools are provided) has not yet been shown, especially for the more complex Petri nets. Programs are familiar to more potential authors.

4. PATHS IN SCRIPTED DOCUMENTS

The Scripted Documents system extends the previous mechanisms to address several key issues: conditional and programmable paths; automated path playback; navigational support for authors and readers; active entries with a wide range of actions; and multiple media, including voice.

The primary goal of Scripted Documents is to explore the role of hypermedia sequencing and action in a variety of uses of online multimedia documents, including idea organization, interpersonal communication, user documentation, programming tasks, memoranda, and audio-visual presentations. Scripted Documents is a bit unusual as a hypertext system, because it provides paths as its sole linking mechanism. The Scripted Documents system is implemented in the Cedar programming environment [Swin86] at Xerox PARC. Scripted Documents runs on Dorado personal workstations [Lamp81] in a distributed environment connected by Ethernet.

This paper concentrates on the path mechanism available in Scripted Documents. An earlier paper provides many examples of scripts and discusses the object-oriented approach to underlying documents that permits smooth integration with existing editors [Zell88b]. A longer report also details the consistency operations needed to maintain scripts that traverse existing files that users continue to move, copy, and edit [Zell89].

4.1 Overview of Scripted Documents

Briefly, a script is an active directed path through one or more documents that need not follow the linear order of the documents. Each entry in a script consists of a document location, together with an associated action and timing. A script is typically played back with automatic control, displaying the location of the first entry and performing its action, then continuing automatically to the second entry, and so on.

As a simple example, consider the following use of scripts to review a manuscript. Each reviewer prepares a script for the manuscript, including voice annotations as well as branches through supporting reference documents. Each script follows an arbitrary path through the document to collect related points. The author plays the scripts back later to hear what the reviewers have said. This use provides much of the value of a face-to-face interaction between a reviewer and the author, in which the reviewer makes comments while flipping back and forth through the manuscript and other documents to substantiate those comments.

As a result of the efforts of the Etherphone project, recorded and synthesized voice are widely available in the Cedar environment [Zell88a]. A major motivating goal behind Scripted Documents was to be able to collect and order related voice annotations to form narrations of one or more documents. The essential sequentiality of narrations led directly to our exploration of paths.

4.2 Expressive power of scripts

Scripted Documents provides procedural, programmable paths with active entries, called *scripts*. A script has three parts:

- A set of *documents*, which are typically pre-existing files of various types, such as text, 2D or 3D illustrations, VLSI diagrams, database entries, or combinations thereof.
- A set of *script entries*, which associate locations or objects within those documents with actions and timing information. Entries can perform arbitrary actions, ranging from issuing system commands to manipulating variables via an interpreter. Sample actions might be to play back a previously-recorded voice annotation, send text to a text-to-speech synthesizer, animate a picture, or query a database. Although actions can be forked while the path continues, there are no script-level primitives for synchronization among the resulting forks.
- A *path specification*, also known as a *script header*, which is a program written in the Cedar language, a Pascal-like language that is the major programming language for the Cedar programming environment [Swin86]. These programs can be as simple or as complex as the author needs to express his or her sequencing constraints. A sequential script appears as a textual list of script entries. More complex scripts can call other scripts, add conditions or loops, declare variables, or manipulate the environment in an arbitrary way.

4.3 Creating and editing scripts

The author edits script headers and entries via form-based tools, as shown in Figure 2.

Script headers. Sequential paths can be created by direct manipulation: the author selects a document region and clicks a menu button in the script header tool. The system creates a new entry and adds it to the script at the current insert point, which is typically positioned at the end of the script. To create a programmable or procedural path, the author uses a text editor to add loops, conditionals, or calls to other scripts. The text editor is also used to copy entry names from other scripts and rearrange or delete entries. Although using a text editor is convenient for the author, it has drawbacks: it provides opportunities for creating scripts with syntactic errors or references to nonexistent entries or scripts. These errors are detected either by playing a script or by *verifying* it.

Script entries. The author specifies the action, timing or other fields of a script entry by filling in the fields of a script entry tool. The author can name a script entry or provide keywords, if desired. The system provides a unique identifier for each script entry and records the creator and create time. Narrative entries can be created without forms: the default action is to play back all voice annotations at the location, if any, and the default timing is to continue when the action completes. The system also provides a library of commands for interacting with the user to ask questions, and for manipulating windows and buttons on the display so that arbitrary applications can be controlled from a script.

