Palo Alto Research Center

Etherphone: Collected Papers 1987-1988

An Overview of the Etherphone System and its Applications
by Polle T. Zellweger, Douglas B. Terry, and Daniel C. Swinehart

Telephone Management in the Etherphone System
by Daniel C. Swinehart

Managing Stored Voice in the Etherphone System
by Douglas B. Terry and Daniel C. Swinehart

System Support Requirements for Multi-media Workstations
by Daniel C. Swinehart

Active Paths through Multimedia Documents
by Polle T. Zellweger

XEROX

Etherphone: Collected Papers 1987-1988

CSL-89-2 May 1989 [P89-00002]

Abstract: The Etherphone™ system integrates live and recorded voice into an office workstation environment. It supports a wide range of applications, including telephony, voice mail, voice annotation and editing, audio user interfaces, audio meeting services, and narrated hypermedia documents. This report brings together a group of previously-published papers that describe these applications and the distributed voice system architecture on which they are built. It includes an overview of the Etherphone system; a vision of the role that workstation-based telephones can play in an office environment; a discussion of the voice manager that provides facilities for recording, editing, and playing stored voice; an outline of the systems support required to permit flexible, extensible multimedia applications; and a description of a hypermedia presentation system built upon the capabilities of the Etherphone system.

An Overview of the Etherphone System and its Applications, by Polle T. Zellweger, Douglas B. Terry, and Daniel C. Swinehart. This paper appeared in the *Proceedings of the 2nd IEEE Conference on Computer Workstations* (Santa Clara, CA, March 1988), 160-168. An earlier version appeared as: D. Swinehart, D. Terry, and P. Zellweger. An experimental environment for voice system development. *IEEE Office Knowledge Engineering Newsletter*, **1**(1), February 1987, 39-48.

Telephone Management in the Etherphone System, by Daniel C. Swinehart. This paper appeared in the *Proceedings of the IEEE/IEICE Global Telecommunications Conference* (Tokyo, November 1987), 1176-1180.

Managing Stored Voice in the Etherphone System, by Douglas B. Terry and Daniel C. Swinehart. A version of this paper appeared in *ACM Transactions on Computer Systems* **6**(1), February 1988, 3-27.

System Support Requirements for Multi-media Workstations, by Daniel C. Swinehart. This paper appeared in the *Proceedings of the SpeechTech '88 Conference* (New York; April 1988); Media Dimensions, Inc., New York, April 1988, 82-83.

Active Paths through Multimedia Documents, by Polle T. Zellweger. This paper appeared in *Document Manipulation and Typography*, J.C. van Vliet (ed.), Cambridge University Press, 1988. Proceedings of the EP'88 Conference on Electronic Publishing, Document Manipulation and Typography, (Nice, France; April 1988).

XEROX

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

An Overview of the Etherphone System and Its Applications

Polle T. Zellweger, Douglas B. Terry, and Daniel C. Swinehart

Abstract: The Etherphone™ system has been developed to explore methods for extending existing multi-media office environments with the facilities needed to handle the transmission, storage, and manipulation of voice. Based on a hardware architecture that uses microprocessor-controlled telephones to transmit voice over an Ethernet that also supports a voice file server and a voice synthesis server, this system has been used for applications such as directory-based call placement, call logging, call filtering, and automatic call forwarding. Voice mail, voice annotation of multi-media documents, voice editing using standard text editing techniques, and applications of synthetic voice use the Etherphones for voice transmission. Recent work has focused on the creation of a comprehensive voice system architecture, both to specify programming interfaces for custom uses of voice, and to specify the roles of different system components, so that equipment from multiple vendors could be integrated to provide sophisticated voice services.

This paper appeared in the *Proceedings of the 2nd IEEE Conference on Computer Workstations* (Santa Clara, CA, March 1988), 160-168. An earlier version appeared as: D. Swinehart, D. Terry, and P. Zellweger. An experimental environment for voice system development. *IEEE Office Knowledge Engineering Newsletter*, 1(1), February 1987, 39-48.

CR Categories and Subject Descriptors: C.2.4 **[Computer-Communication Networks]**: Distributed Systems – *distributed applications*, C.3 **[Special-Purpose and Application-Based Systems]**: Real-time Systems, D.2.6 **[Software Engineering]**: Programming Environments, D.4.7 **[Operating Systems]**: Organization and Design – *hierarchical design; distributed, real-time, and interactive systems*, H.1.2 **[Models and Principles]**: User/Machine Systems – *human factors*, H.4.1 **[Information Systems Applications]**: Office Automation, H.4.3 **[Information Systems Applications]**: Communications Applications, I.7.2 **[Text Processing]**: Document Preparation.

General Terms: Design, experimentation, human factors.

Additional Keywords and Phrases: Etherphones, Cedar, telephones, recorded voice, voice system architecture, workstation telephone management, multimedia conferencing, collaborative work, voice editing, voice-annotated documents, multimedia documents, active documents.

XEROX

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

AN OVERVIEW OF THE ETHERPHONE SYSTEM AND ITS APPLICATIONS*

Polle T. Zellweger, Douglas B. Terry, and Daniel C. Swinehart

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

Abstract

The Etherphone™ system has been developed to explore methods for extending existing multi-media office environments with the facilities needed to handle the transmission, storage, and manipulation of voice. Based on a hardware architecture that uses microprocessor-controlled telephones to transmit voice over an Ethernet that also supports a voice file server and a voice synthesis server, this system has been used for applications such as directory-based call placement, call logging, call filtering, and automatic call forwarding. Voice mail, voice annotation of multi-media documents, voice editing using standard text editing techniques, and applications of synthetic voice use the Etherphones for voice transmission. Recent work has focused on the creation of a comprehensive voice system architecture, both to specify programming interfaces for custom uses of voice, and to specify the roles of different system components, so that equipment from multiple vendors could be integrated to provide sophisticated voice services.

1. Introduction

Suppose Alexander Graham Bell had waited to invent the telephone until personal workstations and distributed computing networks had been invented. What approach would he take in introducing voice communications into the modern computing environment? It was an attempt to answer this question that led to the creation of a voice communications project within the Computer Science Laboratory at Xerox PARC.

Stated more concretely, the project's aim was to extend our existing multi-media office environment with the facilities needed to handle the transmission, storage, and manipulation of voice. We believed that it should be possible to deal with voice as easily as we can manage text, electronic mail, or images. The desired result was an integrated workstation that could satisfy nearly all of a user's communications and computing needs.

A basic requirement was to provide conventional telephone facilities (so that casual users would not have to read a manual to make a phone call), but our goals went well beyond that. We had observed that most enhanced voice communications facilities had been developed by designers of telephone systems. In contrast, we wished to draw on our experience as developers of personal information systems running on powerful workstations with graphical interfaces. We were convinced that the user would prefer to perform voice management functions using the power and convenience of workstation facilities such as on-screen menus, text editors, and comprehensive informational displays.

These aims led us to explore two related research domains:

- *"Taming the telephone":* Despite an immense investment in research and development over the last 110 years, the user interface and the functionality of the telephone still leaves much to be desired. We contend that the personal workstation, combined with a telephone system whose characteristics we can control, make it possible to better match the behavior of the office telephone with the needs of its users. There are gains to be had in the placement of calls, the handling of incoming calls, and the capabilities available to telephone attendants (that is, switchboard operators, receptionists, and secretaries).
- *"Taming the tape recorder":* We also believe that workstation techniques for creating, editing, and

*An earlier version of this paper appeared as: D. Swinehart, D. Terry, and P. Zellweger. An experimental environment for voice system development. *IEEE Office Knowledge Engineering Newsletter 1*, 1, February 1987, 39-48.

storing text or images can be modified to deal with digitally-recorded voice. Application areas such as electronic mail, document annotation, and dictation are candidates for improvement. Speech synthesis and recognition devices can be added to provide translation between textual and spoken information.

These two sets of activities are clearly related. A carefully designed system can support novel applications of both live and recorded voice.

In this overview we will describe the Etherphone™ system that we have developed and used to explore the integration of voice into a personal information environment. The following sections sketch the present hardware architecture, describe some of the more compelling applications that have been built to exploit it, and briefly explore the software and systems issues that have surfaced.

2. Etherphone System Description

In designing our prototype voice system, we surveyed several possible hardware architectures, including extensions of our existing Centrex service or of a commercially available PABX. Primarily because Centrex and PABX systems did not support programmable switching control, we concluded that the most effective way to satisfy our needs was to construct our own transmission and switching system[17]. Ethernet local-area networks, which provide the data communications backbone supporting personal distributed computing at Xerox PARC, have proven to be an effective medium for transmitting voice as well as data.

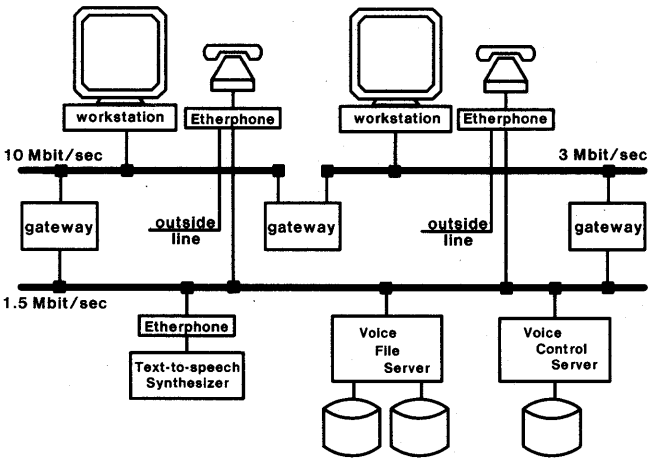


Figure 1. Etherphone system components.

Our prototype voice system consists of the following types of components connected by Ethernets, as shown in Figure 1:

Etherphones: telephone/speakerphone instruments, each including a microcomputer, encryption hardware, and an Ethernet controller. Etherphones digitize, packetize, and encrypt telephone-quality voice (64 kilobits/second, with silence suppression) and send it to each other directly over an Ethernet; they support conferencing by digitally summing the voice packets from other participants before converting to analog. Etherphone software is written in C. The current environment contains approximately 50 Etherphones, which are used daily by members of the Computer Science Laboratory as their only telephone service. Each Etherphone includes a connection to a standard direct-dial telephone line for access to telephones outside the Etherphone system.

We chose to separate the voice-processing functionality from the workstation for reliability, performance, and flexibility: voice-processing peripherals allowed us to provide reliable voice processing without compromising workstation performance on other tasks and to add voice to different types of workstations without changing the voice hardware. Etherphones transmit voice on a separate 1.5 megabits/second Ethernet because the only Ethernet chips available to us in 1981 could not operate at standard speeds. Tests and calculations convinced us that our existing Ethernets would provide adequate bandwidth and service to carry voice as well as data. Additional information on the Etherphone hardware and the Voice Transmission Protocol can be found in a previous report[17].

Voice Control Server: a program that provides control functions similar to a conventional PABX and manages the interactions among all the other components. It runs on a dedicated server that also maintains databases for public telephone directories, Etherphone-workstation assignments, and other shared information. The Voice Control Server is programmed in the Cedar programming environment[18]. Centralized server software limited the necessary size and speed of the Etherphone processor, and thus its cost, as well as limiting the amount of C software.

Voice File Server: a service that can hold conversations with Etherphones in order to record or play back user utterances. In addition to managing stored voice in a special-purpose file system, the Voice File Server provides operations for combining, rearranging, and replacing parts of existing voice recordings to create new voice objects. For security reasons, voice remains encrypted when stored on the file server.

Text-to-speech Server: a service that receives text strings and returns the equivalent spoken text to the user's Etherphone. Each text-to-speech server is constructed by connecting a commercial text-to-speech synthesizer to a dedicated Etherphone and is available on a first-come-first-served basis. Two speech synthesizers, purchased from different manufacturers, have been installed.

Workstations: high-performance personal computers with large bitmapped displays and mouse pointing devices. Workstations are the key to providing enhanced user

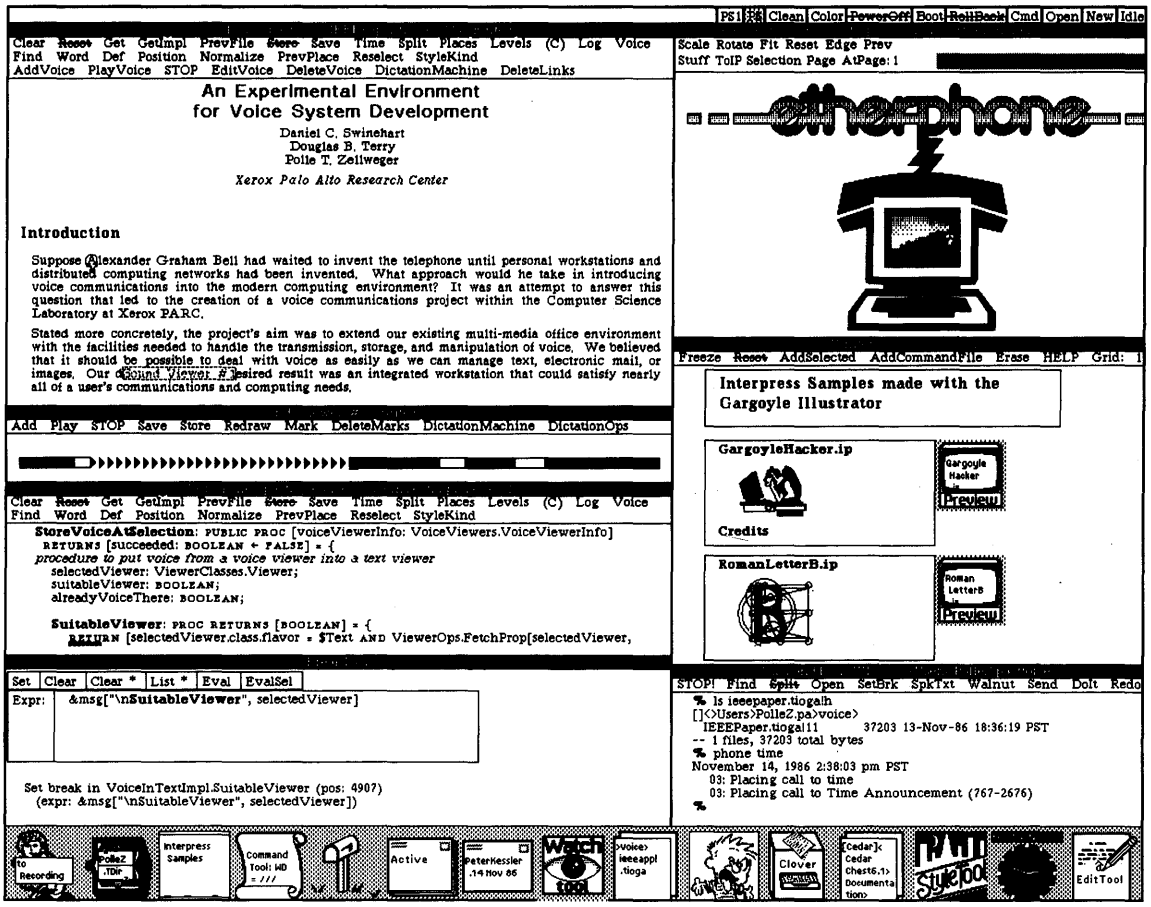


Figure 2. A Cedar screen in use. The two windows in the upper left show a document preparation task, including voice annotation of an earlier version of this paper for communication among the authors. The bottom window of this pair shows new voice (represented by arrowheads) being inserted in the middle of an existing voice annotation. The two windows in the lower left show a programming task that is monitoring part of the voice annotation system. The two windows in the upper right show images created with several graphical illustration packages. The command interpreter window at the lower right accepts user commands, similar to a Unix shell. The bottom row of icons contains files and tools that are active but are not currently being manipulated by the user.

interfaces and control over the voice capabilities. We rely on the extensibility of the local programming environment—be it Cedar, Interlisp, Unix, or whatever—to facilitate the integration of voice into workstation-based applications. Workstation program libraries implement the client programmer interface to the voice system.

In addition, the architecture allows for the inclusion of other specialized sources or sinks of voice, such as speech recognition equipment or music synthesizers.

All of the communication required for control in the voice system is accomplished via a remote procedure call (RPC) protocol[3]. For instance, conversations are established between two or more parties (Etherphones, servers, and so on) by performing remote procedure calls to the Voice Control Server. During the course of a conversation, RPC calls emanating from the Voice Control Server inform participants about various activities concerning the conversation. Active parties in a conversation use the Voice Transmission Protocol for the actual exchange of voice. Multiple implementations of the RPC mechanisms permit the integration of workstation programs and voice applications programmed in different environments.

3. Examples of Applications to Date

Most of our user-level applications to date have been created in the Cedar environment, although limited functions have been provided for Interlisp and for standalone Etherphones. This section describes the voice applications that are currently available in Cedar, including telephone management, text-to-speech synthesis, and voice annotation and editing. Figure 2 shows a typical Cedar screen using voice, text, and graphics to support programming and document preparation activities.

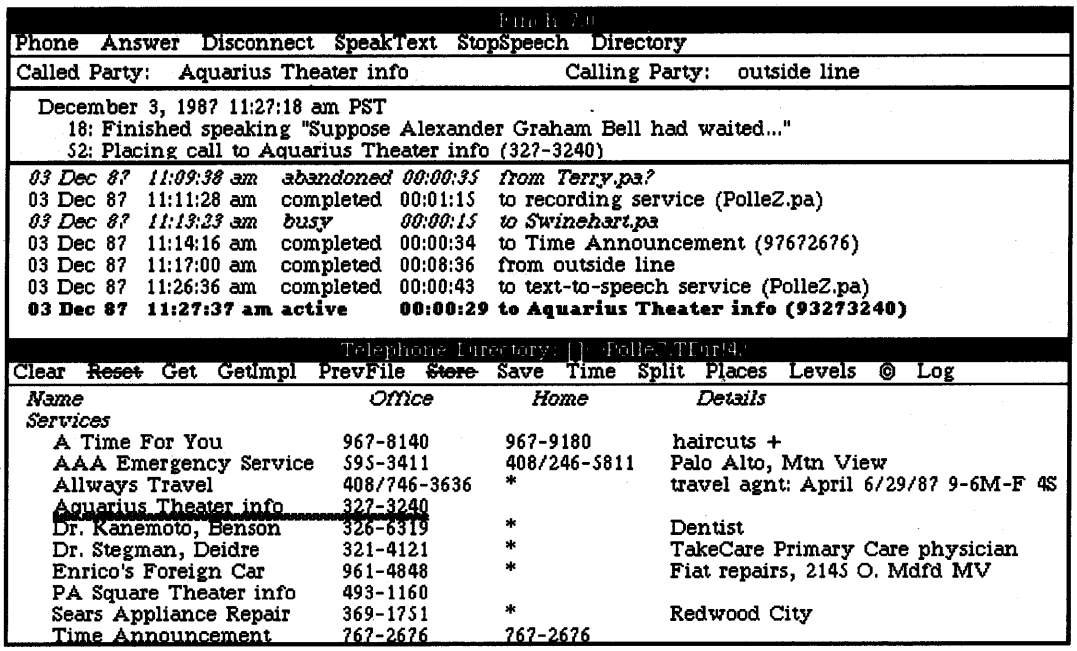


Figure 3. Two workstation telephone management windows. The upper Finch window provides a two-dimensional user interface to the Etherphone system. It includes an Etherphone control menu (the first line, including 'Phone', 'Answer', etc. buttons), a redialing area (the second line), an area for system status reports, and a conversation log (indicating a call in progress to Aquarius Theater info). The lower window shows a portion of a personal telephone directory, which is a set of speed-dialing buttons that can be created easily from an ordinary text file. The call in progress was placed by clicking on the underlined 'Aquarius Theater info' entry.

In order to make voice a first-class citizen of the Cedar environment, Etherphone functions are typically available in several ways: through an Etherphone control panel, through commands that can be issued in a command interpreter window, and through procedures that can be invoked from client programs. This integration of voice capabilities will be discussed more fully in the next section.

3.1. Telephone management

The telephone management functions provide improved capabilities for placing and receiving calls. Figure 3 shows an Etherphone control window, called *Finch*, and a personal telephone directory window.

Users can place calls by specifying a name, a number, or other attributes of the called party. A system directory database for local Xerox employees (about 1000 entries) is stored on the Voice Control Server. Users can also create personal directories, which are consulted before the system directory to locate the desired party. A soundex search mechanism[8] helps compensate for spelling uncertainties.

A variety of convenient workstation dialing methods are provided: a user can fill in fields in the Finch tool, select names or numbers from anywhere on the screen, use either of two directory tools that present browsable lists of names and associated telephone numbers as speed-dialing buttons, or redial any previously-made call by clicking on its conversation log entry. Calls can also be placed by name or number from the telephone keypad.

Calls are announced audibly, visually, and functionally. Each Etherphone user selects a personalized ring tune, such as a few bars of "Mary Had a Little Lamb". This tune is played at a destination Etherphone to announce calls to that user. The caller hears the same tune as a ringback confirmation. During ringing, the telephone icon jangles with a superimposed indication of the caller's name, as shown in the bottom portion of Figure 4. An active conversation is

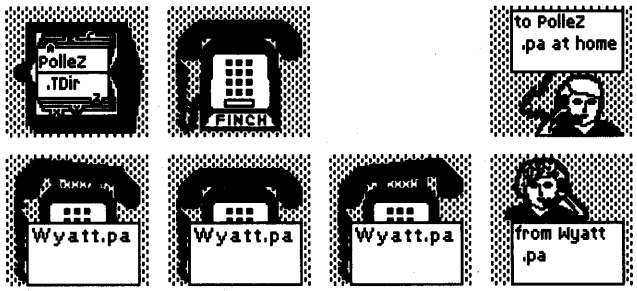


Figure 4. Etherphone system icons. The two icons at the upper left show a closed personal telephone directory and a Finch icon at rest. The icon at the upper right shows an outgoing call to Polle Zellweger's home (username PolleZ.pa). The four bottom icons show several stages of an incoming call from Doug Wyatt: the three left icons of the group are animated during ringing, while the right conversation icon is used after the call has been answered. Animation and visual feedback in the icons provide useful information without consuming valuable screen area.