4.4 Visualization and navigation support

Local visualization and navigation help authors maintain scripts by alerting them when changes to an underlying document might damage a script. They also help readers find scripts of possible interest. At the document content level, each scripted location is distinctively marked. Textual scripted locations are marked with a surrounding rectangle; other scripted objects may be marked differently. The marker indicates the presence of one or more script entries at that location, which may appear at multiple places in multiple scripts. Navigation commands display all scripts or script entries that include that scripted location.

In addition, an author who wishes to change an entry must be able to discover the associated document location and all scripts that the entry appears in. Similarly, authors

must be able to discover what scripts include calls to a given script. Menu commands are provided for these operations in script header and entry tools.

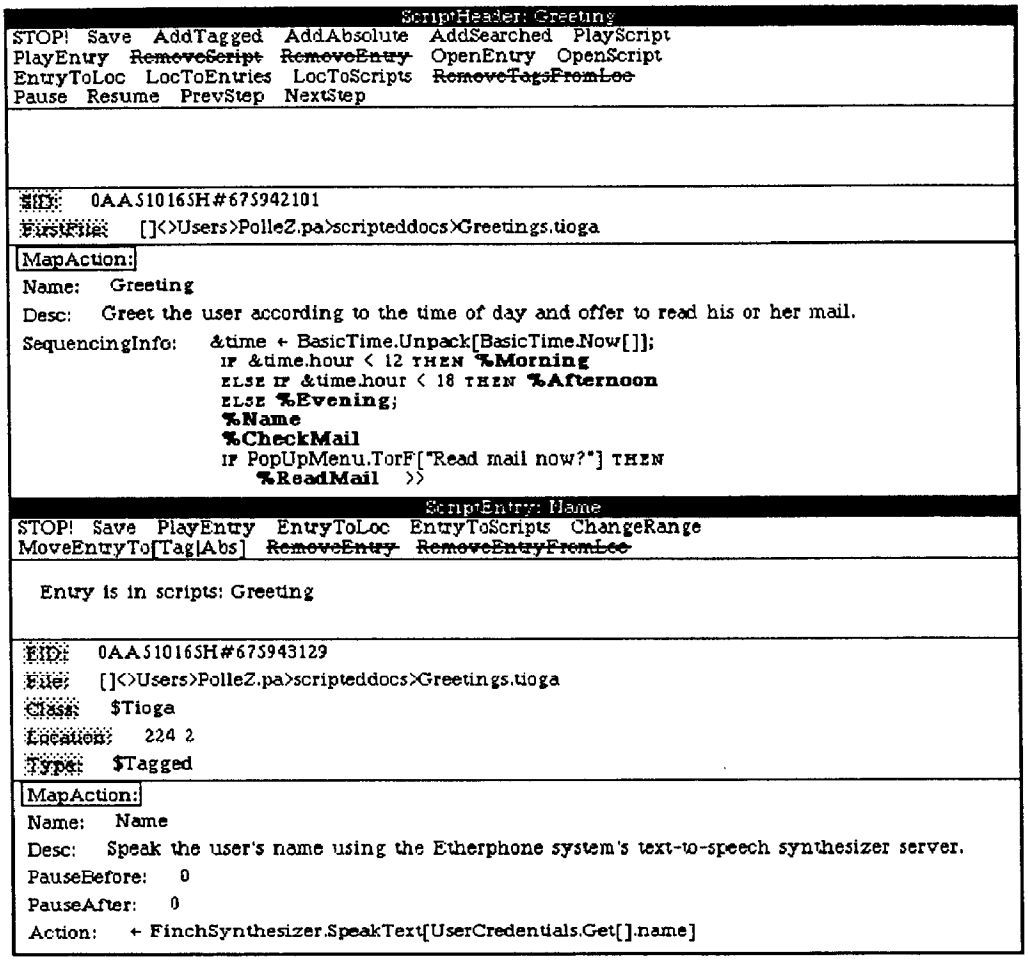

Figure 2. Script header and entry tools. Each tool has four parts: the top section has a menu of script operations, the second section is a feedback area, the third section displays read-only fields that the system manages, and the bottom section has user-editable fields. The upper window shows a script header tool that contains a programmable path. The path, displayed in the SequencingInfo: field, refers to entries as %<entry name> (translated from internal unique identifiers by the user interface). Sequential parts of the path appear as lists (e.g. %Name %CheckMail). Double brackets ">>" mark the current insert point. The lower window shows an entry tool for the Name entry.

Other methods for finding scripts are more global. The Scripts button in a document header displays a list of all scripts that start in or cross that document and allow the user to create a script header to edit or play back a script. A user can also access a script by name or by browsing the script database for keywords, descriptions, or other script fields.

Assorted visualization mechanisms help to orient the user. Because script headers show a textual program representation of a path, users can get a feel for how long the script is (at least in steps, if not in time), how complex it is, and what entries it visits. The system also displays playback feedback: the entry currently being executed is highlighted in the script header. This provides a valuable percent-done indicator [Myer85]. Finally, the system provides a form of dynamic location visualization: the user can inhibit the

execution of the actions and play the script, which will then traverse all of its locations at a controllable speed.

Figures 3 and 4 illustrate several visualization and navigation mechanisms in Scripted Documents.

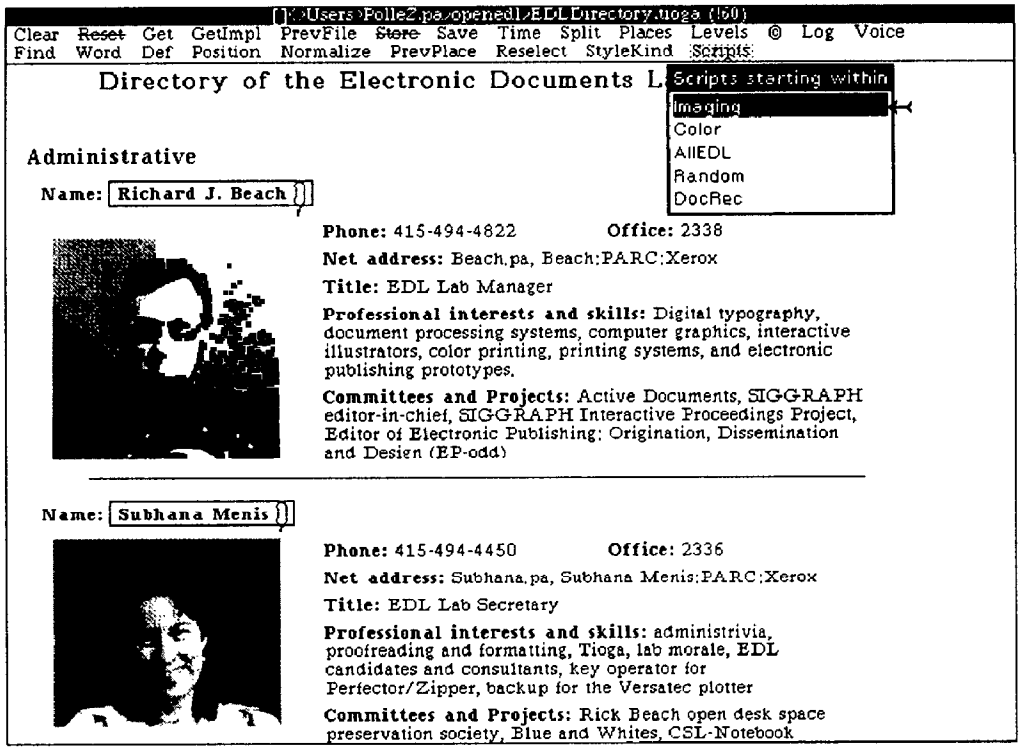

Figure 3. Finding scripts. This multimedia document contains color photos (reproduced here in black and white), formatted text, and voice annotations describing the staff of PARC's Electronic Documents Laboratory. Voice annotations are indicated by small "voice balloons" after each person's name. Several scripts traverse the document to show members of various projects and play back their voice descriptions of themselves; some people are on multiple projects. The user has just clicked the Scripts button in the document header, which has created a menu of scripts that start in this document. Script entries are shown as boxes around text.

4.5 Playback control

Scripted Documents provides rich playback control: automatic control, single-stepping control and browsing control. The PlayScript command plays an entire script from the beginning, proceeding automatically from one location to the next, executing the associated actions with the associated timings. This feature is particularly useful for narrations, presentations, and document processing situations. Readers can also pause, resume, stop, or single-step forward or backward (which consults a history of the current execution). In addition, readers can browse a script by selecting and playing individual entries. As discussed earlier, active entries and programmable scripts make browsing control potentially dangerous. Scripted Documents does not currently protect readers from executing entries without their prerequisites.

The Scripted Documents system does not take over the entire screen. Users can perform other tasks simultaneously while interacting with Scripted Documents. Unfortunately,

4.6 Script storage

Script entries and headers are stored separately from the documents, in a simple database. This design has several advantages.