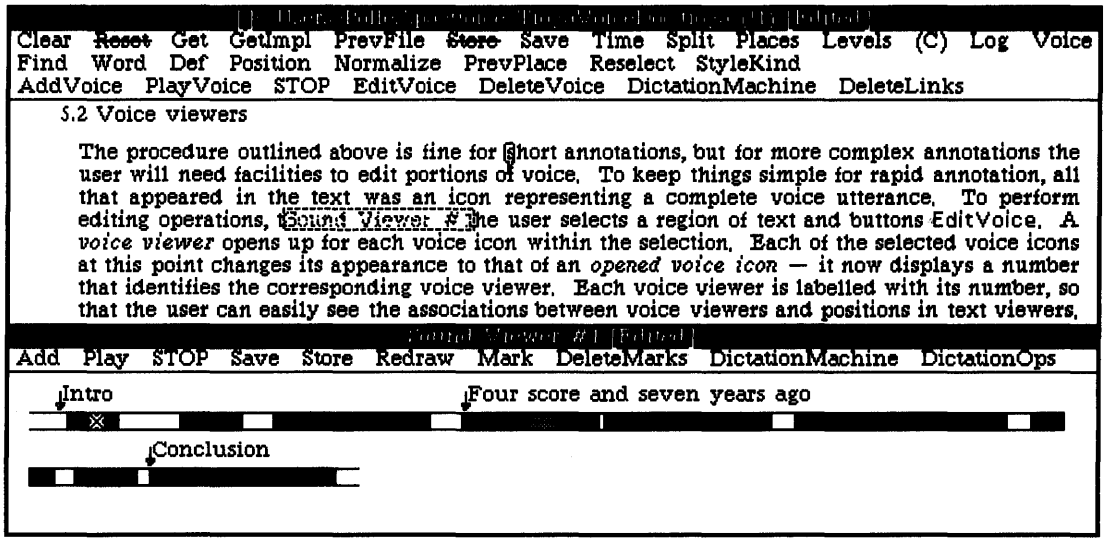


Figure 5. Voice annotation and editing. The upper window shows voice annotations being added to a Tioga document (the voice annotation system documentation). The third line of menu buttons ("AddVoice PlayVoice ...") near the top of the window are used to manipulate voice. In the second line of text, a voice annotation appears on the word "short", indicated by the comic-strip dialog 'balloon' surrounding the character. In the fifth line, a similar annotation has been opened for editing in the lower window, labelled "Sound Viewer #1". In the sound-and-silence profile in the lower window, white rectangles depict silence, while dark rectangles depict sound. The profile contains several contextual indicators to orient the user during editing. The playback cue (the gray rectangle underneath the word "score") indicates the progress of voice playback. A temporary marker in the form of a small cross has been placed in the voice section marked "Intro". The textual annotations with arrows are permanent markers that will be stored with the document.

represented as a conversation between two people with a superimposed indication of the other party's name (also shown in Figure 4). The system automatically fills in the Finch tool's Calling Party or Called Party field to allow easy redialing of the last call. It also creates a new entry in a conversation log. A user can consult the conversation log to discover who called while he was out of the office.

Our methods of locating users in an office building utilize the personalized ring tunes, which allow Etherphone users to identify calls to them wherever they may be: in their own offices, within earshot, or at other Etherphones. If an Etherphone user logs in at a workstation, his calls can be automatically forwarded to the adjacent Etherphone. An additional feature, called *visiting*, allows him to register his presence with a second workstation or Etherphone, such as during a meeting. Registering with the destination location allows users to travel more freely than forwarding calls from the home location does. Each visit request cancels any earlier requests. The common problem of forgetting to cancel forwarding is eased by ringing both Etherphones during visiting.

A more detailed series of examples that demonstrate how the Etherphone system's telephone management capabilities can increase office productivity appears in a related paper[16].

3.2. Text-to-speech synthesis

A user or program can generate speech as easily as printing a message on the display by using one of the Text-to-speech Servers. A user can select text in a display window and click the Finch tool's SpeakText menu button. A program can call a procedure with the desired text as a parameter. These features are implemented by creating a "conversation" between the user's Etherphone and a Text-to-speech Server. The system sets up a connection to the Text-to-speech Server, sends the text (via RPC), returns the digitized audio signal (via the Voice Transmission Protocol), and closes the connection when the text has been spoken. A similar mechanism is used for voice recording and playback.

Our primary uses for text-to-speech so far have been in programming environment and office automation applications. Programming environment tasks have included spoken progress indicators, prompts, and error messages. Office automation applications have included proofreading (especially comparing versions of a document when one version has no electronic form, such as proofing journal galleys) and audio reminder messages generated by calendar programs.

3.3. Voice annotation and editing

This section describes the addition of a voice annotation and editing mechanism to *Tioga*, the standard text-editing program in Cedar. More information about this system can be found in an earlier paper[2].

Tioga is a high-quality galley editor, supporting the creation of text documents that contain a variety of type faces, type styles, and paragraph formats as well as illustrations and scanned images. Tioga is the underlying editor for all textual applications in Cedar, including the electronic mail system,

the system command interpreter, and other tools that require the user to enter and manipulate text. Wherever Tioga is used, all of its formatting and multi-media facilities are available. Thus, by adding voice annotation to Tioga, we have made it available to a variety of Cedar applications. Figure 5 shows a closeup of a Tioga document containing voice annotations, one of which is being edited by the user.

Any text character within a Tioga document can be annotated with an audio recording of arbitrary length. The user interface of the voice annotation system is designed to be lightweight and easy to use, since spontaneity in adding vocal annotations is essential. Voice within a document is shown as a distinctive shape superimposed around a character, so that the document's visual layout is unaffected. Furthermore, adding voice to a document does not alter its contents as observed by other programs (such as compilers).

To add an annotation, the user simply selects the desired character within a text window and buttons AddVoice in that window's menu. Recording begins immediately, using either a hands-free microphone or the telephone handset, and continues until the user buttons STOP. A voice annotation becomes part of the document, although the voice data physically resides on the Voice File Server. Copies of the document may be stored on shared file servers or sent directly to other users as electronic mail. To listen to voice, a user selects a region containing one or more voice icons and buttons PlayVoice.

Simple voice editing is available on a visual representation of the voice. Users can select a voice annotation and open a window showing its basic sound-and-silence profile. Sounds from the same or other voice windows can be cut and pasted together using the same editing commands supported by the Tioga editor. Editing is done largely at the phrase level, representing the granularity at which we believe editing can be done with best results and least effort for an office situation*. A lightweight 'dictation facility' that uses a record/stop/backup model can be used to record and incorporate new sounds conveniently. The dictation facility can also be used when placing voice annotations directly into documents.

Sound-and-silence profiles alone do not supply adequate contextual information for users to identify desired editing locations, so several contextual aids are provided. A playback cue moves along the voice profile during playback, indicating exactly the position of the voice being heard. While playback is in progress, a user can perform edits immediately or mark locations for future edits. Simple temporary markers can be used to keep track of important boundaries during editing operations, while permanent textual markers can be used to mark significant locations within extended transcriptions. Finally, the system provides a visual indication of the voice-editing history in an editing window. Newly-added voice appears in a bright yellow color, while less-recently-added phrases become gradually darker as new editing operations occur. This use of color is similar to Lippman *et al*'s use of color in text editing[9].

The voice annotation facility is implemented on top of a flexible set of voice editing and management primitives[20]. Instead of rearranging the contents of edited voice files, the Voice File Server builds a data structure to represent the edited voice and stores it in a server database. This data structure, called a *voice rope*, consists of a list of intervals within voice files. Voice is included in documents, electronic mail, and client programs solely by reference to voice ropes. To promote sharing voice throughout our distributed personal computing environment, voice ropes are immutable: recording and editing produce new voice ropes rather than modify existing ones. A modified style of reference counts enables unwanted voice ropes to be garbage collected.

4. Progress toward a Voice System Architecture

The original goals of the Etherphone project were to produce experimental prototypes that could "tame the telephone" or "tame the tape recorder" in novel and useful ways. As the project developed, however, a more fundamental goal emerged: to create and experimentally validate a comprehensive architecture for voice applications. The best way to explain the value of a voice architecture is to list some of the properties it should have:

- *Completeness.* It must be able to specify the role of telephone transmission and switching, workstations, voice file servers, and other network services in supporting all the kinds of capabilities we have identified, such as telephone services and recorded voice services.
- *Programmability.* It must permit workstation programmers to modify existing voice-related applications and to create new ones. Simple applications should be easy to write, requiring little or no detailed understanding of how the system is implemented. More elaborate applications might require a more thorough knowledge of the underlying facilities. The architecture must be designed to minimize the effect of faulty programming on the reliability and performance of the overall system. (Users of experimental software might experience program failure or reduced performance, but other users should not.)
- *Openness.* It should define the role of each major component, so that different kinds of components could be used to provide the same functions. In this way, multiple vendors of telephone and office equipment could cooperate to provide advanced voice functions in conjunction with workstation-based applications. For example, a conventional PABX (business telephone system) could be used in place of the Etherphones to provide voice switching. Ideally,

*Editing on a word or phoneme level causes two difficulties: the system cannot easily identify word or phoneme boundaries for the user, and inserting or deleting words or phonemes is likely to cause results that sound choppy.

such an architecture would evolve into a standard for voice component interconnection.

The development of the Etherphone system has included an ongoing effort to define such an architecture, and to implement the system in compliance with it. Following the general methodology employed by such modern communications architectures as the ISO reference model[7] or the Xerox Network Systems protocols[22], the voice architecture is expressed as a set of layers, each calling on the capabilities of the layer below it through well-defined interfaces or protocols. We have identified five distinct layers. From highest to lowest, these are the *Applications layer*, the *Service layer*, the *Conversation layer*, the *Transmission layer*, and the *Physical layer*.

The best way we have found to explain this organization is from the inside out, beginning with the heart of the architecture, the *Conversation layer*. It provides a uniform approach to the establishment and management of voice connections, or *conversations*, among the various services. It also provides a standard method for distributing conversation state transitions and other progress reports to the various participants in each conversation. All communications between services are mediated by Conversation layer facilities. In the Etherphone system, the functions of the Conversation layer are implemented entirely within the Voice Control Server. However, the architecture does not mandate centralized control. For example, Etherphones built with larger memories and more powerful processors could support a distributed implementation, each managing the conversations that it or its associated workstation initiated.

The *Service layer* defines the various voice-related services—such as telephone functions, voice recording and storage, voice playback, speech synthesis, and speech recognition—that form the basis for the voice applications. Each of the services must follow the uniform Conversation layer protocols in creating voice connections with other services. However, each can register with the Conversation layer additional service-specific interfaces (protocol specifications). Connections may be formed between similar services (as in a call from one telephone to another), or among different services (such as a connection from a telephone to the recording service, mediated by a workstation program). It is not expected that ordinary programmers will produce new services; the services provide both the basic user facilities and interfaces to the building blocks for higher-level applications. In the Etherphone system, some services are implemented on the server machine that contains the Voice Control Server, others on separate server machines, still others on individual workstations.

The *Applications layer* represents client applications that use the voice capabilities of the architecture. To establish voice connections, a client uses simplified facilities provided by a service that resides on the workstation along with the application. Client applications also negotiate with the Conversation layer to gain access to specialized interfaces provided by other services. The previous section illustrated many of the present voice applications using the Etherphone environment.

Logically below the Conversation layer is the *Transmission layer*. This layer represents the actual methods for representing digital voice, for transmitting and switching voice, and for communicating control information among the components of the system. In the Etherphone system, voice is transmitted digitally, in discrete packets, using a standard 64 kilobits/second voice representation and our own voice transmission protocol. Other transmission and switching methods could be substituted without affecting the nature of the programs operating in layers above the Conversation layer. Possibilities include synchronous digital transmission, or even analog transmission. The only requirement is that these components provide interfaces that allow the implementation of Conversation layer protocols. As we have mentioned, the control protocol selected for all control communications in the system was a locally-produced remote procedure call protocol. Other remote procedure protocols or message-based protocols would work equally well.

Finally, the *Physical layer* represents the actual choice of communications media, for the transmission of both voice information and control (not necessarily the same media). Besides the research Ethernet that we use (operating at 1.5 megabits/second), voice transmission on standard Ethernets, synchronous or token-oriented ring networks, digital PABX switches, or analog telephone switches could be used.

Looking at the architectural layers, it becomes easy to see how our efforts differ from work being done elsewhere. Most of the current efforts to "integrate voice and data", such as those systems built around the Integrated Systems Digital Network (ISDN) definitions[4], deal only with the Transmission and Physical layers. Other systems that include voice, such as the Diamond research effort at BBN[21] and commercial voice mail services, support some specialized applications exhibiting very scanty Service and Conversation layers. They mostly build directly on capabilities corresponding to our Transmission layer. By contrast, we have concentrated our efforts on Conversation and Service layer specifications, and on the architecture in general. A project recently initiated at Bell Communications Research has been exploring similar goals and methods[6].

To date, only one instance each of the Physical, Transmission, and Conversation layers has been implemented. We have used the resulting facilities extensively to produce the various Services and Applications described in the preceding sections. We have produced a relatively complete workstation service for Cedar workstations, and a preliminary implementation for Interlisp.

We are not yet fully satisfied with the architecture, particularly the interface between the Conversation and Service layers. This interface has proven to be somewhat clumsy to use, while at the same time restricting the number of capabilities that can be readily produced. Recent progress is encouraging, however.

5. Related Work

Incorporating voice and/or telephony into a workstation environment is an active research area.

Others have explored the use of local area networks to transport packetized voice. For example, the packet XCS system developed by Sincoskie and his colleagues at AT&T Bell Laboratories and later at Bell Communications Research also uses Ethernet[5]. In the Island system, the slotted Cambridge ring supplies guaranteed transmission bandwidth; voice and data are transported on the same network[1].

Work at IBM by Ruiz addressed both voice and telephony[12]. Off-the-shelf voice and telephone hardware was added to an office workstation. The system's prototype applications included directory-based autodialing, call logging, voice mail, telephone message recording and retrieval, and voice annotation of documents (one annotation per line). Annotated documents were stored in three parts: the text, a file of pointers to voice annotations, and a separate file for each annotation. However, there were no voice editing capabilities except for the ability to append new voice, and there was no visual representation of the recorded voice.

The Sydis Information Manager is a commercial system that provides voice and telephony[11]. It includes a shared central processor cluster with voice file storage, a voice processor, and a connection to the local PBX. Special workstations with integrated telephones called VoiceStations support directory-based autodialing, voice and text mail, dictation and voice editing, voice annotation of documents (again, one annotation per line), and voice as a user interface output medium for items such as help messages or event announcements. The Sydis voice editing system uses a sound-and-silence representation similar to ours, but it lacks the ability to annotate the voice with text markers.

Maxemchuk's speech storage system[10], developed at AT&T Bell Laboratories in the late 1970's, provided many of the same facilities for recording, editing, and playing voice as our Voice File Server. Several additional features were included to permit scanning long voice messages: text markers that could be searched by a text editor, the ability to increase playback speed, and the ability to skip forward or backward in the message. Maxemchuk's system edited voice via divide and join operations that modified the control sectors of the stored voice messages. By contrast, our scheme of building data structures that reference segments of voice files allows many users and/or voice annotations to share a single copy of the voice bits.

The Diamond multimedia message system[21], like the Etherphone system, manages documents that contain various media elements by reference. Voice annotations appear in a document as an icon with a text caption; thus their placement is even more restricted than the one-annotation-per-line systems above, but the captions are available for text searches. Because only Diamond documents can reference voice passages, they can use a simpler reference counting scheme than ours.

The MICE project at Bell Communications Research has constructed a centralized voice architecture to support rapid prototyping of communications services[6], including remote call-forwarding, voice paging, voice mail, and combined voice/data messages with editing. The central control process of the MICE system interprets finite state logic tables that specify the voice services. These tables allow changes to algorithms and services without recompiling or reloading the system (which would interrupt service to users).

In contrast to the MICE and Etherphone centralized server architectures, the approach used by Schmandt and his colleagues at the MIT Media Laboratory associates a PC-based voice peripheral with a workstation[15]. The voice peripheral uses commercial speech and telephony cards to support voice filing and editing, telephony functions, and speech recognition and synthesis. The system has been used to provide several sophisticated applications, including combined voice editing and recognition[13], conversational telephone answering[14], and office activity management.

6. Current and Future Directions

Recall that the Etherphone project began with a set of applications-level functional goals for taming the telephone and the tape recorder. After building a basic hardware platform for achieving these goals, early applications-level efforts convinced us of the need for a voice system architecture. More recent efforts have focused on developing the architecture and a few interesting applications to demonstrate and explore its unique characteristics and flexibility. Voice project members have built most of the applications, although a few programmers have included telephone management or voice synthesis activities in their applications using interfaces provided by the Services layer.

We have begun to explore a number of new directions and enhancements to the current capabilities. We have a skeletal implementation of call filtering that provides options based on the subject, urgency, or caller's identity to decrease the intrusiveness of the telephone for the callee. Our plan to integrate telephone conversation logs into the electronic mail system should have a side benefit of making the additional filtering information natural for the caller to supply.

We are working on novel kinds of interactive voice connections, such as all-day "background" telephone calls, use of the telephone system to broadcast internal talks or meetings (as a sort of giant conference telephone call), and conference calls that allow side-conversations to take place. We have recently made it possible to register conversations by name (such as BudgetMeeting or RadioService); users may join these conversations as either listeners or participants.

We plan to use our hardware-supported conferencing capabilities to incorporate text-to-speech or recorded voice into telephone calls. Among possible uses for text-to-speech are reading electronic mail over the telephone to a remote lab member as in PhoneSlave[14] or MICE[6] (but without dedicating a synthesizer solely to this task) and playing program-generated messages to callers, such as prompts or

reports of the user's location (possibly by consulting the user's calendar program, such as "Dr. Smith is at the Distributed Systems seminar now, please call back after 5 o'clock").

We are also exploring a novel *scripting* mechanism for creating viewing paths through one or more electronic documents[23]. Built on the capabilities of the voice architecture, scripted multi-media documents can contain a combination of text, pictures, audio, and action. Scripts can be used in a wide variety of ways, such as for formal demonstrations and audio-visual presentations, for informal interpersonal communications, and for organizing collections of information. Scripted documents are a dynamic form of hypermedia document whose additional structure can be layered on top of existing Tioga documents.

Finally, we would like to extend the system to other media, such as still and real-time video, other workstations, and other architectures.

In concert with work on providing new features, we intend to continue to investigate the underlying systems and theories. Managing real-time and stored voice in a distributed environment presents many interesting problems in the areas of distributed systems[19], user interface design, and voice transmission and processing technologies.

Acknowledgments

Larry Stewart, Severo Ornstein, and Susan Owicki were instrumental in the design and implementation of the Etherphone telephone management facilities. Lia Adams developed an early prototype of voice in electronic mail, Stephen Ades was responsible for the voice annotation system, and Luis Felipe Cabrera participated in the design of voice ropes and an analysis of the Voice Transmission Protocols. Dan Greene designed a prototype call filtering mechanism, John Osterhout implemented the Voice File Server, and Ken Pier implemented ring tunes.

References

1. S. Ades. *An Architecture for Integrated Services on the Local Area Network.* PhD thesis, Cambridge University, February 1987.
2. S. Ades and D. Swinehart. Voice annotation and editing in a workstation environment. *Proc. of AVIOS'86 American Voice Input Output Society Voice Applications Conference*, September 1986, 13-28. Also available as Xerox PARC report CSL-86-3, September 1986.
3. A. Birrell and B. Nelson. Implementing remote procedure call. *ACM Trans. Computer Systems 2*, 1, February 1984, 39-59.
4. M. Decina. Progress towards user access arrangements in Integrated Services Digital Networks. *IEEE Trans. Communications 30*, September 1982, 2117-2130.
5. J. DeTreville and W. D. Sincoskie. A distributed experimental communication system. *IEEE Journal on Selected Areas in Communications SAC-1*, 6, December 1983, 1070-1075.
6. G. Herman, M. Ordun, C. Riley, and L. Woodbury. The Modular Integrated Communications Environment (MICE): a system for prototyping and evaluating communications services. *Proc. of International Switching Symposium '87*, Phoenix, AZ, March 1987, 442-447.
7. International Organization for Standardization. ISO open systems interconnection – Basic reference model. ISO/TC 97/SC, *16*, 719, August 1981.
8. D. Knuth. *The Art of Computer Programming, Volume 3.* Addison-Wesley, 1973 page 392.
9. A. Lippman, W. Bender, G. Salomon, M. Saito. Color word processing. *IEEE Computer Graphics and Applications*, June 1985, 41-46.
10. N. Maxemchuk. An experimental speech storage and editing facility. *Bell System Technical Journal 59*, 8, October 1980, 1383-1395.
11. R. Nicholson. Integrating voice in the office world. *BYTE 8*, 12, December 1983, 177-184.
12. A. Ruiz. Voice and telephony applications for the office workstation. *Proc. 1st International Conference on Computer Workstations*, San Jose, CA, November 1985, 158-163.
13. C. Schmandt. The Intelligent Ear: a graphical interface to digital audio. *Proc. IEEE Conference on Cybernetics and Society*, October 1981, 393-397.
14. C. Schmandt and B. Arons. Phone Slave: A Graphical Telecommunications Interface. *Proc. Society for Information Display 1984 International Symposium*, San Francisco, CA, June 1984.
15. C. Schmandt and M. McKenna. An audio and telephone server for multi-media workstations. *Proc. 2nd IEEE Conference on Computer Workstations*, Santa Clara, CA, March 1988.
16. D. Swinehart. Telephone management in the Etherphone system. *Proc. of GlobeCom'87, IEEE Communications Society Conference*, Tokyo, Japan, November 1987, 392-402.
17. D. Swinehart, L. Stewart, and S. Ornstein. Adding voice to an office computer network. *Proc. of GlobeCom'83, IEEE Communications Society Conference*, November 1983, 392-402. Also available as Xerox PARC report CSL-83-8, February 1984.
18. D. Swinehart, P. Zellweger, R. Beach, and R. Hagmann. A structural view of the Cedar programming environment. *ACM Trans. Programming Languages and Systems 8*, 4, October 1986, 419-490.
19. D. Terry. Distributed System Support for Voice in Cedar. *Proc. of Second European SIGOPS Workshop on Distributed Systems*, August 1986.
20. D. Terry and D. Swinehart. Managing stored voice in the Etherphone System. To appear in *ACM Trans. Computer Systems 6*, 1, February 1988. An extended abstract appears in *Proc. of Eleventh ACM Symposium on Operating System Principles*, Austin TX, November 1987, 103-104.
21. R. Thomas, H. Forsdick, T. Crowley, R. Schaaf, R. Tomlinson, V. Travers, G. Robertson. Diamond: A Multimedia Message System Built Upon a Distributed Architecture. *IEEE Computer*, December 1985, 65-77.
22. Xerox Corporation. *An Internetwork Architecture.* XSIS 028112, Xerox Corporation, Stamford, Conn., December 1981.
23. P. Zellweger. Active paths through multimedia documents. To appear in *Proceedings of EP'88 International Conference on Electronic Publishing, Document Manipulation, and Typography*, Nice, France, April 1988.r

Telephone Management in the Etherphone System

Daniel C. Swinehart

Abstract: Several examples illustrate telephone management features that exist or are planned for the Etherphone system, which is an experimental environment that augments an existing electronic office system with methods for transmitting, storing, and manipulating digital voice. Some Etherphone capabilities are available from any telephone, while more sophisticated functions are obtained through applications running in an associated workstation. Knowledge of the user's identity and preferences form the basis for features not usually available in telephone systems.