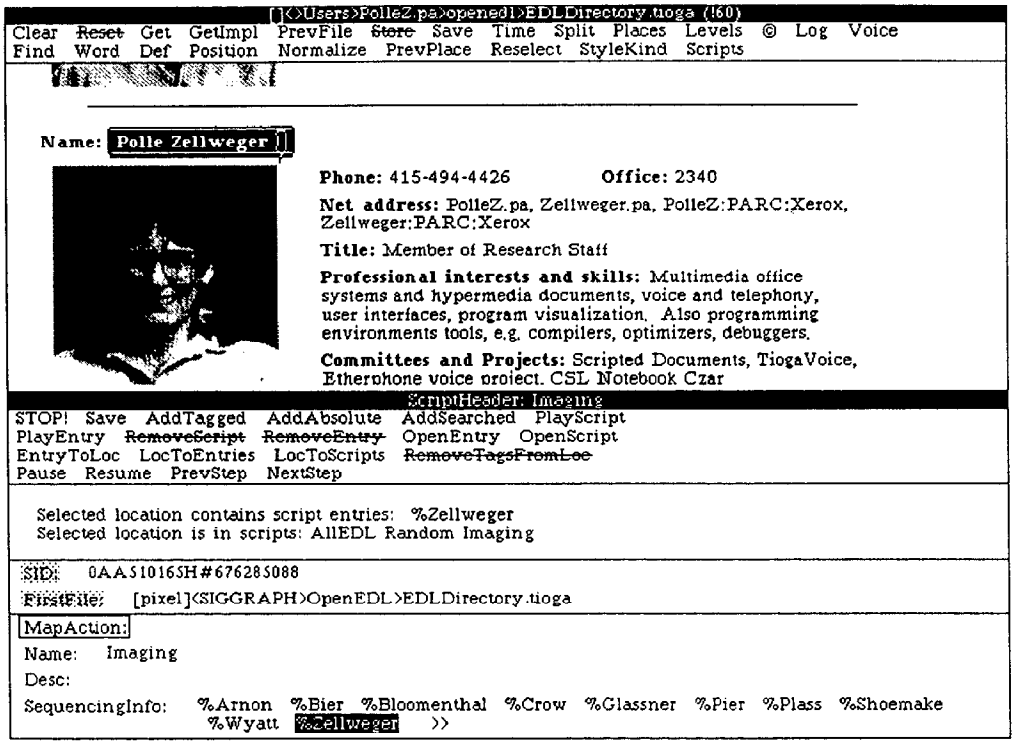

Figure 4. Script navigation and playback. These two windows show the playback of the last entry in the sequential Imaging script. In the document in the upper window, the location of the Zellweger script entry is highlighted and the voice annotation is being played. The lower window shows the script's header tool (created by the Scripts button click in Figure 3). To indicate playback progress, the Zellweger entry is also highlighted in the SequencingInfo: field. In addition, the tool's feedback area shows the result of two earlier navigational queries regarding the text "Polle Zellweger" in the document.

First, separating script entries from script headers allows multiple actions or timings to be associated with the same document location, even in a purely sequential script. Entries can also be re-used in multiple scripts.

Second, separating script information from documents allows existing documents to be involved in scripts without requiring authors to copy information into a closed hypertext system. Aside from the inconvenience of an extra copying step, copying presents two other problems: it removes the information from its original context (within the document and/or within the file system), and it requires authors to maintain both copies across future changes.

Finally, storing script information in a database provides rich query capabilities to support visualization and navigation aids for script users. The default script database resides on a centralized server to permit sharing scripts among users. Users can also specify private databases to replace or supplement the public database.

4.7 Experience with scripts

We have created educational scripts consisting of multiple narrations for language lessons; scripts that provide narrative reviews of manuscripts; several versions of slide presentations for short, medium and long talks; and user documentation that demonstrates the execution of an application on examples.

Scripts have also proven useful in software engineering tasks, such as task management, debugging, system documentation, and testing. Scripts can mark unfinished routines or questionable sections that should be re-examined. They can also collect groups of program locations to form sets of potential breakpoints that will be automatically maintained by the system throughout the program's lifecycle. Breakpoints can be set or cleared at a given group of locations by playing back the appropriate script. Script narrations can provide system documentation, and scripts can execute software tests and record results and timing information.

4.8 Future work

Although Scripted Documents has already demonstrated the power of scripts, much work remains to be done. Scripts currently have no versioning mechanism, they provide only minimal support for collaboration and sharing among users, and they do not support the synchronization of multiple simultaneous branches along a path. In addition, we would like to provide better visualization of scripts for both script readers and writers, including browsing tools and visual displays of script sequencing (such as a graphical document-level view of a script's path, or a view that embeds the entries' actions in the path specification program).