Etherphones have been designed to support the ready implementation of additional applications. Most enhanced telephone options and other voice services can be entirely implemented in the workstation environment. Those that prove particularly useful may migrate to a server in order to extend their availability to stand-alone telephones and to a wide range of operating system environments and workstation types.

This paper appeared in the *Proceedings of the IEEE/IEICE Global Telecommunications Conference* (Tokyo, November 1987), 1176-1180.

CR Categories and Subject Descriptors: C.2.4 **[Computer-Communication Networks]**: Distributed Systems – *distributed applications*; D.4.7 **[Operating Systems]**: Organization and Design – *hierarchical design; distributed, real-time, and interactive systems*; H.1.2 **[Models and Principles]**: User/Machine Systems – *human factors*; H.4.3 **[Information Systems Applications]**: Office Automation; H.4.1 **[Information Systems Applications]**: Communications Applications.

General Terms: Design, experimentation, human factors.

Additional Keywords and Phrases: Etherphones, telephones, recorded voice, voice system architecture, workstation telephone management, multimedia conferencing, collaborative work.

XEROX

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

TELEPHONE MANAGEMENT IN THE ETHERPHONE SYSTEM

DANIEL C. SWINEHART
Computer Science Laboratory
Xerox Palo Alto Research Center

Abstract: Several examples illustrate telephone management features that exist or are planned for the Etherphone system, which is an experimental environment that augments an existing electronic office system with methods for transmitting, storing, and manipulating digital voice. Some Etherphone capabilities are available from any telephone, while more sophisticated functions are obtained through applications running in an associated workstation. Knowledge of the user's identity and preferences form the basis for features not usually available in telephone systems.

Etherphones have been designed to support the ready implementation of additional applications. Most enhanced telephone options and other voice services can be entirely implemented in the workstation environment. Those that prove particularly useful may migrate to a server in order to extend their availability to stand-alone telephones and to a wide range of operating system environments and workstation types.

1. Introduction

The Etherphone system is an experimental facility that has been developed at the Xerox Palo Alto Research Center for the purpose of exploring a wide range of telephony, voice mail, voice annotation, and other multi-media document applications. Our approach has been to augment an existing office workstation environment with the capabilities needed to handle the transmission, storage, manipulation, and synthesis of digital voice and music, taking care to preserve conventional telephone behavior to avoid confusing unwary users. The system includes program packages and network services that permit other developers to incorporate sophisticated voice functions into their own applications with relatively little effort. All of the applications described here are implemented using these packages and services.

From the beginning, although the project has required the construction of a complete system, our primary research focus has been not on the particular choices of switching or voice transmission methods, terminal equipment, or network architectures, but on the functionality we wished to achieve. Ironically, most of the earlier publications on the Etherphone system have emphasized the system objectives, the overall hardware and software organization [16, 18], and the specific software methodologies used to handle recorded voice [19], although we have published descriptions of the designs of the user interfaces that support voice annotation and editing [2] and scripted documents [22]. The intent of this report is to demonstrate through examples how the Etherphone system can be used to improve the nature of the telephone in the office setting. Here we concentrate primarily on telephone control and management functions, referring to the annotation and editing capabilities only as they relate to these functions.

Section 2 contains the substance of this paper. It consists of a series of scenarios describing typical office situations and the way the Etherphone system might deal with them. Section 3 outlines the implementation, concentrating on the way in which control responsibilities are shared by the workstation and other system components. Section 4 concludes with a discussion of the state of the Etherphone system and the future challenges we face.

This is an active research area. We should mention some of the more relevant activities here. Many others have investigated the challenges of transmitting voice as datagrams on local area networks [4, 1]. Some notable approaches to high-level methods for programming telephone and voice functions include work by Ruiz [10], Richards *et al* [9], DeTreville [3], and Herman *et al* [5]. Advanced applications of recorded voice and document annotation have been extensively reported [7, 8, 20, 21]. A few products containing advanced telephone management and integrated voice capabilities are beginning to appear [7]. The Speech Research Group at MIT's Media Laboratory has produced remarkable results in their exploration of intelligent voice-directed interfaces to telephones, answering machines, and office systems in general [12, 13]. However, only recently have other broad-based experimental testbeds for office voice applications begun to emerge, most notably the Mice system [5] and the Island project [1].

2. A day in the life of an Etherphone user

In this section, we wish to give the reader an understanding of the present and potential capabilities of the Etherphone system through a series of descriptive examples. This is not an exhaustive enumeration of the Etherphone's features, but is intended to evoke its look and feel, and to illustrate our general approach to the integration of voice facilities into an electronic office system.

We will follow the activities of a peripatetic professional, Karmen Foozle, through a very busy day. Karmen spends a part of her day in the office, the rest visiting colleagues, attending meetings, and so on. She has frequent dealings with people over the telephone, and many of them have schedules as frantic as hers.

In the scenarios that follow, we assume that each user's desk is equipped with a multipurpose workstation, such as a Xerox 6085 or an Apple Macintosh. All of our examples are, in fact, based on an implementation on a Xerox Dorado workstation running the Cedar programming environment [17]. Each workstation is connected to a communication network. Associated with each workstation is a conventional telephone handset and keypad, as well as a hands-free speakerphone, whose speaker is used in place of a ringer. The network functions include PABX capabilities for managing voice connections to other internal phones, to outside lines, and to a variety of services, including text-to-speech synthesis and storing and playing digitally-recorded voice.

2.1 Call forwarding is replaced by *visiting*

As the day begins, Karmen is visiting Lee Strum in his office three doors from her own. Heard above the sounds of the office is a simple melody that Karmen recognizes as her personal ring tune, or *motif*. A few seconds later, Kim

Figure 1. Visiting.

Rendell's motif joins in, in counterpoint to Karmen's tune. So that Karmen need not rush down the hallway to answer Kim's call, Lee turns to his workstation and registers Karmen as a *visitor* (Figure 1) Immediately, Lee's own telephone repeats the ring-duet. Lee's workstation presents a description of the call (Figure 2), and Karmen lifts the receiver with a friendly "Hi, Kim, what's up?" During Karmen's meeting with Lee, two additional calls find her in a similar way, and Lee also answers one that rings with his own motif. An additional call to Karmen after she has returned to her office reminds Lee to terminate the visiting arrangement. Had the visit to Lee's office been a scheduled meeting, Karmen's appointment calendar would have made and cancelled the visiting arrangements automatically.

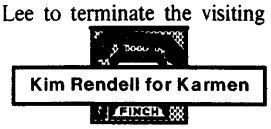


Figure 2. Call for visitor.

2.2 Many flexible call-filtering methods are available

Karmen uses the morning hours for activities requiring intense concentration that the disturbance of frequent telephone calls would disrupt. Yesterday, she modified her personal profile to reject all normal-urgency calls altogether for several hours (Figure 3). For the remainder of the morning, internal callers were given an on-screen explanation for being turned away, while outside callers were routed to an attendant.

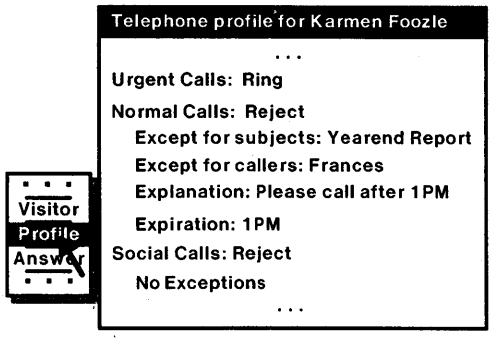


Figure 3. Choose Do-not-disturb option.

Today, Karmen simply wants to reduce the annoyance level of the telephone without ignoring it altogether. She again changes her personal profile (Figure 4), this time to institute *subdued ringing.* As a result, when Frances Brodsky calls, Karmen hears only a single, short, soft tone. On her screen appears the usual visual information (Figure 5), including Frances' name and an indication that it's neither extremely urgent nor merely a social call. If she were to ignore the call completely, it would be handled as if Karmen were not there at all: although it would appear to Frances that Karmen's phone was ringing, Karmen would hear no additional signals. Hearing no answer, Frances would have the standard options of speaking to an attendant, composing and sending a voice message, negotiating a later call with Frances' calendar, or simply trying again later.

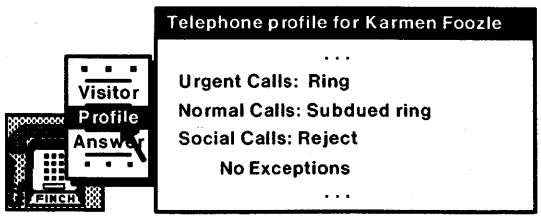


Figure 4. Choose Subdued-ringing option

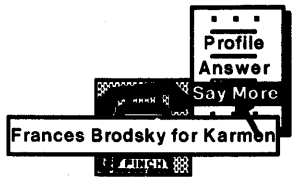


Figure 5. Negotiating a call.

This time, Karmen indicates to Frances that she'll take the interruption if it's important enough. Frances, after a moment's thought, decides that it is and reissues the call at a higher priority, backed up by a short statement of the topic. Karmen answers, automatically switching the call to her speakerphone, and they agree to meet later in the day to resolve a budget crisis (Figure 6).

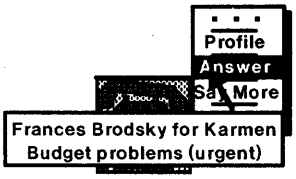


Figure 6. Negotiation successful.

2.3 Calls are to individuals, not locations

After lunch, Karmen takes advantage of Brett Kornwald's temporary absence to spend an hour preparing a presentation at Brett's full-color workstation. Logging in tells the telephone system where Karmen is. For the duration of Karmen's occupancy, although Brett's incoming calls still ring there, so do Karmen's. She can tell them apart because her calls ring with her own motif. Outgoing calls are identified to other colleagues as coming from her; in fact, the workstation and associated telephone behave in all ways as if they belonged to Karmen.

2.4 Background calls and recorded voice support distributed meetings

For their mid-afternoon budget discussion, Karmen and Frances remain in their own offices, connected by a *background* telephone call. Both of them place and receive several calls to other parties during this meeting, sometimes adding them to the background call, sometimes superceding it. The system knows enough about the background connection to reestablish it, in a speakerphone configuration, whenever the standard behavior would otherwise be to hang up the phone. By remaining in their own offices, the participants have full access to their paper documents, workstation documents, policy manuals, etc. Using teleconferencing capabilities such as those described by Sarin [11], Lantz [6], or Stefik *et al* [14], they view a shared budget document, discussing changes as they make them. Karmen and Frances create, edit, and attach voice annotations to specific locations in this document [2], combining these annotations into a narrative intended to convince their manager that several proposed new projects, although expensive, will be well worth it (Figure 7). When the

manager later plays this list of annotations, the corresponding budget document locations will be brought into view on his workstation, in synchrony with the audio presentation [22].

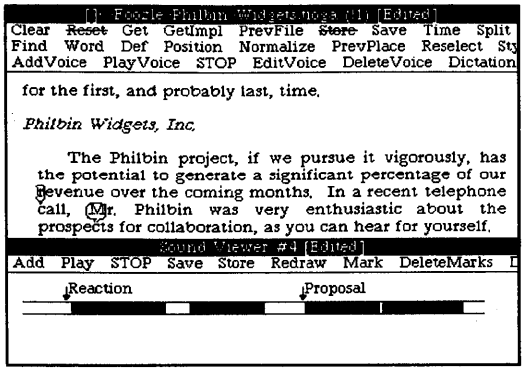


Figure 7. Voice annotation of text documents. The cartoon-like "talks balloons" surrounding selected text characters represent voice annotations. The lower window is used to edit or extend voice annotations.

2.5 Negotiated conference calls save time and annoyance

Following the budget discussion, Karmen wants to advise her own team of the outcome. She turns to her workstation and composes a request for a conference telephone call, to take place ten minutes later. The call will involve Karmen, Lee, Kim, Pat Fisher, and Frances. Karmen selects their names from a hierarchical telephone/personnel directory that merges her personal entries with organizational and community directories, and adds them to the control panel for the conference call (Figure 8). Each participant or his workstation agent is consulted to determine if they will be available. Pat's calendar system accepts on his behalf, since it has been informed that he will be in his office, accepting appointments, and not otherwise engaged. Similarly, Frances already knows about the call, and has indicated a willingness to participate. The others must be consulted. Reacting to a distinctive melody on their telephones, Kim and Lee independently consult their workstations to learn of the proposed call. Each of them indicates whether he or she will be available (Kim can make it; Lee cannot). Meanwhile, Karmen engages in other activities, undisturbed until the scheduled time arrives, when another distinctive tone advises all parties that the conference connection has been established among their telephones. For the duration of the call, a control panel on each workstation screen will inform the participants of the current state of the conference.

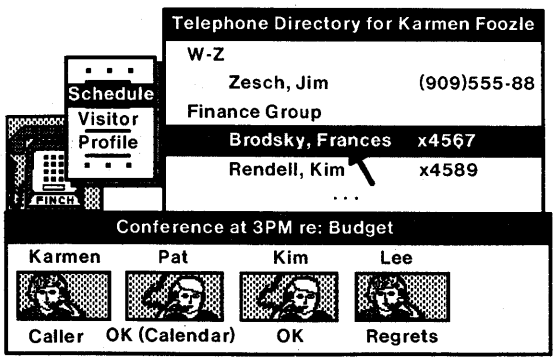


Figure 8. Arranging a conference.

As the call begins, Karmen has the "floor": hers is the only voice that will be transmitted to the other conference participants. After her opening introduction, she grants the floor to each of the other participants in turn for a reaction to the budget proposal, then opens the conference up for the remainder of the call to full interactive discussion [6, 15]. During the conversation, she lets all the participants see and hear the narrated document that she and Frances prepared earlier. A copy of the document itself appears on each workstation, and sections appear, as before, in synchrony with the narration.

2.6 Broadcast facilities permit distributed meetings

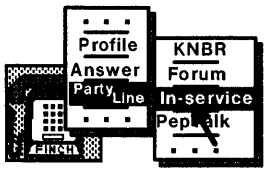


Figure 9. Listening in.

Despite her best intentions, Karmen finds herself with so much paperwork to clean up at the end of the day that she cannot attend the late afternoon in-service training session. Instead, she places a call to the training room (Figure 9), which adds her speakerphone to an ongoing conference call carrying the audio portion of the meeting. During the meeting she mostly listens, occasionally obtaining the floor in order to ask a question or clarify a point. Using similar methods, Karmen has access to television and radio broadcasts, shared recorded audio files, etc.

2.7 The telephone attendant also benefits

After Karmen has left for the day, George Geargrinder, a salesman from another firm, telephones her. Since Karmen has signed off from her workstation, the call rings immediately at an attendant's workstation. As the attendant converses with Mr. Geargrinder, she uses the voice annotation capabilities [2] to record into a form that has appeared on her workstation his answers to questions such as his name, the reason for his call, and when he might be reached (Figure 10). The attendant also enters his name and telephone number textually. Having assured the caller that the message will be delivered, the attendant sends the message, which is already addressed to Karmen. It is waiting for her in her electronic mailbox the next morning, along with her conventional text-only mail. Having listened to him tell her why he wishes to speak with her, she uses the handy "return call" button on the message border to call him back. Voice mail from internal callers may include their photographs to identify them quickly, in place of the evocative sketch that accompanies the message from George Geargrinder.

Readers familiar with the Phone Slave system [12] will realize that the Geargrinder call follows closely the Phone

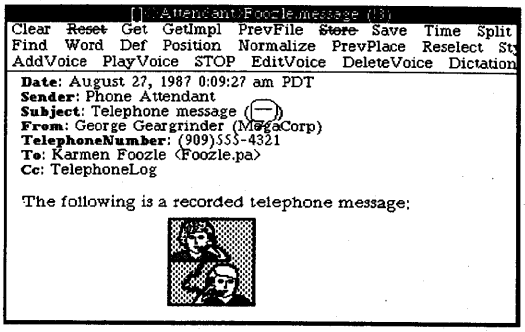


Figure 10. A recorded voice message.

Slave's automated scenario. We do not consider existing automated answering facilities, even those that are as advanced as the Phone Slave, sufficiently advanced to provide adequate service in the office setting, and therefore have proposed enhancing the role of the attendant rather than eliminating it.

3. Implementation

3.1 Hardware environment

A rather unconventional hardware architecture provides the voice services for Karmen and her colleagues (Figure 11). In place of a conventional PABX, voice is transmitted as digital packets, or *datagrams*, on an Ethernet. Digital-to-analog conversion, voice encryption, and datagram transmission are performed by microprocessors, called *Etherphones*, that connect each user's telephone hardware to the Ethernet. A *telephone control server* provides control functions, discussed more fully below. Separate network servers provide voice recording and playback, voice synthesis, and other services.

The telephone control server manages voice switching by sending to each Etherphone or service the network addresses of the other participants. Thereafter, voice datagrams are transmitted directly among the participants, bypassing the control server. Etherphones achieve full-duplex conferencing by digitally summing the multiple voice datagram sequences as they arrive from the other participants and then converting the results to analog form and relaying it to the user. Thus, no special conference hardware is required. All control is also accomplished through datagram-based protocols in the internetwork. This architecture has been described in more detail elsewhere [16, 18].

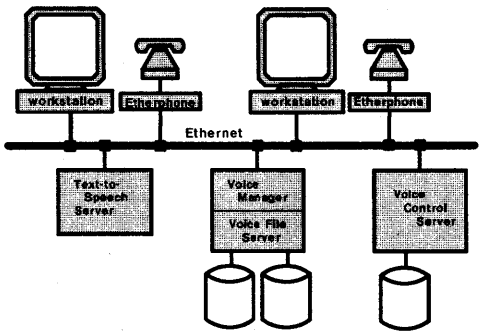


Figure 11. A simple Etherphone System Environment.

3.2 A distributed approach to telephone control

The *telephone control server* controls voice conversations, implements the stand-alone behavior of telephone instruments, and coordinates the activities of workstations and adjacent telephones in their implementation of the various voice capabilities. In addition, it stores personal preference information about each user that allows it to support advanced features such as *ring motifs* and *subdued ringing*, without involving workstation programs. It uses dynamic information linking users to workstations in order to provide calls to individuals rather than fixed locations, and the registration of *visitors* in the offices of their colleagues.

Finally, and most importantly, the telephone control server provides a set of network protocols that workstations use to participate in the operation of the system. Through these protocols, each workstation can monitor the status of the adjacent Etherphone and its voice conversations. Using additional functions, it can place calls to individuals, groups, or recording services. It can edit previously-recorded utterances. It can, if it chooses, selectively override basic call progress decisions. For instance, in response to information describing an incoming call, the workstation can prevent the phone from ringing and instead either reject the call without disturbing the user or even answer it immediately (providing an intercom arrangement). These functions form the basis for the manual and calendar-based call filtering, sophisticated conference call management, and advanced telephone attendant features that were depicted in Section 2. Because we expect many of these features to be experimental, timeout-based safeguards in the server cause a return to conventional telephone behavior if the workstation does not respond in a timely manner to the events reported to it.

During the development of the Etherphone system, we have discovered that as enhanced workstation-based applications mature, it makes good sense to transfer some of their functions to the voice control server or to other network servers. The server protocols are then extended to provide access to these functions. For some features, such as on-line telephone directories, this move has extended their capabilities to telephones in offices without workstations or whose workstations are not active. For other features, such as the package that manages voice editing, a server implementation permits applications written for other operating environments or workstation hardware to use the existing facilities without the necessity of rewriting them from scratch. A function that would benefit from this kind of migration, but which at present is implemented in workstation code, is the ability to filter incoming calls based on a caller's name, a call's subject, or a call's urgency.

5. Conclusions

Most of the capabilities implied by the vignettes of Section 2 either exist today in the Etherphone system, have been shown possible in experimental prototypes, or could be readily achieved by applications programs developed on workstations using existing Etherphone services. Others, such as the telephone attendant features exhibited by the Geargrinder episode in Section 2.7, would require some additional telephone control server development, but no modifications to the overall architecture.

One disappointment has been the difficulty of providing a sufficiently straightforward interface to the applications programmer. At present, the applications programmer must choose between either very simple and limited capabilities or an undesirably complex programming task; workstation applications require the programmer to know far too much about the detailed state of each conversation. Extending these facilities to capture the full range of voice functions in a voice architecture that is easy to use is a challenge for future work.

Overall, however, the distributed systems paradigm of a rich environment of personal workstations and services, connected by a general-purpose internetwork, has extended well to voice applications. Many of the user facilities of the Etherphone system are unmatched in existing voice products. We believe this is due to a system design that combines specialized servers, existing network facilities, and a flexible set of workstation functions that permit the applications programmer to produce sophisticated voice management tools for the office worker.

Acknowledgments

Larry Stewart, Severo Ornstein, and Susan Owicki were instrumental in the design and implementation of the Etherphone telephone management facilities. Polle Zellweger and Doug Terry have kept this project advancing for the past two years, their most recent contributions being invaluable assistance in the preparation of this paper. Subhana Menis applied her considerable copy-editing skills in an attempt to reduce inscrutability by at least a little. I am grateful to Ed McCreight for suggesting the narrative approach (and some of the names) used in Section 2.

Author's address: Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94306.

References

1. S. Ades. *An Architecture for Integrated Services on the Local Area Network.* Ph.D. Thesis, Cambridge University, February 1987.
2. S. Ades and D. C. Swinehart. "Voice annotation and editing in a workstation environment." *Proceedings AVIOS Voice Applications '86*, September 1986, pages 13-28.
3. J. DeTreville. Phoan: "An Intelligent System for Distributed Control Synthesis." unpublished draft.
4. J. DeTreville and W. David Sincoskie. "A Distributed Experimental Communications System." *IEEE Journal on Selected Areas in Communications* SAC-1(6):1070-1075, December 1983.
5. G. Herman, M. Ordun, C. Riley, and L. Woodbury. "The Modular Integrated Communications Environment (MICE): a system for prototyping and evaluating communications services." *Proceedings International Switching Symposium*, Phoenix, AZ, March 1987.
6. K. Lantz. "An Experiment in Integrated Multimedia Conferencing." *Proceedings Conference on Computer-Supported Cooperative Work*, Austin, TX, December, 1986.
7. R. Nicholson. "Integrating voice in the office world." *BYTE* 8(12):177-184, December 1983.
8. J. K. Reynolds, J. B. Postel, A. R. Katz, G. G. Finn, and A. L. DeSchon. "The DARPA experimental multimedia mail system." *Computer* 18(10):82-89, October 1985.
9. J. T. Richards, S. J. Boies, and J. D. Gould. "Rapid Prototyping and System Development: Examination of an Interface Toolkit for Voice and Telephony Applications." *Proceedings CHI'86 Conference on Human Factors in Computing Systems*, Boston, Mass., April 1986, pages 216-220.
10. A. Ruiz. Voice and telephony applications for the office workstation. *Proc. 1st International Conference on Computer Workstations*, San Jose, CA, November 1985, 158-163.
11. S. K. Sarin. *Interactive On-line Conferences.* Ph.D. thesis, Massachusetts Institute of Technology, Report MIT/LCS/TR-330.
12. Schmandt, C. and Arons, B. Phone Slave: A Graphical Telecommunications Interface. *Proc. Society for Information Display 1984 International Symposium*, June 1984.
13. C. Schmandt and B. Arons. "Voice Interaction in an Integrated Office and Telecommunications Environment." *Proceedings 1985 Conference of American Voice Input/Output Society, October 1985.*
14. M. Stefik, G. Foster, D. Bobrow, K. Kahn, S. Lanning, and S. Suchman. "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings." *Comm. ACM* 30(1):32-47, January 1987.
15. D. Stodolsky. "Video Conferencing Process Management." *Proceedings 33rd Annual Conference of the International Communication Association*, Dallas, TX, May 1983.
16. D. C. Swinehart, L. C. Stewart, and S. M. Ornstein. "Adding voice to an office computer network." *Proceedings IEEE GlobeCom '83*, November 1983. Also available as Xerox Palo Alto Research Center, Technical Report CSL-83-8, February 1984.
17. D. C. Swinehart, P. T. Zellweger, R. J. Beach, and R. B. Hagmann. "A structural view of the Cedar programming environment." *ACM Transactions on Programming Languages and Systems* 8(4):419-490, October 1986.
18. D. C. Swinehart, D. B. Terry, and P. T. Zellweger. "An experimental environment for voice system development." *IEEE Office Knowledge Engineering Newsletter*, February 1987.
19. D. B. Terry, D. C. Swinehart. "Managing Voice Stored Voice in the Etherphone System." to appear in *ACM Transactions on Computer Systems, February 1988.*
20. R. H. Thomas, H. C. Forsdick, T. R. Crowley, R. W. Schaaf, R. S. Tomlinsin, V. M. Travers, and G. G. Robertson. "Diamond: A multimedia message system built on a distributed architecture." *Computer* 18(12):65-78, December 1985.
21. H. Wilder and N. Maxemchuk. "Virtual Editing II: the User Interface." *Proceedings of the SIGOA Conference on Office Automation Systems*, Philadelphia, Penn. 1982.
22. Zellweger, P. "Scripted Documents." In preparation.

Managing Stored Voice in the Etherphone System

Douglas B. Terry and Daniel C. Swinehart

Abstract: The *voice manager* in the Etherphone™ system provides facilities for recording, editing, and playing stored voice in a distributed personal computing environment. It provides the basis for applications such as voice mail, annotation of multi-media documents, and voice editing using standard text editing techniques. To facilitate sharing, the voice manager stores voice on a special *voice file server* that is accessible via the local internet. Operations for editing a passage of recorded voice simply build persistent data structures to represent the edited voice. These data structures, implementing an abstraction called *voice ropes*, are stored in a server database and consist of a list of intervals within voice files. Clients refer to voice ropes solely by reference. *Interests*, additional persistent data structures maintained by the server, serve two purposes. First, they provide a sort of directory service for managing the voice ropes that have been created. More importantly, they provide a reliable reference counting mechanism, permitting the garbage collection of voice ropes that are no longer needed. These interests are grouped into classes; for some important classes, obsolete interests can be detected and deleted by a class-specific algorithm that runs periodically.

A version of this paper appeared in *ACM Transactions on Computer Systems* **6**(1), February 1988, 3-27.

CR Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**] Distributed Systems; D.4.2 [**Operating Systems**] Storage Management – *allocation/deallocation strategies, storage hierarchies*; D.4.3 [**Operating Systems**] File Systems Management; D.4.6 [**Operating Systems**] Security and Protection – *access controls, cryptographic controls*; E.2 [**Data**] Data Storage Representations; H.2.8 [**Database Management**] Database Applications; H.4.3 [**Information Systems Applications**] Communications Applications – *electronic mail.*

General Terms: Design, management, performance, security.

Additional Keywords and Phrases: Etherphones, recorded voice, digitized voice, voice editing, voice-annotated documents, storage reclamation, voice file server, network services.

XEROX

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

1. Introduction

Voice is an important and widely used medium for interpersonal communication. Computers facilitate interpersonal communication through electronic mail and shared documents. Yet, our computer systems have traditionally forced us to communicate textually. A major focus of the Etherphone™ system developed at Xerox PARC was to allow voice to be incorporated into computing environments and used in much the same way as text. This paper addresses the problems associated with managing stored voice in a distributed computing environment.

The voice management facilities of the Etherphone system were designed with the following goals in mind:

- *Unrestricted use of voice in client applications*

  As with text, we want the ability to incorporate voice easily into electronic mail messages, voice-annotated documents, user interfaces, and other interactive applications. Nicholson gives a good discussion of many office applications that are made possible by treating voice as data [17].

- *Sharing among various clients*

  If stored voice is to be used as a means of interpersonal communication, then it must be possible for users to easily share a voice passage with one or more colleagues. Clients should be able to share voice as freely as they share files.

- *Editing of voice by programs*

  Clients should be able to combine previously recorded voice in various ways and insert fresh voice into existing voice passages. The system should permit programmer control over all of these functions.

- *Integration of diverse workstations into the system*

  Our environment contains a diversity of workstations and associated operating systems that must access the voice management facilities. Requiring users to learn and adapt to facilities that cannot be well-integrated into their existing workplaces is unacceptable.

- *Security at least as good as that of conventional file servers*

  People are rightfully concerned about the privacy of their communications; the system should take all means to protect this privacy.

- *Automatic reclamation of the storage occupied by unneeded voice*

  Requiring clients to explicitly delete voice passages when they are no longer needed places an unwarranted burden on users and hinders sharing in a distributed system. The system itself should aid in the automatic reclamation of voice storage.

Many of these features are common in traditional file servers that store text files [21]. The characteristics of voice, however, differ greatly from those of text. Standard telephone-quality uncompacted voice occupies 64 Kbits of storage per second of recorded voice. This is several orders of magnitude greater than the equivalent typed text. Voice also requires special devices for recording and playing it; that is, a user cannot simply type in a voice passage. More importantly, voice transmission

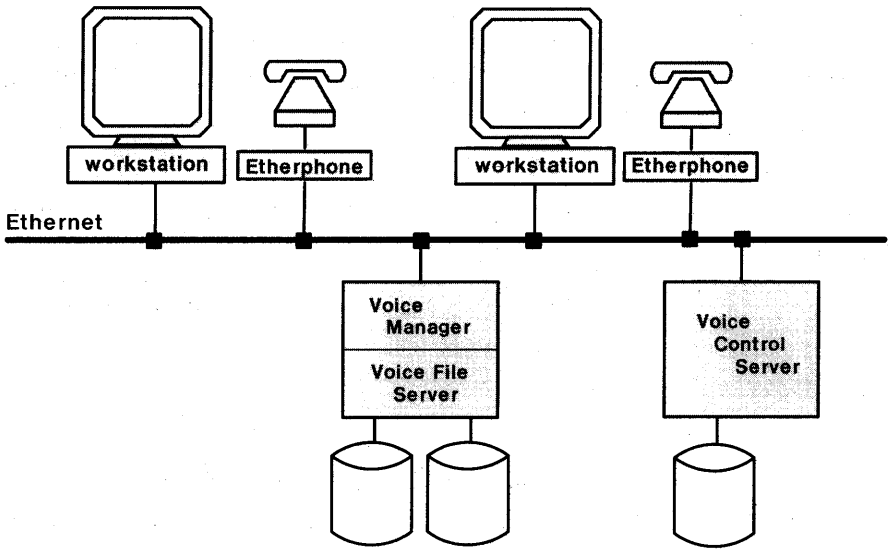


Figure 1. A simple Etherphone system environment.

has stringent real-time requirements. These differences dictate special methods for manipulating and sharing voice.

An abstraction that we call *voice ropes* serves as the basis of application-independent methods for recording, playing, editing, and otherwise manipulating digitized voice. The major technical contributions described in this paper involve the use of simple databases to:

(1) describe the results of editing operations such that existing voice passages need not be moved, copied, or decrypted, and

(2) provide a modified style of reference counting required to allow the automatic reclamation of obsolete voice.

The next section provides the background for the rest of the paper including a quick overview of the Etherphone system architecture as well as some sample voice applications. Section 3 presents the operations on voice ropes designed to support these and future application programs. Section 4 then discusses the design and implementation of the voice rope operations and the rationale governing various design choices. Section 5 relates our experiences thus far with incorporating voice into workstation applications and examines the system's performance. Section 6 contrasts related work. We conclude by reviewing our design goals and how they were met.

2. Background

2.1 The Etherphone system

The Etherphone system is intended for use in a locally distributed computing environment containing multiple workstations and programming environments, multiple networks and communication protocols, and perhaps even multiple telephone transmission and switching choices. The system is intended to be extensible in that introducing new applications, network services, workstations, networks, and other components is possible.

Figure 1 depicts the basic components of the Etherphone system in a simple configuration [24]. Each personal workstation is associated with, but not directly attached to, a microprocessor-based telephone instrument called an *Etherphone.* Etherphones digitize, packetize, and encrypt telephone-quality voice and transmit it directly over an Ethernet. A *voice manager,* the main topic of this paper, provides storage for voice, telephone conversations, music, and other sounds, recorded at reasonable fidelity. The system can also include other specialized sources or sinks of voice, such as a *text-to-speech server* that receives text strings and returns the equivalent spoken text to the user's Etherphone.

A *voice control server* provides control functions similar to a conventional business telephone system and manages the interactions between all the other components. In particular, it allows voice-carrying *conversations* to be established rapidly among two or more Etherphones or voice services. An Etherphone conversation is represented by a conversation identifier, a *ConversationID.* The ConversationID is distributed by the control server to all participants in the conversation, including Etherphones, workstations, and voice services, when the conversation is established. It is used to identify the conversation in requests and reports issued by the server and the participants.

All of the communication required for control in the voice system, such as conversation establishment and the distribution of encryption keys used in voice transmission, is accomplished via a secure remote procedure call (RPC) protocol [4] [5]. Multiple implementations of the RPC mechanisms permit the integration of workstation programs and voice applications programmed in different environments. During the course of a conversation, reports emanating from the voice control server via RPC inform participants about various activities concerning the conversation.

Active parties in a conversation exchange voice using a specialized *voice transmission protocol* [22]. During each conversation, all transmitted voice is encoded using DES electronic-codebook (ECB) encryption [16] with a randomly generated encryption key issued by the voice control server.

Workstations are the key to providing enhanced user interfaces and control over the voice capabilities. The extensibility of the local programming environment—be it Cedar, Interlisp, or the Xerox Development Environment—expedites the integration of voice into workstation-based applications. Workstation program libraries implement the client programmer interface to the voice system. The voice control server associates each workstation with the physically adjacent Etherphone and interprets control requests accordingly. The physical distinction between Etherphones and workstations allows a variety of workstations to be accommodated without requiring additional voice hardware development.

The server software and the initial workstation software was developed in the Cedar programming environment [23]. More information on the equipment and protocols used in the Etherphone system, as well as the applications built to date, can be found in related papers [22] [24].

2.2 Some applications of recorded voice

The desire to annotate documents with voice passages spurred the development of facilities for recorded voice in the Etherphone system. More specifically, users want to attach voice to any point in a document, to play that voice on request, to move or copy annotations readily and quickly, and to store such annotated documents in their host file systems and common file servers. Users want to share annotated documents with their colleagues, who are perhaps using different types of workstations with different file systems and document editors. Users also want the ability to edit voice annotations.

Figure 2 depicts two Cedar *viewers* (windows) involved in a voice editing session. In the top viewer, the Tioga multi-media editor displays a document that includes two voice annotations, normally shown as cartoon balloons. One of the annotations has been "opened" to produce the lower viewer, which uses the alternating dark and light bars to represent intervals of voice and silence proportional to the lengths of the bars. The user employs conventional Tioga editing operations to delete, copy, or reorder sections of the original voice annotation. New spoken material can be recorded into selected locations by invoking special editor extensions. At any time, the user can listen to selected passages. Once edited, the result is then stored back as an amended annotation. This annotation remains with the document and with any copies that are made. The voice editor encourages the replacement and reorganization of recorded voice at the level of entire sentences or phrases. Finer-

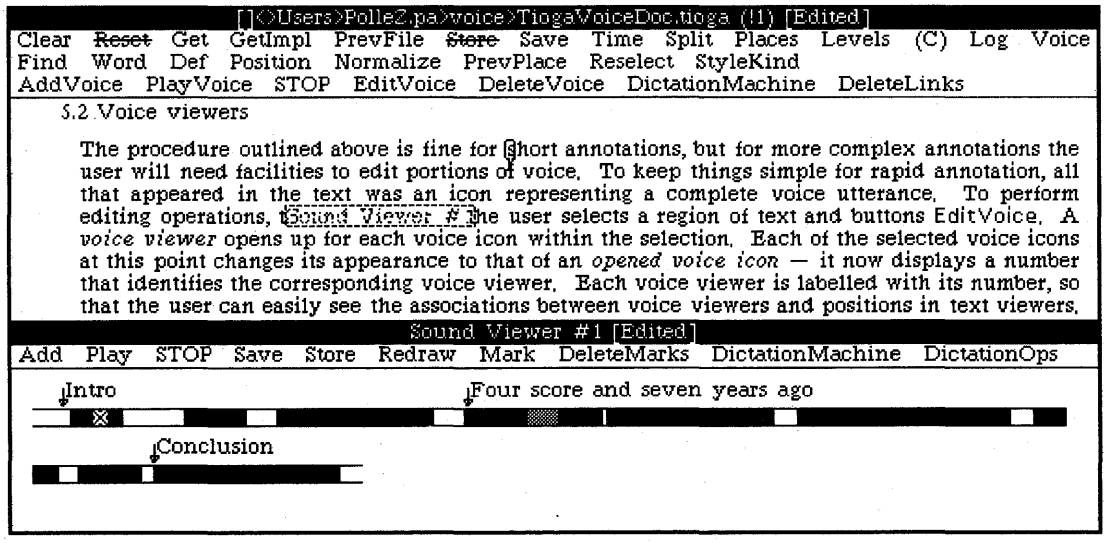


Figure 2. An example of voice annotation and editing.

grained word or phoneme-level operations are discouraged for a variety of practical reasons explored in more depth in related publications [1].

The Etherphone system also includes a voice mail prototype, which is another application of the voice annotation and editing facilities just described. The main difference between mail messages and ordinary annotated documents is that the messages are sent by a general mail transport mechanism to lists of recipients, not manually filed by name in a filing system. Adding voice messaging to an existing environment is complex since mail is often read by a variety of mail programs running on a variety of systems. Most of these programs, both within our own and other interconnected networks, are not yet prepared to play or otherwise deal with recorded voice annotations.

Other applications that need to be able to record, modify, and store voice may employ substantially different user interfaces and storage models than those required of annotated documents. For example, we have found it useful to retain a number of prerecorded prompts, announcements, and attention-getters in a simple directory. Another application that has been proposed would improve the ability to gather data during user studies, by recording the extemporaneous statements of users and synchronizing these recordings with time-stamped logs of other user actions. Additional applications, whose purposes and user interface requirements we cannot know in advance, are expected to be developed in the future.

In general, adding such voice facilities to a diverse and complex software base presents challenging problems to the systems builder since much of the existing workstation and server software cannot be changed or extended. The voice management facilities described in this paper were designed to make such applications less tedious to build, more robust, and easier to comprehend.

3. Operational Overview

3.1 Voice ropes

The implementation of facilities for recorded voice is somewhat involved, but the actions to be performed are conceptually quite simple. In looking for an application-independent abstraction to present to application programmers, it occurred to us that many of these actions closely resembled operations normally associated with text string manipulation. The Cedar system provides a powerful text string abstraction called a *rope* [23]. Cedar ropes are text strings of arbitrary length, represented as memory pointers, or references, to storage that is managed automatically by the Cedar system. When no references to a rope remain active, its storage is reclaimed. Cedar ropes are immutable: once created, its value will not change, so that ropes can be shared simply by sharing references. To create modified values, editing operations are supplied that create new ropes.

By analogy, in the Etherphone system, we refer to sequences of stored voice samples as *voice ropes*. A unique identifier, called a VoiceRopeID, is used instead of a memory pointer to identify each voice rope, because, unlike Cedar ropes, voice ropes are persistent objects with a potentially long lifetime.

To aid in sharing and to facilitate the use of voice by heterogeneous workstations, the storage for voice ropes, as well as the operations on them, are provided by a network service, the voice manager.

Clients refer to voice ropes solely by reference, that is, by their unique VoiceRopeIDs. The voice manager places no restrictions on a client's use of voice ropes. Most uses involve embedding speech in some type of document, such as an annotated manuscript, program documentation, or electronic mail. The use of such embedded references to refer to voice, video, and other diverse types of information has been termed a hypermedia system [26].

From a client's perspective, a voice-annotated document should behave as though the voice were stored directly in the document's file rather than being included by reference. For example, once a voice message is sent using electronic mail, the author or another user should not be able to change the message's contents. For this reason, voice ropes, like Cedar ropes, are immutable. The recording and editing operations create new voice ropes; they do not modify existing ones.

3.2 Recording and playing

To record or play a voice rope, a conversation is set up between the voice manager and an Etherphone. The main operations supported by the voice manager are as follows:

RECORD[conversationID] →

voiceRopeID, requestID

Voice received by the server over the communication path defined by the given conversationID is stored and assigned a unique voiceRopeID; recording continues until a subsequent STOP operation is issued. The requestID identifies this operation in subsequent reports (see below).

PLAY[conversationID, voiceRopeID, interval] →

requestID

The specified interval of the voice rope is transmitted over the given conversation. An interval denotes either the entire voice rope or a time-indexed portion of it at a resolution of about 1 ms.

STOP[conversationID]

Any recording or playing operations that are in progress or queued for the given conversation are immediately halted.

The RECORD and PLAY operations are performed asynchronously. That is, the remote procedure call returns after the operation has been queued by the server. Queued operations are performed in order.

The voice manager generates event reports upon the start and completion of a queued operation. The requestID returned by each invocation is used to associate reports with specific operations. In particular, the voice manager makes the following call to all participants in a conversation to inform them of the status of various requested operations concerning that conversation:

REPORT[requestID, {started | finished | flushed}]

The requested operation has been started, successfully completed, or halted by a STOP operation.

Having reports flow from server to clients is conceptually similar to Clark's upcalls and accomplished in a similar manner [7].

3.3 Editing support

Once recorded, voice ropes can be used in editing operations to produce new, immutable voice ropes. Several of the operations on Cedar ropes, such as producing substrings or concatenating existing strings, are directly applicable to voice. Their transliteration for voice ropes yields these functions:

CONCATENATE[voiceRopeID$_1$, voiceRopeID$_2$, ...] → voiceRopeID

Produces a new voice rope that is the concatenation of the given voice ropes.

SUBSTRING[voiceRopeID$_1$, interval] →

voiceRopeID

Produces a new voice rope consisting of the specified interval of voiceRopeID$_1$.

REPLACE[voiceRopeID$_1$, interval, voiceRopeID$_2$] → voiceRopeID

Produces a new voice rope that is obtained by replacing the particular interval of voiceRopeID$_1$ with voiceRopeID$_2$. This is a composition of the CONCATENATE and SUBSTRING operations provided for efficiency and convenience.

LENGTH[voiceRopeID] → length

Returns the length of the given voice rope in milliseconds.

One additional operation peculiar to voice ropes was provided to aid in editing:

DESCRIBE[voiceRopeID] → intervals

Returns a list of time intervals that denote the nonsilent *talkspurts* of the given voice rope. A talkspurt is defined to be any sequence of voice samples separated by some minimum amount of silence.

These operations, available via RPC calls to the voice manager, are intended for use by programmers. Applications that handle voice must employ these operations to construct the facilities visible to the end user.

3.4 Access control

To ensure privacy, access control lists govern who is permitted to play or edit particular voice ropes. Associated with each voice rope are two types of access: *play* access allows a client to play a voice passage while *edit* access allows the client to use it in editing operations. Access control lists may contain any number of individual or group names that are registered with the local name service, Grapevine [3]. The creator of a voice rope can change these access control lists at any time by calling:

PERMIT[voiceRopeID, players, editors]

Restricts access to the specified voice rope to the lists of players and editors.

Each newly-created voice rope is given the default access controls specified by its creator. Typically, voice ropes are initially given unrestricted access or else restrict access to the voice rope's creator. Clients can later adjust permissions explicitly by calling PERMIT. For instance, a mail sending program could routinely set the play access control list for a voice message to be the set of intended recipients.

3.5 Interests

The voice manager also provides operations for managing voice references. These operations provide a sort of directory for voice ropes. Although simple applications can use this directory as their means for naming and locating voice ropes, that is not its primary purpose. As with any storage system, unreferenced storage space should be reclaimed. With voice, or other voluminous media such as video, the need is particularly acute. Because voice ropes are shared by multiple users and multiple documents, manual management is impractical, and some form of garbage collection is required. In the Etherphone system, client code must assist with garbage collection by using the directory operations to express an *interest* in each referenced voice rope. The client operations are listed here while a discussion of the rationale for this approach and a description of the underlying implementation is deferred to Section 4.5:

RETAIN[voiceRopeID, class, interest]

Registers an interest of the particular class in the given voice rope. The interest uniquely identifies a reference to the voice rope within the class. This operation is idempotent; successive calls with the same arguments register at most one interest in the given voice rope.