5. CONCLUSIONS

Simple sequential paths can collect locations, order them (such as for printing), or perform sequential activities. Indeed, if authors provide enough sequential paths, the resulting structures begin to approximate the computational power of conditional or branching paths. However, the issues involved in managing and presenting to the reader a large forest of paths argues strongly for fewer, more powerful paths.

Paths are particularly valuable when combined with multiple media and actions, as in the Scripted Documents system. Attaching audio actions to visual entries provides authors with easily understood synchronization between audio and visual material.

Paths should become first-class citizens of their hypertext systems. When a reader is lost in hypertext, it is important to be able to recognize nearby paths. Maps or overviews of paths, particularly situated overviews (such as thumbnails or a similar representation), are also helpful.

Finally, paths provide expressive power to authors. Complex paths allow great flexibility in creating presentations specifically tuned to readers, thus reducing their cognitive load. These paths will not be trivial to create, but with the appropriate tools the creative effort will continue to outweigh the technical effort.

ACKNOWLEDGMENTS

Cecelia Buchanan implemented the programmable paths in Scripted Documents, as well as the user interaction library. Rok Sosic improved the playback control. Other Etherphone project members, including Dan Swinehart, Doug Terry, and Stephen Ades, contributed to the voice substrate and helped hone the script concept. Jock Mackinlay and Rick Beach improved the organization and presentation of this paper.

REFERENCES

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

[Conk87a] Conklin, J. Hypertext: an introduction and survey. *IEEE Computer*, **20**, 9, (September 1987), 17-41.

[Conk87b] Conklin, J. *A survey of hypertext*. MCC Technical Report STP-356-86, Rev. 2, December 1987.

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

[Hamm88] Hammond, N. & Allinson, L. Travels around a learning support environment: rambling, orienteering or touring? *Proc. ACM CHI'88 Conf.*, Washington, DC, May 15-19, 1988, 269-273.

[Hala87] Halasz, F., Moran, T. & Trigg, R. NoteCards in a nutshell. *Proc. ACM CHI+GI'87 Human Factors in Computing Systems and Graphics Interface Conf.*, Toronto, Canada, April 5-9, 1987, 45-52.

[Hala88] Halasz, F. Reflections on NoteCards: seven issues for the next generation of hypermedia systems. *CACM*, **31**, 7, (July 1988), 836-852.

[Myer85] Myers, B. The importance of percent-done progress indicators for computer-human interfaces. *Proc. ACM CHI'85 Conf.*, San Francisco, CA, April 14-18, 1985, 11-17.

[Lamp81] Lampson, B. and Pier, K.; Lampson, B., McDaniel, G., & Ornstein, S.; Clark, D., Lampson, B., & Pier, K. *The Dorado: a high performance personal computer, three papers*. Xerox PARC Report CSL-81-1, 1981.

[Stot88] Stotts, P.D. & Furuta, R. Adding browsing semantics to the hypertext model. *Proc. ACM Document Processing Systems Conf.*, Santa Fe, NM, December 5-9, 1988, 43-50.

[Swin86] Swinehart, D., Zellweger, P., Beach, R. & Hagmann, R. A structural view of the Cedar programming environment, *ACM Trans. Prog. Lang. and Syst.*, **8**, 4, (October 1986), 419-490.

[Trig86] Trigg, R. & Weiser, M. TEXTNET: a network-based approach to text handling. *ACM Trans. Office Info. Syst.* **4**, 1, (January 1986), 1-23.

[Trig88] Trigg, R. Guided tours and tabletops: tools for communicating in a hypertext environment. *ACM Trans. Office Info. Syst.* **6**, 4, (October 1988), 398-414.

[Yank88] Yankelovich, N., Haan, B., Meyrowitz, N., and Drucker, S. Intermedia: The concept and the construction of a seamless information environment, *IEEE Computer*, **21**, 1, (January 1988), 81-96.

[Zell88a] Zellweger, P., Terry, D. & Swinehart, D. An overview of the Etherphone system and its applications, *Proc. 2nd IEEE Computer Workstations Conf.*, Santa Clara, CA, March 8-10, 1988, 160-168.

[Zell88b] Zellweger, P. Active paths through multimedia documents. In J.C. van Vliet (editor), *Document Manipulation and Typography*, Proc. Int'l Conference on Electronic Publishing, Document Manipulation and Typography, Nice (France), April 20-22, 1988. Cambridge University Press, 1988, 19-34.

[Zell89] Zellweger, P. *Scripted Documents.* Xerox PARC Report, in preparation.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1989 ACM 089791-339-6/89/0011/0014 $1.50