FORGET[voiceRopeID, class, interest]

Deregisters the specified interest and deletes the voice rope unless there are other interests for it.

LOOKUP[class, interest] → LIST OF voiceRopeID

Returns the unordered list of voice ropes associated with a particular interest.

Both interest and class are arbitrary text string values. The form of the interest value is generally class-specific. That is, each class controls its own name space of interests and may choose to use hierarchical, flat, or some other form of identifiers for its interest values. Clients are responsible for generating unique values for different interests within a class.

The class of an interest identifies the way in which voice ropes are being used by a particular application. For example, we use the class "FileAnnotation" to indicate that a document stored in a named file is annotated by a set of voice ropes; the interest field is the file name. The class "Message" indicates that an electronic mail message incorporates recorded voice; in this case, the interest is the

unique postmark supplied by the message system.

A combination of client workstation software and automatic collection methods must hide these interest operations from actual users. Client applications must always register interests in order to ensure retention of voice ropes to which they hold references. For some classes, clients must also explicitly FORGET their interests; for others, such as the "FileAnnotation" class, automatic methods described in Section 4.5 make this unnecessary.

4. Detailed Design Decisions

4.1 Voice manager architecture

The voice manager uses a *voice file server* to store voice data. This server provides RECORD, PLAY, and STOP operations that are semantically similar to those described in Section 3.2, but operate on *voice files.* The more complex voice rope editing and directory structures have been implemented as separate, higher-level components, in part to make the voice facilities independent of the choice of the underlying file storage. Voice ropes in the current implementation are actually made up of pieces of one or more voice files, but they could just as well be references to ordinary Cedar files, direct disk addresses, or address values from any other meaningful space.

The implementations of both voice rope editing and interest management depend on a simple but robust *database facility* that was developed for these purposes. Although the operations for editing voice ropes have been patterned after the Cedar Rope package, a different underlying implementation was necessitated by the disparate characteristics of voice and text. Specifically, voice ropes are persistent, not transitory, values. Additionally, editing voice by actually copying the bytes, as is sometimes done for Cedar's ropes, is expensive since voice is voluminous. Thus, rather than rearranging the contents of voice files to edit them, the voice manager simply builds a data structure using the facilities of the database package. Interests are registered in a similar database.

A *garbage collector* uses interests to reclaim voice storage that is no longer needed. Devising techniques for automatically collecting garbage in a distributed, heterogeneous environment was one of the most difficult problems faced in the design of the voice manager.

The components of the voice manager are logically layered as in Figure 3. Each is discussed in more detail in the following sections.

4.2 Voice file server

A voice file server differs from conventional file servers [21] in that it must support the real-time requirements of voice. In particular, it must be able to maintain a sustained transfer rate of 64 Kbits/sec, and it should be able to support several such transfers simultaneously. There is no inherent reason why a general-purpose file server could not be extended to support these stringent real-time requirements. However, the file systems we had available at the onset of this project, having been

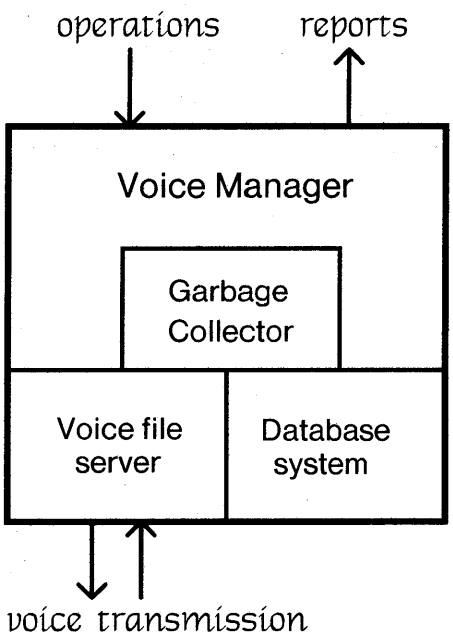


Figure 3. Voice Storage Components

optimized for different styles of access, could not, in fact, support them. At present, our file server is a special-purpose extension of Cedar's standard file system [23].

The Etherphone voice file server implements the operations needed to allocate, record, and play voice files that are named by unique identifiers, *VoiceFileIDs*. As previously mentioned, Etherphones encrypt voice as it is transmitted, using the DES encryption key associated with the current conversation. The voice file server simply stores the voice in its encrypted form. The voice rope implementation assumes responsibility for managing the encryption keys associated with various voice files. The stored voice is never decrypted except by an Etherphone when being played. Tables locating the boundaries between sound and silence are stored along with the voice to permit efficient execution of the DESCRIBE operation.

We will not describe the workings of the voice file server in greater detail. For the purposes of this paper, we assume that the reliable recording of voice files and the reliable playing of arbitrary queued sequences of voice file intervals are solved problems. We also presuppose the existence of relatively high-bandwidth network connections between the voice file server and Etherphones, a condition that is easy to meet in our local network environment.

4.3 Database facilities

Voice ropes and interests rely on a simple database management system for their implementation. The requirements placed on the database system are not particularly stringent. It need only store immutable objects, provide basic query and update mechanisms, and support sharing among many client programs. There is no need for multi-object atomic updates, join operations, or comprehensive query languages. Any database system satisfying these modest requirements would suffice. Since such a system was not available in the Cedar environment, we developed a simple, robust database representation that is particularly well-suited to voice ropes.

The database system stores each entry, a sequence of attributes expressed as key/value pairs, in a write-ahead log [10]. Unlike most database systems in which the data is logged only until it can be committed and written to a more permanent location, the log is itself the permanent source of data. Once logged, the data is never moved. To allow rapid queries or enumeration of database entries, B-Tree indices [2] are built to map the values of one or more keys to the corresponding locations in the log file.

This log-based database package was inspired by similar methods found in the Walnut electronic mail system [8] and recommended by Lampson [12]. In addition to managing the storage of voice ropes and interests, it has been used successfully for other data management needs in the Etherphone system.

4.4 Voice rope structure

The data structure representing a voice rope consists of a list of [VoiceFileID, key, interval] tuples. Additionally, an entry in the voice rope database contains attributes for the identifier, creator, access control lists, and overall length of the voice rope. Thus, a typical database log entry for a voice rope is as follows:

VoiceRopeID: Terry.pa#575996078
Creator: Terry.pa
Length: 80000
PlayAccess: VoiceProject↑.pa
EditAccess: none
VoiceFileID: 235
Key: 17401121062B 10300460457B
Interval: 0 80000

A database index allows voice rope structures to be retrieved efficiently by VoiceRopeID; an index is also maintained on VoiceFileIDs, which is useful in garbage collection.

The database entry given above is for a simple voice rope consisting of a single interval within a single voice file. More complex voice ropes can be constructed using the editing operations presented

in Section 3.3. For example, suppose two simple voice ropes, VR_1 and VR_2, exist with the following structures:

VR_1 = <VoiceFileID: VF_1, Key: K_1, Interval: [start: 0, length: 4000]>

VR_2 = <VoiceFileID: VF_2, Key: K_2, Interval: [start: 500, length: 2000]>

Then the operation

REPLACE[base: VR_1, interval: [start: 1000, length: 1000], with: VR_2]

produces a new voice rope, VR_3, with the structure:

VR_3 = <VoiceFileID: VF_1, Key: K_1, Interval: [start: 0, length: 1000],
VoiceFileID: VF_2, Key: K_2, Interval: [start: 500, length: 2000],
VoiceFileID: VF_1, Key: K_1, Interval: [start: 2000, length: 2000]>

as depicted in Figure 4.

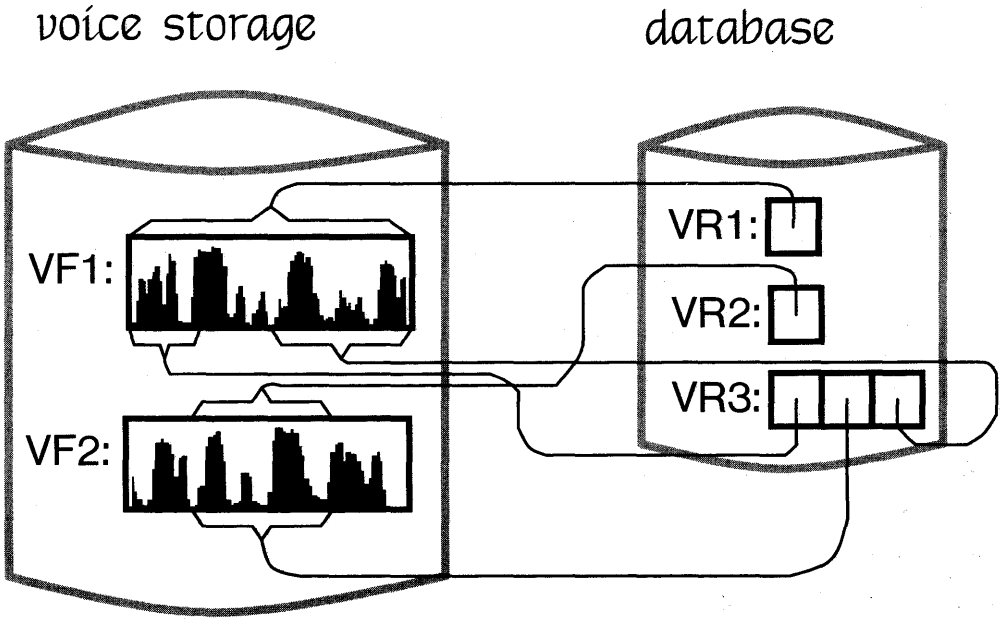


Figure 4. Structure of VR_3 after REPLACE operation.

To record a new voice rope, the voice manager calls on the voice file server to create a new voice file and store the voice arriving over a specified conversation as its contents. Once recording completes, a simple voice rope is added to the voice rope database to represent the complete voice file just recorded. The encryption key used to encode the conversation, and hence the voice file, is stored in the voice rope's entry. This key is carried along with the VoiceFileID during subsequent editing operations involving intervals of the voice rope. Independently enciphering small blocks of voice using ECB ecryption [16] ensures that voice can be edited on millisecond-resolution boundaries while remaining encrypted. Note that the actual voice is neither moved nor copied once recorded in a voice file, even

during editing.

When playing a voice rope, the voice manager retrieves the voice rope's structure from the database, checks access permissions against the caller's authenticated identity, distributes the encryption keys of the various intervals to the parties participating in the conversation, and calls upon the voice file server to play the intervals of the voice rope in the appropriate order. The voice file server provides sufficient buffering to support playing a queue of two or more voice file intervals without introducing gaps between the intervals.

The structure of voice ropes is kept "flat" to enhance playing performance. By having each voice rope refer directly to voice files, only a single database access is required to determine the voice rope's complete structure. An alternative design, more closely modeled on the Cedar Rope abstraction, would store complex voice ropes as intervals of other voice ropes. In such a design, a voice rope would be expressed as a tree of other voice ropes with intervals of voice files at the leaves of the tree. This alternative design would reduce the work associated with each editing operation, but would increase the number of database accesses required to play a voice rope. The flat design was chosen because it improves playing behavior, and, in practice, playing is much more frequent than editing. Moreover, it yields simpler and more compact data structures when used to represent small numbers of coarse-grained edits to voice. The alternative tree design would be a sensible approach within an environment optimized for a different distribution of usage patterns.

4.5 Storage reclamation

4.5.1 Rationale for interests

Once all voice ropes that reference a given voice file have been deleted, no voice rope will ever again refer to that voice file, so the voice file can be deleted as well. This condition is easily determined by a database query. The more difficult problem is deciding when voice ropes themselves can be reclaimed. The interest operations of Section 3.5 were included primarily to permit automatic reclamation of storage for voice ropes and their associated voice files.

From the client standpoint, the most straightforward method for garbage collecting voice ropes would be periodically to examine all of the clients' storage for references to voice ropes, then to collect any unreferenced ropes in a conventional sweep pass. Unfortunately, this sort of distributed garbage collector would be impossible to implement in our open, heterogeneous environment. We do not wish to restrict the uses clients make of voice ropes, or how and where clients store VoiceRopeIDs.

A common alternative is to provide a reference counting scheme in which counters are used to determine the number of clients interested in a particular object. When an object's counter goes to zero, the object's storage can be reclaimed. The burden is placed on clients to increment and decrement the counts for the objects that they are using.

The use of standard reference counting presents formidable problems in a distributed environment. Reference counts cannot be managed reliably unless an atomic transaction can be made

to span all of the activities involved in the creation and deletion of a reference. We consider this impractical to arrange, given our desire to accommodate a heterogeneous collection of participating systems. Without transactions, if the server or client fails in the process of incrementing or decrementing a reference count, the client may be left in an uncertain state regarding the outcome of the operation. In particular, client failure-recovery procedures might incorrectly repeat a reference count operation. Furthermore, it is not always possible to arrange for a reference count to be decremented, such as when a reference-containing file residing on a non-collaborating server is deleted. Lastly, reference counts are anonymous, giving no help in locating erroneous references.

Interests were designed to remedy these shortcomings associated with simple reference counts. In a way, interests represent a return to the full-scan method first proposed: since an examination of the entire environment is not possible, clients are required to record their voice rope interests in a known place. The interests serve as proxies for the actual references for reclamation purposes. Interests address two problems: how to retain voice ropes for which references exist, and how to determine when voice ropes can be reclaimed.

4.5.2 Retention of voice ropes

The use of interests to retain voice ropes is straightforward. Calling RETAIN adds an entry to the system's *interest database*, provided the entry is not already there, recording the supplied VoiceRopeID, class, and interest values along with the user's identification. The interest database includes indices permitting queries based on any of these attributes. Since a given entry appears in the database at most once, the RETAIN operation is idempotent. It can be safely retried in case of failures or uncertainty. The information stored in the interest database is sufficient to allow either human administrators or client applications to determine whether or not an interest is still valid.

4.5.3 Interest invalidation

Determining when to invalidate interest entries is more involved. Some client programs can determine both when to RETAIN an interest and when to FORGET it. One example is the electronic mail system that implements the "Message" class for which the interest design was first developed. When the mail system deletes an instance of a text message that contains voice references, it issues a FORGET for all of the associated voice ropes. Another example is the "SysNoises" class, a set of useful recorded announcements and sounds that system administrators manage manually. The FORGET implementation simply deletes the associated interest entry or entries from the interest database, ignoring requests to delete nonexistent entries to ensure that FORGET is also an idempotent operation. When every interest for a voice rope has been forgotten, the voice rope is vulnerable for deletion.

As we discovered when we began including voice ropes as annotations embedded in ordinary structured text documents, arranging for clients to issue FORGET actions at the necessary times is not always easy. Consider the following scenario. A user records a voice rope and embeds a reference to it in a document; an interest in the voice rope is then registered for the document. The user then copies

this document from his workstation to a public file server and announces its existence in a message to interested parties. Several months later, he deletes the file, without remembering that it had voice annotations. Unless further actions are taken, the interest, and hence the referenced voice rope, will never be reclaimed.

Expecting an arbitrary file server to delete interests is not really reasonable. In the above scenario, one could argue that the file server should have issued the necessary FORGET operation when the file was deleted. However, this implies that the software running on the file server could or should be modified to recognize the existence of files containing voice references, to understand their internal structures, and to take appropriate action. Many of the file servers in a typical network environment, including ours, cannot be so modified, either because they are old and written in some obscure programming language, or because they were purchased from outside vendors from whom source code is not available.

In this case, although interests cannot be explicitly deleted by the agents whose actions invalidate them, the voice manager can determine automatically when a particular interest is no longer valid. Suppose that a knowledgeable workstation client program issues the operation

```
RETAIN[vrID: voiceRopeID, class:
    "FileAnnotation", interest: "annotatedFile"]
```

for each voice rope referred to in a file as the file is moved from temporary workstation storage to the file named "annotatedFile" on a public file server. At any later time, standard directory operations can be used to determine whether that specific named instance of the file still exists; if not, the associated interest is no longer valid and can be deleted.

4.5.4 Garbage collection

For automatically locating and removing outdated interests, the implementor of any interest class can register a procedure of the following type with the voice manager:

GARBAGE[voiceRopeID, interest] → {Yes | No}

> Determines in a class-specific way whether or not the given interest still applies to the particular voice rope.

As an example, for the class "FileAnnotation", this procedure returns Yes if and only if the file instance identified by the interest parameter still exists on some file server. A "Timeout" class records an expiration time as its interest value; its GARBAGE procedure returns YES when the timeout has expired.

An *interest verifier* periodically enumerates the database of interests and calls the class-specific GARBAGE procedure for each interest. If the procedure returns Yes, then the verifier calls FORGET[voiceRopeID, class, interest] to delete the interest from the database.

A garbage collector for voice ropes also runs periodically. For each voice rope in the database, the collector (1) deletes the voice rope if no interests exist that reference it, and (2) deletes voice files used by the voice rope if they are no longer a part of any other voice rope. This process refuses to collect

voice ropes that are too young in order to prevent a newly created voice rope from being collected before a client has the opportunity to express an interest in it. This method will find all unreferenced voice ropes, including those for which no client ever expressed an interest and those that might have been orphaned due to system errors; otherwise, the actions of FORGET alone would be sufficient to eliminate unreferenced entries from the voice rope database. Note that, unlike a mark-and-sweep style garbage collector, these algorithms can be safely executed while the system is running and need not complete a full pass through the database in order to perform useful work.

The decision to protect voice ropes for a time after their creation was a pragmatic one. An earlier design required the client programmer to specify an interest, possibly of class "Timeout", for every voice rope created in order to force an explicit statement of the minimum conditions for deleting it. But since many voice ropes are often created in the process of editing an annotation or voice message, this approach would generate unnecessary interests protecting these intermediate values. Instead, by choosing a minimum lifetime comparable to the interval between invocations of the garbage collection procedures, we leave ample time for a properly functioning application to express an interest in the final result.

In summary, garbage collection takes place on three levels. The voice manager deletes voice files when they are no longer referenced by voice ropes. Voice ropes are deleted if they are old enough and no interests exist for them. Interests are either explicitly forgotten by client applications or automatically deleted based on a class-specific test for validity.

5. Experience and Evaluation

5.1 Current usage

Approximately 50 Etherphones are in daily use in the Computer Science Laboratory. Our current voice file server runs on a Dorado [11], a 2-3 MIPS workstation developed at Xerox PARC, with a 300 Mbyte local disk. Thus, it has the capacity to store over 7 hours of recorded voice; the actual storage capacity depends on the amount of suppressed silence. Most of our user-level applications to date have been created in the Cedar environment [23], although limited functions have been provided for Interlisp and for stand-alone Etherphones. We have had a voice mail system running since 1984 and a prototype voice editor available for demonstrations and experimental use since the spring of 1986.

The current voice rope database contains approximately 2000 voice ropes with a total of 7000 segments referencing about 800 distinct voice files. Thus, the average voice rope consists of 3.5 segments, and an average voice file is used by 2.5 voice ropes. Half of the voice ropes have only a single segment, but some have over 20 segments. Half of the voice files are used by exactly one voice rope.

Voice ropes have an average length of 15.5 seconds. 95% are less than 45 seconds in duration. The average segment size among edited voice ropes is 3 seconds, which is in accordance with our preference

for large-grained edits.

51 MBytes of storage is used by the voice file server to store the collection of voice files whose aggregate length is 54 MBytes. Thus, only about 5% in storage is saved by suppressing silence. Summing up the total lengths of all voice ropes yields 180 MBytes, which indicates that a factor of 3 is gained through using the database to represent edits.

A majority of the voice ropes in the current database were created in order to test the voice manager itself or its applications during their development, and are therefore somewhat artificial. We would expect voice ropes representing annotations in actual production use to be somewhat longer on average, to consist of longer average segments, and to contain fewer edits. Presently, the interest database has not grown large enough to provide interesting statistics.

5.2 Voice-annotated documents

Manipulating stored voice solely by textual references, besides allowing efficient sharing and resource management, has made it easy to integrate voice into documents. For example, we were able to build a local voice mail system without changing the mail transport protocols or servers. Also, annotated documents can be stored on conventional file servers that are not aware that the documents logically contain voice. The annotation applications utilized facilities already available in the Tioga editor for associating additional named properties with arbitrary locations within a document.

Significant performance benefits accrued by having documents refer to voice that is stored remotely. Although most requests to record or play voice ropes are initiated from a workstation, the voice data is never received by the workstation; instead, it is transmitted directly to the associated Etherphone. A typical text document containing several voice ropes as annotations might occupy 30,000 bytes of storage, whereas inclusion of the voice, assuming a minute of total commentary, would swell its size to 500,000 bytes or greater. In fact, scanned images are included in Tioga documents by value, and the resulting sizes do present storage and performance problems even for high-performance workstations.

Sharing documents and electronic messages that have been annotated with voice references, however, requires high-bandwidth network connectivity among the participants. Providing the applications described here among remote sites connected by limited-bandwidth channels would require additional mechanisms, such as those developed as part of DARPA's experimental multimedia mail project [18], which are beyond the scope of this paper.

5.3 Editing voice

We gained considerable experience with the voice manager by building the Cedar voice editing system described in Section 2.2 and illustrated in Figure 2. The small set of editing operations provided by the voice manager, including the specialized DESCRIBE operation that identifies the sound and silence intervals, has proven to be a sufficient base on which to build a complex voice editor

and a dictation machine. However, to reduce traffic to the voice manager, the Cedar voice editor maintains its own data structures to represent the edited voice temporarily. That is, the voice editor ended up replicating much of the functionality of the voice manager, something we were trying to avoid. Only when a user elects to save the edited voice passage does the voice manager get called to perform the necessary operations. Given this situation, a better approach may have been to let clients simply pass the voice manager a complete voice rope specification that it could store in its database.

Experience with the initial implementation of the voice manager revealed that editing a voice passage invariably produced a set of "temporary" voice ropes used only in the construction of the finished result. These objects were eventually collected by the garbage collector and did no permanent harm, but they did create additional work for the voice manager. To alleviate the problem somewhat, the voice manager's interface was changed slightly so that an interval could be supplied for any voice rope parameter in any operation. This substantially reduced the voice editor's use of the SUBSTRING operation.

Event reporting is important in allowing the voice editor to coordinate its visual feedback with the activities of the voice file server. In particular, the voice editor moves a cursor along the screen as a voice rope is being played (the gray marker below the word "score" in the voice displayed in Figure 2). A report indicating that the playing of a particular voice passage has started or finished is essential to synchronize the movement of the cursor with the transmission of voice data.

Although the voice file server writes files on disk so that 1-second segments can be continuously transferred, clients are allowed to edit voice ropes on 1-millisecond boundaries. The file server could not possibly play a voice rope in real time if it had to perform a disk seek every millisecond. Fortunately, users of the voice editor are encouraged to insert, delete, and rearrange voice passages at the granularity of a sentence or phrase rather than trying to modify individual words or phonemes [1]. Thus, in practice, one rarely sees segments of a voice rope that are less than several seconds in length.

5.4 Interests

The notion of grouping interests into classes and providing class-specific garbage collection algorithms is a useful and workable concept. However, we are still groping with the details of how best to use these mechanisms. We have found several interest classes to be useful in Cedar.

The Cedar mail system automatically registers and deregisters interests of class "Message" as voice messages are saved and deleted by recipients. In addition, a "Timeout" class has been used to retract an interest automatically after a certain amount of time. For instance, when sending a voice message, a timeout of a week or two can be set by the sender to give recipients a chance to receive the message and register their own more permanent interests if so desired. Of course, problems can arise if a recipient is on vacation for a period of time longer than the timeout. For this reason, voice files are archived before being deleted from the server.

For annotated documents in Cedar, the workstation software detects when a file is copied from the local disk to a public file server; it then calls RETAIN to register the appropriate "FileAnnotation"

interests for the public file. Having workstation software automatically register interests as a file is copied to a file server works remarkably well. However, some important operations are not covered by this approach: renaming a file on a file server or copying files between two file servers. We see no way to detect such operations except by modifying file server software.

We have defined the "FileAnnotation" interest class such that its interest represents a publicly stored file name including the version number. With this scheme, interests must be reregistered for each new version of the file, that is, whenever a file is written to a public server. Unfortunately, the times that people want to annotate documents are precisely those times when the document is being updated often, so many interests are registered repeatedly. We rely on the garbage collector to get rid of old interests. An alternative would be to register a file without a version number, but that causes minor problems if voice is deleted from the file but the file itself remains in existence.

If we could obtain the cooperation of the file servers that store our documents, we could do a significantly better and less cumbersome job of managing interests. A collaborating file server could examine files for voice rope references and execute the necessary RETAIN and FORGET operations as the files were stored, copied, and deleted. This would reduce or eliminate the burden on application programs. We consider this a promising area for additional research.

5.5 Reliability

The voice file server, voice manager, and voice control server were implemented so that they could run on separate physical processors. That is, they all communicate among themselves and with voice clients using RPC. In practice, we run all three on the same Dorado. There is little to be gained by running them separately, since the voice file server cannot record or play voice files if the control server is down. Similarly, the voice manager cannot record or play voice ropes if the voice file server is down. Moreover, for all practical purposes, voice cannot be edited if the voice file server is down because users invariably need to listen to the voice passages that they are editing.

Thus, availability is not adversely affected by having the voice manager and file server co-located with the control server. If this server crashes or is otherwise unavailable, then no operations can be performed on stored voice. For the most part, this is simply an inconvenience to users in the same way that the unavailability of conventional file servers is an inconvenience. In Cedar, the file servers containing the important system files, fonts, and documentation are replicated to improve their availability. In our prototype, we have not found it necessary to pay the cost to provide a highly-available voice file server.

The one exception to this concerns voice interests. Clients often wish to register or deregister interests in voice ropes independently of playing the referenced voice. For example, an interest of type "FileAnnotation" is registered when a voice-annotated document is copied from a personal workstation to a public file server. A user should not be prevented from performing such a copy simply because the voice manager is unavailable. We have also observed that the interests for voice messages fail to get properly registered or deregistered if a person saves or deletes a voice message while the voice server is

Operation	Time: simple	Time: complex
RECORD †	264-390 (318)	—
PLAY †	213-821 (394)	450-617 (563)
CONCATENATE	219-367 (279)	878-1276 (1105)
SUBSTRING	203-736 (373)	239-608 (345)
REPLACE	398-509 (444)	683-1109 (937)
LENGTH	33, 72 ‡	80
DESCRIBE	163	1432
RETAIN	285-881 (539)	=
FORGET	290-763 (514)	531-1024 (718)
LOOKUP	33	=

† Time from beginning of RPC call to "started" report received by client.
‡ For voice ropes whose length is not known by the database.
= Identical performance for simple and complex voice ropes.

Table 1. Performance of operations on simple and complex voice ropes (in milliseconds).

down. This has led us to contemplate writing a program that enumerates a person's mail database and checks that all voice messages have properly registered interests. The better solution is to make the voice interest database highly available, at least for updates. Rather than fully replicating the database, we have designed a mechanism whereby operations to RETAIN or FORGET a voice interest are logged locally by a user's workstation if the database server is unavailable; the operations in this log will be retried when the workstation detects that the voice manager has returned to operation.

5.6 Performance

The time performance of both the voice file server and the voice rope facilities easily meet the requirements of intended applications. Table 1 gives the measured performance of the various operations that can be performed on voice ropes. Measurements were obtained for both a simple voice rope that contains a single 5-second voice segment and a complex voice rope consisting of ten 5-second segments obtained from ten different voice files with different encryption keys. For these experiments, the client is a Dorado on a 3 MBit Ethernet while the server is a Dorado on a 1.5 MBit Ethernet; the client and server are separated by a single gateway. Times are given in milliseconds; for operations with significant variance in measured times across several experiments, a range of times is given with the average in parentheses.

The time given for a RECORD or PLAY operation is the period from the client initiation of the RPC call until the receipt by the client of a file server report that recording or playing has started. Clearly, the complete time for the operation depends on the length of the voice rope being recorded or played. This time can be easily computed given a voice transmission rate of 64 KBits/second.

We were quite surprised by the large variance in the time observed for playing a voice rope.

Further examination of the PLAY operation indicated where the time is spent, as depicted in Table 2. The variability is due to the time required to reliably distribute one or more encryption keys.

Sub-operation	Time
Communication (RPC) overhead:	18
Database lookup and access control:	18
Distribution of encryption keys:	137-745
Schedule playback with voice file server:	4
Time to receipt of "started" report:	36
TOTAL	213-821

Table 2. Breakdown of time spent in the PLAY operation for simple voice rope.

As expected, the cost of the DESCRIBE operation increases almost linearly with the length of the voice rope. For most of the other operations, the time depends heavily on the cost of database updates rather than the size or complexity of the voice rope involved. However, the complexity of a voice rope has an indirect effect on the cost of writing that voice rope to the database. The database system maintains a B-Tree index that keeps track of all voice ropes containing part of a given voice file. Thus, when adding a voice rope to the database, an update to this index is required for each segment of the voice rope. This explains why operations such as CONCATENATE and REPLACE are significantly more expensive for complex voice ropes.

We do not have current measurements of the server load during RECORD or PLAY. However, early experience with the voice file server indicated that it can handle about eight simultaneous connections [22].

During the development of the voice manager, we have exercised the garbage collection routines and have verified that they properly identify the objects to delete, but we have not actually allowed them to delete anything. Because of this, there are quite a large number of voice ropes, voice files, and interests in existence. Even so, recent measurements indicate that the interest verifier makes a complete pass through the interest database in 13 seconds. The voice rope garbage collector runs in 140 seconds, or approximately 80 ms. per voice rope. Thus, the entire garbage collection suite for a database of this magnitude can be executed in just a few minutes. Running the garbage collectors once per night, we could support a considerably larger population of voice ropes and interests than currently exist.

This paper has been concerned with the editing and management of recorded voice, dealing with operations whose performances must be compatible with human response times: sub-second response at a peak rate of several operations per second is more than adequate. The measurements reported herein confirm that the voice manager meets these requirements. Moreover, since network communication time is not a significant factor in the timings reported, the voice rope facilities would work in a

substantially more extensive and heterogeneous network environment than we have tried to date.

6. Related work

Several companies provide speech message systems that can be accessed from standard telephones; one of the earliest examples of this type of system was IBM's experimental Speech Filing System, which was operational in 1975 [9]. Certainly the Etherphone system's facilities can be accessed from telephones, but that was not the driving application. We were interested in allowing voice to be integrated easily into a user's existing means of digital communications, rather than forcing users to learn a completely new system. The Sydis Information Manager provides workstation control over the recording, editing, and playing of voice as in the Etherphone system, but requires special workstations called VoiceStations [17]. Ruiz also developed a prototype voice system that integrates voice and data into some simple workstation applications; however, he did not address the important issues of sharing stored voice [19].

Maxemchuk's speech storage system [14] provided many of the same facilities for recording, editing, and playing voice as our voice file server. (Actually, he provided much more control over the playing of voice than we do, such as the ability to vary playback speeds or adjust silence intervals.) The division of function between a main computer and a storage computer is also quite similar to the separation between our voice manager and voice file server. However, Maxemchuk's system edits voice using divide and join operations that modify the control sectors of stored voice messages. Our technique of building data structures that reference voice files better supports sharing by making voice ropes immutable and simplifies the requirements placed on the voice file server. For instance, our techniques are very amenable to write-once storage technologies such as optical disks.

Version Storage in the Swallow system [20] has many similar characteristics to our voice manager. That is, it manages immutable objects of various sizes. Also, its "structured version images" used for large objects are similar to the data structures used by the voice file server to describe voice files. Unlike the voice manager, Swallow provides no editing mechanisms or garbage collection, just read and write operations. It does, however, maintain histories to link together objects that are derived from one another and supports atomic operations on multiple objects.

The Diamond Document Store [25], like the Etherphone system, manages documents that contain various media elements by reference; it also allows documents to be shared among users by reference. Because the Diamond system does not allow documents stored outside the system to reference internally stored objects, a simple reference count mechanism suffices for deallocating objects that reside in the Document Store but are not referenced by any document or document folder. The Etherphone system, on the other hand, strives to provide voice services that can be used along with other existing services, such as the Grapevine mail system [3] and Alpine file servers [6].

The Cambridge File Server [15] was perhaps the first network-accessible storage system to require clients to take an explicit action to prevent files from being automatically garbage collected. In

particular, it deletes files that are not accessible from server-maintained, but client-updated "indices". Thus, these indices play much the same role as the voice manager's interest database.

Several methods for distributed garbage collection have been documented in the literature. For example, Liskov and Ladin have a scheme in which all sites that store references to other objects run a garbage collector locally and send information about non-local references to a *reference server* [13]. In some sense, their use of a reference server is similar to our use of registered interests, but much more limited. One interesting contribution they make is how to build a highly-available reference server; we could use these techniques to build an interest server.

7. Conclusions

The facilities for managing stored voice in the Etherphone system were designed with the intent of moving voice data as little as possible. Once recorded in the voice file server, voice is never copied until a workstation sends a play request; at this point the voice is transmitted directly to an Etherphone. In particular, although workstations initiate most of the operations in the Etherphone system, there is little reason for them to receive the actual voice data since they have no way of playing it.

Maintaining voice on a publicly accessible server, the voice manager, facilitates sharing among various clients. Clients can freely share references to voice ropes without incurring the overhead of transmitting the voice itself. Because voice ropes are immutable, even though they are incorporated into documents by reference, they exhibit copy semantics.

To support efficient editing, a two level storage hierarchy is employed: voice ropes refer to intervals of voice files. A given voice rope can consist of intervals from several voice files, and a given voice file can be used by several voice ropes. A database stores the many-to-many relationships that exist between voice ropes and files. Editing operations simply create new voice ropes from old ones and add them to the database.

The editing operations provided by the voice manager are similar to those in the Cedar Rope package. This is intentional so that programmers can manipulate voice in the ways to which they are accustomed to dealing with text. The basic facilities to support editing reside on a server; workstations are responsible for providing a user interface that is integrated with their programming environment.

Several aspects of the voice manager were designed to accommodate the heterogeneous nature of our environment. Providing a single implementation of the voice rope facilities on a server obviates the need for each different workstation programming environment to provide its own implementation. Moreover, the only requirements placed on a workstation in order to make use of the voice services are that it have an associated Etherphone and an RPC implementation. In particular, workstations need not have direct hardware support for encryption or voice I/O.

The voice manager reduces the work generally associated with building voice applications by providing a convenient set of application-independent abstractions for stored voice. It makes no assumptions about the way clients make use of its services. This particularly impacted the design of the

voice garbage collector.

Automatic reclamation of the storage occupied by unneeded voice ropes is done using a modified type of reference counting. Clients register interests in particular voice ropes. These interests are grouped into classes and can be invalidated according to a class-specific algorithm. For the most part, users of voice applications are not aware of how or when interests are registered since the application software handles this transparently.

The Etherphone system uses secure RPC for all control functions and DES encryption for transmitted voice. These ensure the privacy of voice communication, which is important even in a research environment, although the Ethernet is inherently vulnerable to interception of information. Storing the voice in its encrypted form protects the voice on the server and also means that the voice need not be reencrypted when played. All in all, the voice system actually provides better security than most conventional file servers.

The Etherphone system has provided an environment in which to explore the management of voluminous, shared data among distributed and heterogeneous workstation clients. One could argue that compression of digitally recorded utterances would eliminate the need for special treatment in our current voice applications. However, even with compression, there is a performance penalty in manipulating such large objects. More importantly, high-resolution scanned images and real-time digital video recordings will continue to be voluminous. We believe that the techniques presented in this paper are applicable to and beneficial for the management of these media as well as to voice.

Acknowledgments

The design of voice ropes evolved for several years and many people contributed valuable suggestions, including Polle Zellweger, Stephen Ades, Luis Felipe Cabrera, and Larry Stewart. John Ousterhout designed and implemented the voice file server. Lia Adams built an early version of the voice mail system. Michael Schroeder suggested the use of interests for garbage collection of voice messages. Stephen Ades' implementation of a voice editor allowed us to get some experience with voice ropes. Severo Ornstein, Larry Stewart, and Dan Swinehart started the Etherphone project in 1982 and designed the Etherphone equipment. We are grateful to the many Cedar implementors who provided a wonderful environment for the Etherphone system.

Margaret Butler, Alan Demers, Bob Hagmann, Jack Kent, Guy Steele, Polle Zellweger, and many anonymous reviewers, contributed insightful suggestions, comments, and corrections. Subhana Menis, Bridget Scamporrino, Polle Zellweger, and Rick Beach provided invaluable assistance with the composition and printing.

References

1. Ades, S., and Swinehart, D. C. Voice annotation and editing in a workstation environment. In *Proceedings AVIOS Voice Applications '86*, September 1986, 13-28.

2. Bayer, R., and McCreight, E. Organization and maintenance of large ordered indexes. *Acta Informatica* 1, 3 (1972), 173-189.

3. Birrell, A., Levin, R., Needham, R. M., and Schroeder, M. D. Grapevine: An exercise in distributed computing. *Communications of the ACM* 25, 4 (April 1982), 260-274.

4. Birrell, A. D., and Nelson, B. J. Implementing remote procedure calls. *ACM Transactions on Computer Systems* 2, 1 (February 1984), 39-59.

5. Birrell, A. D. Secure communication using remote procedure calls. *ACM Transactions on Computer Systems* 3, 1 (February 1985), 1-14.

6. Brown, M. R., Kolling, K., and Taft, E. A. The Alpine file system. *ACM Transactions on Computer Systems* 3, 4 (November 1985), 261-293.

7. Clark, D. D. The structuring of systems using upcalls. *Proceedings Tenth Symposium on Operating Systems Principles*, Orcas Island, Washington, December 1985, 171-180.

8. Donahue, J., and Orr, W.-S. Walnut: Storing electronic mail in a database. Xerox Palo Alto Research Center, Technical Report CSL-85-9, November 1985.

9. Gould, J. D., and Boies, S. J. Speech filing – An office system for principles. *IBM Systems Journal* 23, 1 (January 1984), 65-81.

10. Gray, J. N. Notes on database operating systems. In Bayer *et al.*, *Operating Systems: An Advanced Course*, Springer-Verlag, 1978, 393-481.

11. Lampson, B. W. and Pier, K. A. A processor for a high-performance personal computer. *Proceedings 7th Symposium on Computer Architecture*, La Baule, May 1980, 146-160.

12. Lampson, B. W. Hints for computer system design. *Proceedings Ninth Symposium on Operating Systems Principles*, Bretton Woods, New Hampshire, October 1983, 33-48.

13. Liskov, B., and Ladin, R. Highly-available distributed services and fault-tolerant distributed garbage collection. *Proceedings of Symposium on Principles of Distributed Computing*, Calgary, Alberta, Canada, August 1986, 29-39.

14. Maxemchuk, N. An experimental speech storage and editing facility. *Bell System Technical Journal* 59, 8 (October 1980), 1383-1395.

15. Mitchell, J. G., and Dion, J. A comparison of two network-based file servers. *Communications of the ACM* 25, 4 (April 1982), 233-245.

16. National Bureau of Standards. *Data Encryption Standard.* Federal Information Processing Standard (FIPS) Publication 46, U. S. Department of Commerce, January 1977.

17. Nicholson, R. Integrating voice in the office world. *BYTE* 8, 12 (December 1983), 177-184.

18. Reynolds, J. K., Postel, J. B., Katz, A. R., Finn, G. G., and DeSchon, A. L. The DARPA experimental multimedia mail system. *Computer* 18, 10 (October 1985), 82-89.

19. Ruiz, A. Voice and telephony applications for the office workstation. *Proceedings 1st International Conference on Computer Workstations*, San Jose, CA, November 1985, 158-163.

20. Svobodova, L. A reliable object-oriented data repository for a distributed computer system. *Proceedings Eighth Symposium on Operating Systems Principles*, Pacific Grove, California, December 1981, 47-58.

21. Svobodova, L. File servers for network-based distributed systems. *ACM Computing Surveys* 16, 4 (December 1984), 353-398.

22. Swinehart, D. C., Stewart, L. C., and Ornstein, S. M. Adding voice to an office computer network. *Proceedings IEEE GlobeCom '83*, November 1983. Also available as Xerox Palo Alto Research Center Technical Report CSL-83-8, February 1984.

23. Swinehart, D. C., Zellweger, P. T., Beach, R. J., and Hagmann, R. B. A structural view of the Cedar programming environment. *ACM Transactions on Programming Languages and Systems* 8, 4 (October 1986), 419-490.

24. Swinehart, D. C., Terry, D. B., and Zellweger, P. T. An experimental environment for voice system development. *IEEE Office Knowledge Engineering Newsletter* 1, 1 (February 1987), 39-48.

25. Thomas, R. H., Forsdick, H. C., Crowley, T. R., Schaaf, R. W., Tomlinsin, R. S., Travers, V. M., and Robertson, G. G. Diamond: A multimedia message system built on a distributed architecture. *Computer* 18, 12 (December 1985), 65-78.

26. Yankelovich, N., Meyrowitz, N., and van Dam, A. Reading and writing the electronic book. *Computer* 18, 10 (October 1985), 15-30.

System Support Requirements for Multi-media Workstations

Daniel C. Swinehart

Abstract: Live and recorded voice, speech synthesis, speech recognition, and full-motion video can be of considerable value when integrated into a powerful text and graphics workstation environment. This paper complements a brief videotape that demonstrates a range of experimental voice and telephony applications – developed as components of the Xerox Etherphone project – with a discussion of the architectural and system support requirements that are necessary to form the basis for flexible, extensible multi-media applications.

This paper appeared in the *Proceedings of the SpeechTech '88 Conference* (New York; April 1988); Media Dimensions, Inc., New York, April 1988, 82-83.

CR Categories and Subject Descriptors: C.2.4 **[Computer-Communication Networks]**: Distributed Systems – *distributed applications*; D.4.7 **[Operating Systems]**: Organization and Design – *hierarchical design; distributed, real-time, and interactive systems*; H.4.3 **[Information Systems Applications]**: Communications Applications.

General Terms: Design, experimentation.

Additional Keywords and Phrases: Etherphones, telephones, recorded voice, voice system architecture, workstation telephone management, multimedia conferencing, collaborative work.

XEROX

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

Introduction

Experimental and commercial workstation-based systems are beginning to emerge that support voice and video applications along with their traditional text, graphics, and computational abilities. Examples include sophisticated telephone management functions, telephone conference control, voice and video annotation of so-called hypermedia documents, voice mail, and audio editing facilities tailored both for ordinary applications and for studio-quality production. For these capabilities to achieve their potential, the underlying system support for them must be carefully planned and implemented.

The evolution of computing systems produced for specific application areas has traditionally followed a natural maturation process, from ground-up implementations to applications built on supportive architectures. As an example, when the first workstation applications were built for office workstation systems (such as the Xerox 8010 Star system or the Apple Lisa), the development approach was to begin with the bare machine or perhaps a simple operating system, then to build each application as a complete, monolithic whole. At the time, this was necessary both because memory sizes were limited and because the system substrate supporting these applications had to be invented along with the applications themselves. However, this approach is undesirable because it leads to long development cycles and to systems whose applications are hard to extend and often not well integrated with each other.

More recently, systems such as the Macintosh, Sun workstations, or the Xerox XDE and Cedar systems have been developed based on a layered set of system facilities that support rapid program development, flexibility and extensibility, and integration. Examples include generalized graphics packages, window management systems, and user interface tool kits. Activity in the field of User Interface Management Systems for traditional workstations is vigorous, and it is leading to a rapid maturation of interactive workstation tools. The later phases of this maturation process include industry-wide standardization on sets of facilities (example: X Windows) that permit systems written by multiple service providers to run on a variety of computing platforms, enabling widespread development and use of new applications.

I contend that most workstation-based voice systems, whether they provide telephone management capabilities or sophisticated applications of digital recorded voice, speech synthesis, or speech recognition, are still in the initial "build it directly on top of the operating system" phase, while systems that deal in any interesting ways at all with full-motion video are still in their infancy. This paper presents an argument that the same process of architectural discovery, development, and eventually standardization is needed in order to support the wide range of voice and video applications that we can all envision, given the recent and continuing explosive development in the enabling technologies.

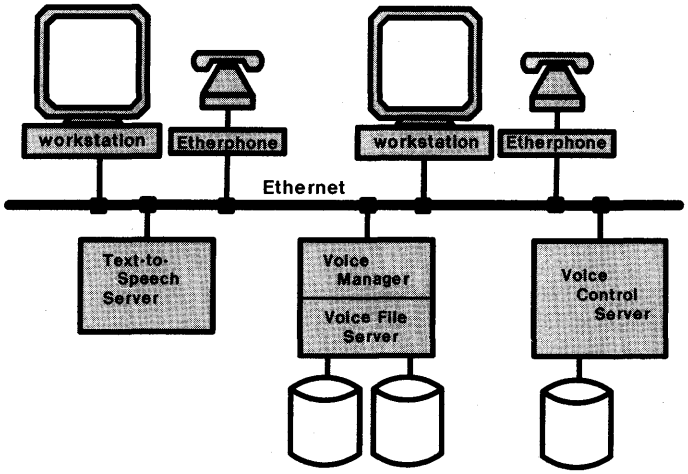
Where things stand

The state of architectural development in this area, as I see it, is that some low-level protocols and architectures for managing voice exist—analogous to the operating system level in the examples above—the most advanced of them represented by the emerging ISDN standard. We can also see in a number of research systems the beginnings of some parochial architectures advanced of them represented by the emerging ISDN standard. We can also see in a number of research systems the beginnings of some parochial architectures that provide higher levels of support. Notable examples are the BellCore *Mice* system [3], special-purpose designs such as the one developed by IBM for the 1984 Olympics in Los Angeles [4], a more general IBM design supporting voice and telephone applications [5], an architectural philosophy advocated by Ades in his PhD dissertation [1], and the architecture of our own Etherphone project [9, 7, 2, 8]. Although each of these systems has its own emphasis, all share the need for a layered architecture providing a range of integrated and extensible capabilities.

As a more detailed example, the architecture with which I am most familiar underlies the Xerox Etherphone system. This is an experimental facility developed to explore a wide range of telephony, voice mail, voice annotation, and other multi-media applications. We have augmented an existing workstation environment with the capabilities needed to handle the transmission, storage, manipulation, and synthesis of digital voice and music, while preserving ordinary telephone functions. These capabilities are presented to applications programmers as program packages and network services. A number of applications have been implemented to test the hypothesis that this environment can support their implementation and incremental extension. Applications to date include ordinary telephone behavior; a workstation-based program providing telephone status display, call control, call filtering, and dialing from a directory database; voice annotation and editing in the context of a multi-media text editor; a "meeting service" permitting the use of the telephone system to monitor remote meetings and to participate orally in them; and a scripted documents facility supporting narrated descriptions of visual documents.

The basic components of the Etherphone system appear in the figure below. Each personal workstation is associated with, but not directly attached to, a microprocessor-based telephone instrument called an Etherphone. Etherphones digitize and encrypt telephone-quality audio and transmit it in packet form directly over an Ethernet. A voice manager provides storage for voice, telephone conversations, music, and other sounds, recorded at reasonable fidelity. The system also includes other

specialized sources or sinks of voice, such as a server connecting a voice synthesizer to the network.



A simple Etherphone System Environment.

In order to achieve many of these applications, the workstation needs to have preemptive control over many of the details of telephone call setup and supervision, while permitting ordinary stand-alone telephone behavior both as a default and as backup when the workstation facilities are unavailable. For many of these features, the workstation needs information about switch ports and users other than the ones associated with the local telephone instrument. To perform these actions with any kind of safety or reliability requires supporting facilities at a higher and more comprehensive level architectures such as present-day ISDN provide behavior both as a default and as backup when the workstation facilities are unavailable. For many of these features, the workstation needs information about switch ports and users other than the ones associated with the local telephone instrument. To perform these actions with any kind of safety or reliability requires supporting facilities at a higher and more comprehensive level architectures such as present-day ISDN provide.

The most compelling published argument I have seen for a standard, rich architecture supporting multi-media applications appears in a recent IEEE Communications article by Strathmeyer [6]. In it, he distinguishes between the *physical integration* enabled and promoted by present-day ISDN protocols, and the *functional integration* needed to support application-level activities. Strathmeyer proposes a voice architecture called *Computer-Integrated Telephony (CIT).* CIT combines ISDN facilities with additional ones providing extended operations. Although it doesn't deal with all of facilities needed for comprehensive voice architecture, the CIT proposal is a strong step in that direction.

Towards a comprehensive voice architecture

In a sense, any system that provides a particular set of capabilities exhibits an architecture for those capabilities. Clearly we are after more than that: a clear, concise, and sensible description of the actions and data structures available to the programmer for producing applications that are more reliable, easier to build, and more likely to integrate well with other applications than could be achieved by starting from scratch. In order to accommodate a wide range of implementations, using possibly differing hardware, the architecture should be expressed as a set of layers, each calling on the capabilities of the layers below it through well-defined interfaces or protocols. This is the same philosophy espoused by the Open Systems Interconnect (OSI) model, and indeed most system and network architectures.

We do not claim to have created such an architecture as yet. What is outlined here has emerged from attempts to extract a clean architectural description of the facilities that comprise the Etherphone system. However, although it was implemented from a careful design, the Etherphone system was not originally designed with all of the required attributes for a general architecture in mind. In particular, we believe that an architecture should be:

- *complete*—expressing all of the underlying facilities needed by its applications,
- *programmable*—capable of extension by workstation programmers, without changing any of the network services, in order to modify existing applications and create new ones, and
- *open*—defining the role of each major component, so that different kinds of components could be used to provide the same functions—for instance, so that a PABX manufacturer could provide a system replacing the Etherphone hardware for voice transmission and switching, without massive reprogramming or loss of functionality.

The Etherphone architecture, as it stands, is reasonably complete and reasonably programmable, but does not meet the openness criteria. Therefore, the following paragraphs represents goals that we would like to meet in completing the Etherphone architecture. This discussion concentrates on voice, but most of the arguments would apply as well to the management of full-motion video applications.

Following the general methodology of the ISO reference model, we have tentatively identified five distinct architectural layers. From highest to lowest, these are the *Applications layer*, the *Service layer*, the *Conversation layer*, the *Transmission layer*, and the *Physical layer*. The heart of this design is the *Conversation layer*, which provides a uniform approach to establishing and managing voice connections among telephones and between telephones and the various services, while also coordinating the activities of telephones with those of their associated workstations. The *Service layer* defines the various voice-related services—

such as telephone functions, voice recording and storage, voice playback, speech synthesis, and speech recognition – that form the basis for the voice applications playback, speech synthesis, and speech recognition – that form the basis for the voice applications. Each of the services must follow the uniform Conversation layer protocols in creating voice connections with other services. However, each can register with the Conversation layer additional service-specific interfaces (protocol specifications) that individual applications use to invoke their specialized capabilities. The *Applications layer* represents client applications that use the voice capabilities of the architecture; the existence of this layer must appear in the architecture, but of course the architecture need not specify in detail the capabilities at this level.

Logically below the Conversation layer is the *Transmission layer.* This layer represents the actual methods for representing digital voice, for transmitting and switching voice, and for communicating control information among the components of the system. Finally, the *Physical layer* represents the actual choice of communications media, for the transmission of both voice information and control (not necessarily the same media). Although these lower layers are represented in the Etherphone system by facilities for transmitting packet voice and control messages on an Ethernet, other local area networks or almost any compliant digital PABX could provide at least most of the same services.

Existing standards, notably ISDN specifications, deal reasonably well with the functions of the Physical and Transmission layers. They also contain facilities corresponding to some of the required capabilities of a comprehensive Conversation layer, although the protocols that have been defined to date are far from complete. It is clear that most of the capabilities of the service layer have not yet been addressed by ISDN. We believe that only after all the architectural levels outlined here have been developed and have been standardized enough for many developers to be using them, will truly successful workstation-based voice and video products become a reality.

References

1. S. Ades. *An Architecture for Integrated Services on the Local Area Network.* Ph.D. Thesis, Cambridge University, February 1987.
2. S. Ades and D. C. Swinehart. "Voice annotation and editing in a workstation environment," *Proceedings AVIOS Voice Applications '86*, Alexandria VA, September 1986, pages 13-28.
3. G. Herman, M. Ordun, C. Riley, and L. Woodbury. "The Modular Integrated Communications Environment (MICE): a system for prototyping and evaluating communications services." *Proceedings International Switching Symposium*, Phoenix, AZ, March 1987.
4. J. T. Richards, S. J. Boies, and J. D. Gould. "Rapid Prototyping and System Development: Examination of an Interface Toolkit for Voice and Telephony Applications." *Proceedings CHI'86 Conference on Human Factors in Computing Systems*, Boston, Mass., April 1986, pages 216-220.
5. A. Ruiz. Voice and telephony applications for the office workstation. *Proc. 1st International Conference on Computer Workstations*, San Jose, CA, November 1985, 158-163.
6. C. Strathmeyer. "Voice/Data Integration: An Applications Perspective." *IEEE Communications Magazine* 25(12): 30-35, December, 1987.
7. D. C. Swinehart, L. C. Stewart, and S. M. Ornstein. "Adding voice to an office computer network." *Proceedings IEEE GlobeCom '83*, November 1983. Also available as Xerox Palo Alto Research Center, Technical Report CSL-83-8, February 1984.
8. D. Terry and D. Swinehart. "Managing stored voice in the Etherphone system." To appear in *ACM Trans. Computer Systems 6*, 1, February 1988. An extended abstract appears in *Proc. 11th ACM Symposium on Operating System Principles*, Austin TX, November 1987, 103-104.
9. P. T. Zellweger, D. B. Terry, and D. C. Swinehart. "An overview of the Etherphone system and its applications." *Proceedings 2nd IEEE Conference on Computer Workstations*, Santa Clara, CA, March 1988.

Active Paths Through Multimedia Documents

Polle T. Zellweger

Abstract: We have developed a scripting mechanism for creating active paths through a document or set of documents. Scripted multimedia documents can contain a combination of text, graphics, audio, and actions. Scripts can be used in a wide variety of ways, such as for formal demonstrations and audio-visual presentations, for informal interpersonal communications, and for organizing collections of information. Scripted documents are a dynamic form of hypermedia document whose additional structure can be layered on top of existing documents.

This paper appeared in *Document Manipulation and Typography*, J.C. van Vliet (ed.), Cambridge University Press, 1988. Proceedings of the EP'88 Conference on Electronic Publishing, Document Manipulation and Typography, held in Nice, France, April 1988.

CR Categories and Subject Descriptors: D.2.6 **[Software Engineering]**: Programming Environments; H.1.2 **[Models and Principles]**: User/Machine Systems – *human factors*; H.2.4 **[Database Management]**: Systems – *distributed systems*; H.2.8 **[Database Management]**: Database Applications; H.4.m **[Information Systems Applications]**: Miscellaneous – *hypertext*; I.7.2 **[Text Processing]**: Document Preparation.

General Terms: Design, experimentation, human factors.

Additional Keywords and Phrases: Etherphones, Cedar, recorded voice, voice-annotated documents, multimedia documents, active documents, hypermedia, hypertext, presentation tools, collaborative work.

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

ACTIVE PATHS THROUGH MULTIMEDIA DOCUMENTS

POLLE T. ZELLWEGER

Xerox Palo Alto Research Center

ABSTRACT

We have developed a scripting mechanism for creating active paths through a document or set of documents. Scripted multimedia documents can contain a combination of text, graphics, audio, and actions. Scripts can be used in a wide variety of ways, such as for formal demonstrations and audio-visual presentations, for informal interpersonal communications, and for organizing collections of information. Scripted documents are a dynamic form of hypermedia document whose additional structure can be layered on top of existing documents.

1. Introduction

The advent of workstations has enabled a new and qualitatively different kind of document: a dynamic one that can incorporate audio and video in addition to text and graphics, one that can present itself in different ways depending on the needs or wishes of a particular reader at a particular time, and one that can serve as the backbone of a variety of "computations" that a reader might wish to perform. A prototype system for investigating these new capabilities has been built in the Cedar programming environment [Swinehart86] at the Xerox Palo Alto Research Center using the capabilities of the Etherphone voice system [Zellweger88] and the Tioga text editor [Teitelman84, Beach85].

1.1. *Overview of scripted documents*

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, such as a contiguous sequence of characters in a text document, together with an associated action and timing. Sample actions might play back a previously-recorded voice annotation, send text to a text-to-speech synthesizer, open a new window, animate a picture, or query a database. Script specifications are stored in a shared database separately from the underlying documents. A single document can have multiple scripts traversing it for different purposes.

A script can be played back as a whole, in which case the first document in the script is displayed on the screen, positioned to show the first location. The location is highlighted to call attention to it, and its associated action is performed. After the associated timing, the system highlights the location of the next script entry, scrolling if needed, performs its action, and so on. The same document location can appear at multiple points in the script, with the same or different associated actions and timing.

Arbitrary actions at a scripted location allow scripted documents to perform a wide variety of tasks, including demonstrations, tutorials, and programming tools. Parameterized actions allow a script to be personalized, such as "Hello <username>," or to more accurately reflect the current state of affairs, such as "There are <currentNumber> entries in this category." Scripts can be very formal items that are carefully crafted for pedagogical reasons, such as a videotape or a presentation, or they can be informal, used to communicate from a single script writer to a single script reader (who might well be the same person).

1.2. *Related work*

The capabilities of electronic documents have been expanding rapidly in recent years. Multimedia document systems have extended the contents of documents from text and formatting to include bitmap images, geometric graphics, spreadsheets, attributes, and voice [Ades86, Crowley87, Luther87]. Hypertext systems allow users to link non-contiguous portions of documents to express the associations between them [Delisle86, Halasz86, Meyrowitz86, Conklin87]. Electronic books and encyclopedias combine multimedia (text, graphic illustrations, and sometimes video) and hypertext (to link related sections) and may also include interactive portions in which readers can run simulations or other experiments to improve their understanding [Feiner82, Weyer85, Yankelovich85]. Presentation tools make it possible for users to capture sequences of actions to create automatic demonstrations [Xerox85]. Other document systems have included animation [Fiume87], actions [Hogg84], or sequencing [Christodoulakis86]. These advanced capabilities have been provided in several ways: by example, by direct manipulation, or by some form of programming language. The scripted documents system described in this paper forms a unique combination of multimedia documents, hypertext links, sequencing, and actions that increases document functionality still further.

2. Key ideas of scripted documents

This section discusses some of the key aspects of scripted documents, emphasizing ways in which scripted documents differ from typical hypertext systems.

2.1. *Directed paths versus browsing*

Most hypertext systems have concentrated on providing links between related nodes, creating an information space that a user can browse at will. The related concept of a *directed path* through a set of documents allows authors or script writers to include more structure. This structure consists of timed sequences of information, unrelated to the ordering of the underlying documents, to help the user understand the material. Although a few hypertext systems allow a sequence of links to be combined to form a path [Conklin87], we have instead made paths our primitive mechanism in order to explore the applications of sequentiality in electronic documents (see Section 3). For example, a later path entry can rely upon the user having seen earlier entries to overlay or animate previously-seen images or to abbreviate explanations.

There is a conceptual continuum from the directed mode of following paths to the browsing mode of exploring links. For example, although standard usage of scripted documents is to experience a path automatically, users can single-step through a group of directed paths to approximate the more conventional browsing paradigm. A more interesting combination of directed paths and browsing is an *interactive path*, in which the user answers questions to construct his or her desired path through the information. In fact, a hypertext link can be expressed as a degenerate script containing two locations that have no corresponding actions.

2.2. *Emphasis on voice*

As a result of the efforts of the Etherphone project, recorded and synthesized voice are widely available in the Cedar environment. Documents prepared with the Tioga editor can contain formatted text, graphics, and an unlimited number of arbitrary-length voice annotations. A direct-manipulation voice editor allows easy editing at the phrase or sentence level.

Voice is a critical component of a script. It provides a unifying thread for presentations and interpersonal communication, and it provides an out-of-band way to organize and comment on written material. Finally, it is easy to collect and modify (but as yet hard to search, although speech recognition systems are rapidly improving).

2.3. *Script information separate from underlying ordinary documents*

Separating scripts from the underlying documents enables an interesting mixture of public and private scripts. Scripts can refer to public documents without modifying those documents, allowing a user to create a personally organized information space (such as during an authoring task) without copying information into a closed hypertext system. Separating scripts from the underlying documents also allows for a smooth integration of scripted documents with other documents: scripts can gradually be added to a set of documents that continue to be accessed as ordinary documents.

2.4. *Arbitrary actions accompany document locations*
Script actions are unconstrained – they can call upon the power of the surrounding computing environment to execute commands or program fragments. The following section illustrates the resulting flexibility and power of scripts.

3. Examples of scripts

Scripts have a wide variety of uses. Script writers can create formal teaching materials and documentation. Users can build scripts that organize information or perform repetitive tasks. In addition, programs can construct scripts to ease complex tasks.

Teaching tool. A language dialog can be represented both textually and as a simultaneous sequence of voice annotations to demonstrate correct pronunciation. The same dialog can include multiple scripts, one for each language desired. Each script visits the same sequence of locations, but has different associated voice annotations.

Interpersonal communication. To review a manuscript, each reviewer can prepare a script for the manuscript, including voice annotations as well as branches through other supporting documents. Each script can follow an arbitrary path through the manuscript to collect related points. This use provides much of the value of a face-to-face interaction between the 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.

Personal information management. A user can create multiple scripts through a set of documents, each organizing a different topic. These scripts can be reordered as needed. The use of voice to annotate each location can be particularly helpful in early stages of idea exploration.

Audio-visual presentation. Voice, text, and graphics can be combined to simulate a "slide show" on a selected subject. Several versions of the slide show, including different or rearranged slides, can be constructed to accommodate different audiences or different time constraints.

User documentation and demonstration. Scripts can be used to explain how to use a complex program, such as a graphic illustrator. Actions can load and start the program, apply it to an example, visit noteworthy places in the user manuals, and explain some beginner's projects. Different scripts can be prepared for novice, intermediate, and expert users. Figure 1 shows the first few entries of such a script applied to the tutorial for the Gargoyle illustrator [Pier88].

"Bouncing ball" for songs or poems (looping actions). The text of a song or poem with a refrain can be successively highlighted with simultaneous audio, returning to the single copy of the text for the refrain as appropriate. Representing the song or poem in

this way emphasizes the identity of the refrain, so that the reader need not carefully compare repeated copies of a linear representation. Similarly, a presentation may repeatedly return to an overview slide to orient viewers or to emphasize important points.

Program-constructed scripts as history mechanisms. Programs can automatically construct scripts to act as history mechanisms for their users. For example, searching through annotations in a voice-annotated document can be tedious. An annotation system could automatically construct three scripts whenever voice annotations were added to a document. The first would connect all annotations to the document in document order. The second would connect all annotations to the document in the order that they were made. A third more ephemeral script would cross document boundaries to connect all annotations made during a single session in the order that they were made.

Scripts can also be used for collecting locations of interest rather than for their sequencing and/or action capabilities. Consider the following two examples.

Program debugging tool (repetitive actions). To provide an easy way of controlling a named group of breakpoints during program debugging, a collection of locations in several program files can be joined into a script with actions that set appropriate breakpoints: standard, conditional, profiling, tracing, etc. This script can be executed whenever a user wishes to activate those breakpoints. They can later be removed without disturbing other active breakpoints by overriding the script's stored actions with a separately-specified action that clears the breakpoint.

Program editing tool (empty actions). A compiler could build a script containing the locations of all the syntax errors in a group of files compiled together. The user could then single-step through the script, making corrections along the way. The script both makes it easy for the user to find the next error location and automatically manages the changing character positions as the user adds or removes characters to correct the errors.

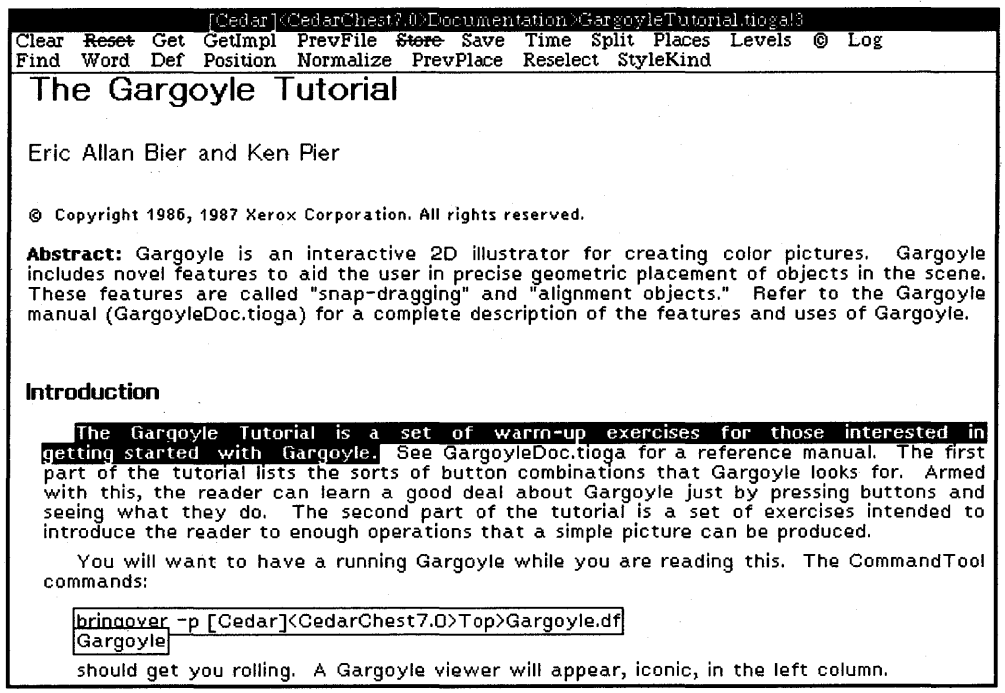
4. Creating and playing back scripts

4.1. *Script creation and editing*

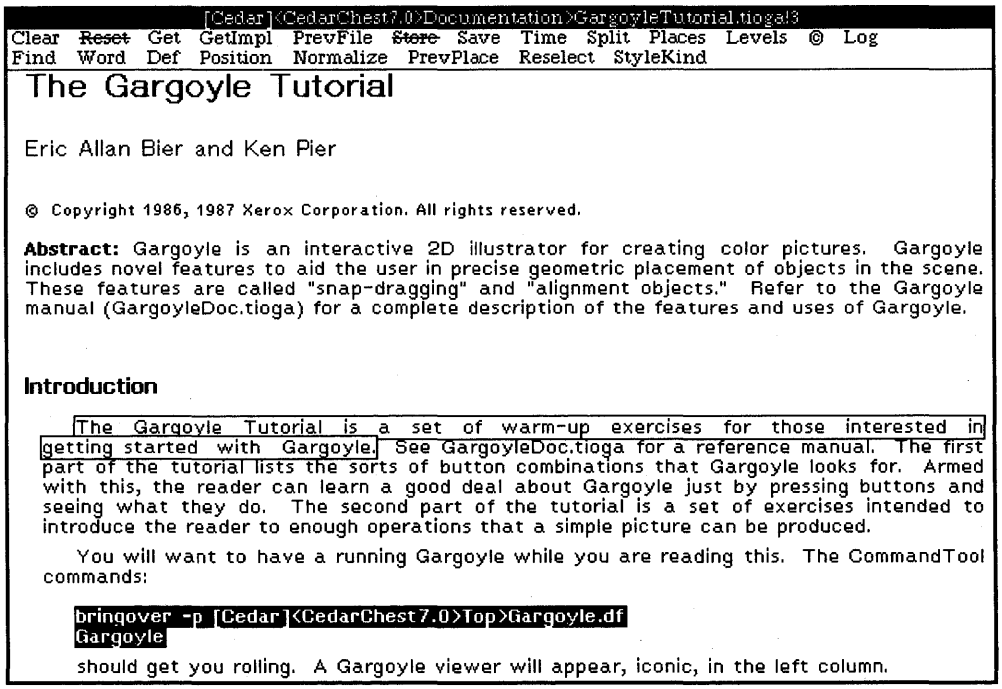
A script has two parts: the script entries, in which the script writer specifies scripted locations, actions, and timing; and the script header, in which the script writer specifies the script name and a simple program describing the sequencing of the entries. Separating the sequencing information allows the same script entry to appear in multiple scripts or multiple times in a single script.

A script tool is used to create and edit scripts. The prototype Script Tool is a form-based tool with a simple set of operations.

Figure 1a: *A sample user documentation script, part 1.*

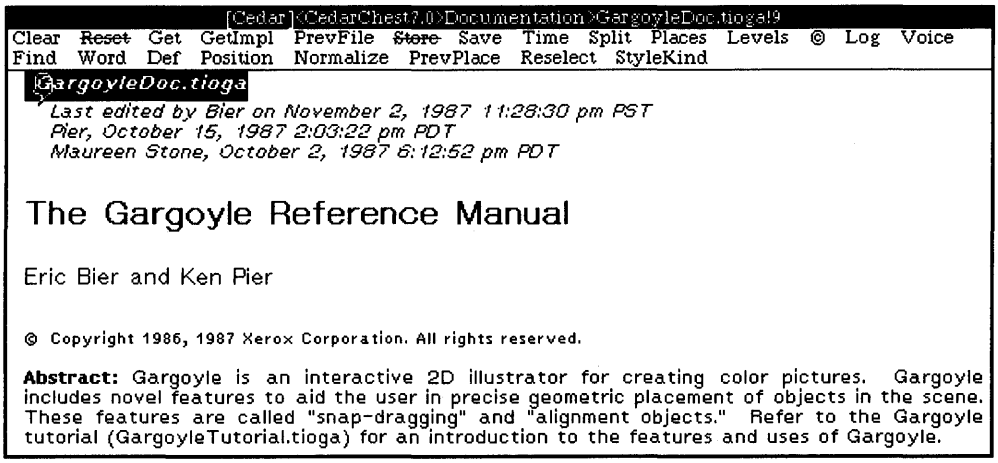


Sample script at the first entry. This action uses the text-to-speech synthesizer to greet the user by name, welcoming him or her to the Gargoyle tutorial.

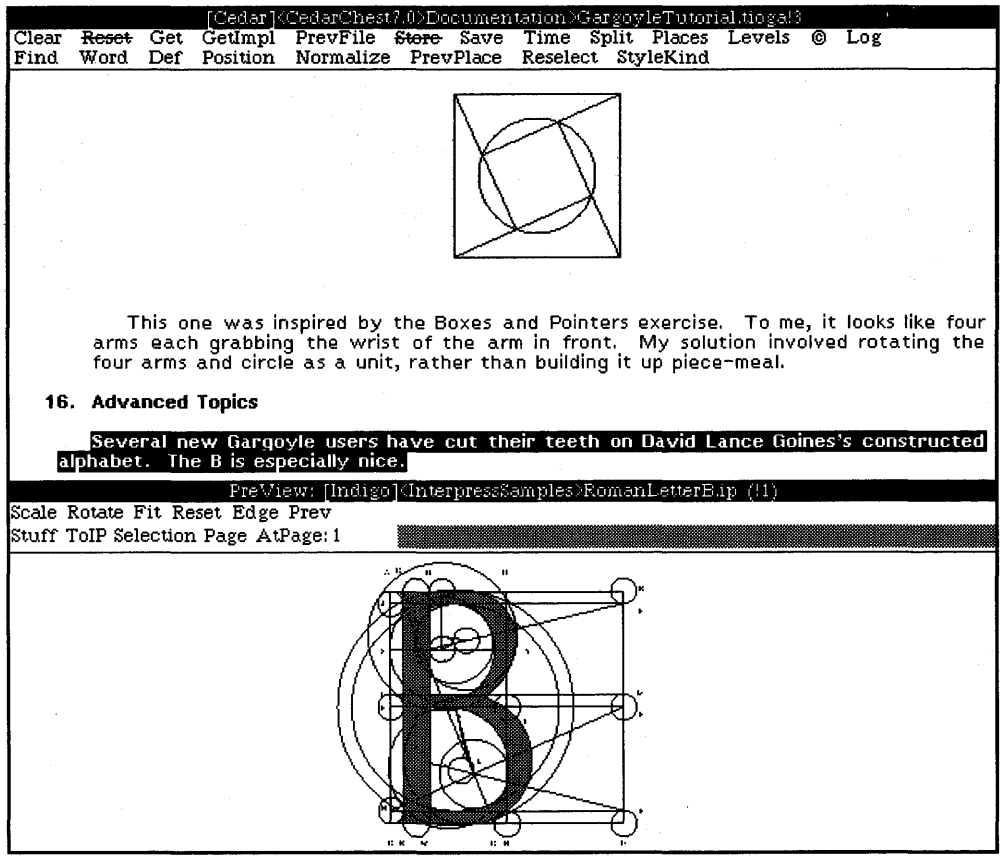


Sample script at the second entry. This action retrieves the executable files of the Gargoyle illustrator and starts the program (the same actions that the text is instructing the user to perform).

Figure 1b: *A sample user documentation script, part 2.*



Sample script at the third entry. This is a different file. The action plays back the previously-recorded voice annotation on the first character of the scripted location (a tiny "word balloon" around the character G indicates the presence of voice). The voice annotation explains the difference between the Gargoyle tutorial and the Gargoyle reference manual.



Sample script at the fourth entry. The script has returned to the tutorial document. The action displays on the screen an Interpress master for an image that was not included in the tutorial itself.

Script entries. To create or edit a script entry, the script writer specifies its action and timing fields by filling in the fields of a script entry form, sets its location by making a screen selection, and then saves the form. The user can name a script entry or provide keywords for later filtering, if desired. The system provides a unique identifier (id) for each script entry, records the creator and the create time, and writes the script entry to the script database. System defaults allow simple script entries to 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.

Action field. Any Cedar system command can appear in the action field of an annotation. Since system commands can invoke the Cedar language interpreter, arbitrary Cedar language statements can also be included.

Timing field. The script system continues to the next script entry when the specified duration has passed, regardless of whether the action has completed. The script writer can specify "*" to continue whenever the action completes or "~" to wait for user confirmation before continuing.

Script sequencing. The sequencing information for a script is specified separately from the script's entries. For a strictly sequential script, it consists of an ordered list of script entries (as unique ids). A script writer can create a sequential script by pointing to the script entries in order. To change the ordering of a sequential script, the script writer indicates the script entry that a selected entry should follow. (This situation can become a bit more complicated, because script entries can appear multiply in the same script, creating ambiguities in both the position being altered and the desired new position.) To create a more complex sequence, the script writer can edit the textual representation of the sequence to add loops, conditionals, or calls to other scripts.

Figure 2 shows a simplified internal representation of the script that appears in Figure 1.

4.2. *Script playback*

The Play command plays an entire script from the beginning, proceeding automatically from one location to the next, executing the associated actions with the associated timings. The user can pause an executing script at the end of the current action or abort the current action. A quiescent script can be continued or single-stepped. The user can also play a script backward or single-step backward from the current location. If the script contains branches or loops, the session history will be used to construct the backward path.

A separate property sheet controls additional playback options. The user can inhibit the actions associated with annotations (so as simply to traverse through the scripted locations), specify a single action to be executed at every scripted location, and increase or decrease the timing by some factor.

Figure 2: *Simplified internal representation of the script for Figure 1.*

Script header

```
script header id: 12345677 # PolleZ.pa
script name: Introduction to Gargoyle
file names:
   /cedar/documentation/GargoyleTutorial.tioga
   /cedar/documentation/GargoyleDoc.tioga
script sequence:
  12345678 # PolleZ.pa   12345679 # PolleZ.pa
  12345680 # PolleZ.pa   12345681 # PolleZ.pa
```

Script entries

```
script entry id: 12345678 # PolleZ.pa
filename: /cedar/documentation/GargoyleTutorial.tioga
class: Tioga text
location: 15..121
action: speak("Hello, ", UserCredentials.Get[].name,
   ". Get ready to blast off.")
time: 15 sec

script entry id: 12345679 # PolleZ.pa
filename: /cedar/documentation/GargoyleTutorial.tioga
class: Tioga text
location: 624..683
action: bringover -pm Gargoyle.df; Gargoyle
time: *

script entry id: 12345680 # PolleZ.pa
filename: /cedar/documentation/GargoyleDoc.tioga
class: Tioga text
location: 1..17
action: play(annotations)
time: 10 sec

script entry id: 12345681 # PolleZ.pa
filename: /cedar/documentation/GargoyleTutorial.tioga
class: Tioga text
location: 55417..55537
action: PreView RomanLetterB.interpress
time: *
```

4.3. *Script visualization and navigation*

Script visualization is a problem for both the script reader and the script writer, because script sequencing and script actions are not directly visible through examination of the document. The script reader must be able to tell that a document has associated scripts and which scripts are appropriate for him or her to play back. The script writer has the more difficult problem, in that he or she must also have tools for debugging faulty scripts.

The script reader is alerted to the presence of scripts in a document in two ways. First, when a document is displayed, a Script button appears in the document header if and only if the document has scripted locations. All script examination and playback operations are available from a popup menu generated by clicking the Script button. Second, 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. The user can view all script entries that include that scripted location or list the names of all scripts that include it. The reader can also ask what scripts have entries in a given document.

In addition, a user can browse the script database for script names, script entry names, keywords, or any other script field. Other navigation commands show the user the document location associated with a script entry, all scripts the entry belongs to, and all possible preceding or following entries in a given script.

5. Implementation

Scripted documents are implemented in the Cedar programming environment in the Computer Science Laboratory of the Xerox Palo Alto Research Center [Swinehart86]. The two systems that they rely upon most heavily are the Tioga editor and the Etherphone system. We describe each of these systems briefly, and then we describe the implementation of scripted documents.

5.1. *The Tioga editor*

The Tioga editor is a WYSIWYG "what you see is what you get" galley editor used for both program text and high-quality documents [Teitelman84, Beach85]. Tioga documents are tree-structured to express the organization of the document into sections, subsections, paragraphs, and so on. Tioga documents can be displayed at any level of detail, omitting nodes that are nested more deeply than the selected level.

Tioga documents can contain rich formatting and typography. Each node has an associated format, which specifies such parameters as its leading, margins, default typeface, and so on. A document as a whole has an associated style, which defines the

meanings of all formats used in that document. Nodes can also have arbitrary named properties, added by the user or by programs. These properties are not directly visible when the document is viewed, but a separate Edit Tool can be used to examine their values. One use of node properties is to specify images, Interpress masters, and additional parameters to support illustrations embedded in Tioga documents.

Individual characters in a Tioga document can have looks, which specify special typeface parameters such as boldface, italic, subscript, and the like. Characters can also have arbitrary named properties. Tioga has search commands that allow rapid searching of documents for given node or character properties.

5.2. *The Etherphone system*

The experimental Etherphone system uses Ethernet communications to transmit digitized voice [Zellweger88]. The system consists of microprocessor-based electronic telephones, a centralized switching server, a voice file server, and workstation programs to support voice communications and voice recording services. From a workstation, a user can place and receive telephone calls, maintain private telephone directories, and manage a database of voice messages. A voice annotation package allows voice to be added to Tioga documents and provides a simple direct-manipulation interface for editing voice [Ades86]. Furthermore, a commercial text-to-speech synthesizer exists as a server in the Etherphone network. The synthesizer allows the system to "speak" text, initiated either by the user (perhaps by selecting the text in a viewer) or by a program (such as speaking an error message or proofreading a document).

This work on scripted documents began as a project to exploit the capabilities of the Etherphone system by creating narrated documents. Narrated documents are scripted Tioga documents with a single type of action: playing back previously-recorded voice annotations.

5.3. *Scripted documents*

The two parts of a script specification, the script entries and the script sequencing information, are stored separately from the underlying documents in a simple B-tree-indexed database [Terry88]. The database's ability to merge the results of queries to multiple databases allows users to simultaneously access private databases containing their private scripts and public databases containing public scripts.

5.3.1. *Tagged and absolute references to scripted locations*

An important goal of the scripting system was to allow script writers to create script entries that refer to any document they can access, including documents they cannot or do not wish to modify (hereafter called *read-only* documents). This capability is especially important when scripts are being used as an organizational tool, such as

during an authoring task. Ideally, the owners of a scripted document would also be able to edit unscripted portions of the document without damaging the script.

Given our additional desire to use the existing version-based file system in our widely-distributed environment, we achieve reasonable functionality by allowing script entries to contain two different kinds of references to scripted locations. *Absolute references* refer to precise locations in documents, such as character or byte positions, while *tagged references* refer to uniquely tagged and hence movable objects within documents.

The initial prototype of the scripting system used only tagged references and stored each document's script entries within the document. However, the desire to script read-only documents suggested database storage for their script entries, and it proved more convenient in the later implementation to store all script entries together.

Absolute references are used to refer to items within read-only documents. The scripting system records the scripted location and the exact timestamp and version of the containing file. The file is considered immutable: if it is subsequently updated, the script will continue to refer to the older version. Note that many public documents, such as user documentation and reports, are likely to change less frequently than personal documents. We are exploring ways to handle such documents more robustly, such as also recording a signature containing the scripted location to permit the location to be found in the new version if it exists.

By contrast, creating a tagged reference to a scripted location modifies its containing file. The scripting system writes the unique id of the script entry in the scripted object. Tagged references allow other portions of the file to be edited without disturbing the scripted location, even when the scripting system is not running. However, editing the scripted location itself generally destroys its pointer to the script entry. These semantics are reasonable, because it is not clear that the new version of the scripted location is related to the old script entry. For example, consider scripting the word "reindeer" in a document to have an action that plays a bit of "Rudolph the Red-nosed Reindeer," and suppose that later edits change "reindeer" to "moose."

For either kind of reference, if a scripted location cannot be found during playback, the system displays an informational message and continues with the next script entry.

5.3.2. *Scripting different kinds of document content*

The design of the scripting system is object-oriented to permit scripting a variety of documents, such as Tioga documents and VLSI diagrams, and a variety of contents, such as text, bitmaps, and synthetic graphics. Each class of visual object must implement the following functions:

identify [object] -- *returns class and location, suitable for storing*

add [object, unique id] -- *tag object with unique id*

remove [object, unique id] -- *remove tag*

showWithHighlight [filename, filedate and version, location, unique id]

-- *used to highlight the object when it is the script entry being executed*

The identify function takes a selected object, such as a sequence of text characters or a piece of a bitmap, and returns both the class of that object and its location (a persistent way of addressing it), suitable for storing in a script. For a sequence of text characters, it returns a character position and a length, while for a piece of a bitmap, it returns the coordinates of its bounding box. This function supports absolute references to scripted objects and provides a hint and a printable value for the location of a tagged object.

The add function supports tagged references to scripted objects by writing into the specified object a unique id that identifies a script entry. For example, for text in a Tioga file, the add function uses Tioga's ability to associate arbitrary property-value pairs with any character. The remove function is the inverse operation.

A class's showWithHighlight function finds an object in a file using either a filedate and location (for an absolute reference) or a unique id (for a tagged reference). This function positions the file so that the corresponding object is visible and highlights the object.

5.3.3. *Making scripted locations visible*

To avoid unknowingly destroying scripted locations, script writers and other document editors must be able to see them. To make tagged references visible, the scripted object's add procedure is responsible for adding the visible scripted location indicator to the object. Similarly, the remove procedure removes the indicator when it removes the script entry id. Making absolute references visible presents more of a challenge. The script system must consult the database whenever a new remote file is opened to see if absolute references to this file exist. If so, it constructs a view of the scripted read-only document with script indicators added.

6. Status and future work

The initial prototype of the scripting system was designed for creating narrated documents. Scripts performed a single action, namely playing back voice annotations, at each scripted location. Each script was constrained to refer to a single document, although a document could contain multiple scripts. Script entries were stored in the document header and contained their own sequencing information; tagged references referred to scripted locations throughout the document. A simple script tool allowed

users to create, edit, play back, and navigate through scripts.

The current prototype separates sequencing information from the remainder of the script entry (to permit multiple scripts to share the same entry) and stores both in a database (to allow scripting read-only documents and to make it easier to refer to multiple documents in a single script). The system has been redesigned to permit scripting different classes of document content, but text is the only class that is currently handled. The language for describing complex sequencing is still in its infancy: the scripting system currently allows only sequential scripts, although they may visit the same location repeatedly. The implementation of the scripting system uses a variety of previously-existing Cedar packages: the Tioga text editor, the Etherphone telephone and voice management system, the LoganBerry database system, the Command Tool command interpreter (with its ability to execute Cedar language statements), and the FS version-based file system. Additional user interface features are still needed, as they have not progressed much from the earlier prototype. In particular, the more advanced user playback options, such as backward playback and variable playback timing control, have not yet been designed.

We are working on extending our prototype scripting tool to allow conditional and/or interactive scripts; better visualization of scripts for both script readers and writers, including browsing tools and visual displays of script sequencing; better control of screen and document layout at each script entry during script execution; and scripting other classes of documents, such as graphic illustrations or VLSI layouts.

7. Summary

The scripting concept unifies action, presentation, and hypermedia to form a useful and flexible mechanism for improving documents of the future. This novel mechanism allows writers to communicate additional information to readers. Scripted multimedia documents can contain any combination of text, graphics, audio, and action. Scripts need not follow the normal linear order of their associated documents. In addition, script writers can construct multiple viewing paths through documents for different readers and for different purposes.

The scripting mechanism can be widely applied to create electronic documents with increased capabilities. Scripted documents can orchestrate formal presentations, arrange informal communications, organize collections of information, and ease repetitive or complex tasks.

Acknowledgments

Dan Swinehart and Stephen Ades provided the initial impetus to create narrated documents. Jock Mackinlay suggested several improvements to the design of the later

scripted documents system. While I would like to thank the many contributors to the Cedar programming environment as a group, I would also like to single out a few whose work has been particularly critical for the scripting system: Rick Beach and Michael Plass provided consultation on its interaction with Tioga, Doug Terry's LoganBerry database package supplied just the needed functionality, and Russ Atkinson's efforts on the Command Tool and Cedar interpreter were invaluable. Pavel Curtis, Subhana Menis, and Ken Pier helped to clarify the exposition of this paper.

References

[1] Ades, S. & Swinehart, D. (1986). Voice annotation and editing in a workstation environment, *Proc. AVIOS'86 American Voice Input Output Society Conf.*, Arlington, VA, 13-28. Also available as Xerox PARC Technical Report CSL-86-3.

[2] Beach, R. (1985). *Setting tables and illustrations with style*, Ph.D. thesis, U. of Waterloo, Canada. Also available as Xerox PARC Technical Report CSL-85-3.

[3] Christodoulakis, S., Ho, F. & Theodoridou, M. (1986). The multimedia object presentation manager of MINOS: a symmetric approach, *Proc. ACM SIGMOD'86 Conf.*, Washington, DC, 295-310.

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

[5] Crowley, T., Forsdick, H., Landau, M. & Travers, V. (1987). The Diamond multimedia editor, *Proc. USENIX Technical Conf.*, Phoenix, AZ, 1-18.

[6] Delisle, N. & Schwartz, M. (1986). Neptune: a hypertext system for CAD applications, *Proc. ACM SIGMOD'86 Conf.*, Washington, DC, 132-143.

[7] Feiner, S., Nagy, S. & van Dam, A. (1982). An experimental system for creating and presenting interactive graphical documents, *ACM Trans. Graphics*, **1**, 1, 59-77.

[8] Fiume, E. & Tsichritzis, D. (1987). Multimedia objects, *IEEE Office Knowledge Engineering Newsletter*, **1**, 1, 60-64.

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

[10] Hogg, J. & Gamvroulas, S. (1984). An active mail system, *Proc. ACM SIGMOD'84 Conf.*, Boston, MA, 215-222; also *SIGMOD Record*, **14**, 2.

[11] Luther, W., Woelk, D. & Carter, M. (1987). MUSE: multimedia user sensory environment, *IEEE Office Knowledge Engineering Newsletter*, **1**, 1, 49-59.

[12] Meyrowitz, N. (1986). Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework, *Proc. OOPSLA'86 Object-Oriented Programming Systems, Languages and Applications Conf.*, Portland, OR, 186-201; also *SIGPLAN Notices*, **21**, 11.

[13] Meyrowitz, N. & van Dam, A. (1982). Interactive editing systems: part 1, *Computing Surveys*, **14**, 3, 321-352.

[14] Pier, K., Bier, E. & Stone, M. (1988). Gargoyle: an interactive illustration tool, *Proc. EP'88 Int'l Conf. on Electronic Publishing, Document Manipulation, and Typography*, Nice, France.

[15] Swinehart, D., Zellweger, P., Beach, R. & Hagmann, R. (1986). A structural view of the

Cedar programming environment, *ACM Trans. Programming Languages and Systems*, **8**, 4, 419-490.

[16] Teitelman, W. (1984). A tour through Cedar, *IEEE Software*, **1**, 2, 44-73.

[17] Terry, D. & Swinehart, D. (1988). Managing stored voice in the Etherphone system, to appear in *ACM Trans. Computer Systems*, **6**, 1. An extended abstract appears in *Proc. of Eleventh ACM Symposium on Operating System Principles*, Austin, TX, 1987, 103-104.

[18] Weyer, S. & Borning, A. (1985). A prototype electronic encyclopedia, *ACM Trans. Office Information Systems*, **3**, 1, 63-88.

[19] Xerox Corporation. (1985). *ViewPoint 1.0 Demo Maker tool*, Xerox Corporation, P.O. Box 470065, Dallas, TX 75427.

[20] Yankelovich, N., Meyrowitz, N. & van Dam, A. (1985). Reading and writing the electronic book, *IEEE Computer*, **18**, 8, 15-30.

[21] Zellweger, P., Terry, D. & Swinehart, D. (1988). An overview of the Etherphone system and its applications, *Proc. 2nd IEEE Conf. on Computer Workstations*, Santa Clara, CA.