Semi-Structured Capture and Display of Telephone Conversations

by

Debby Hindus

B.S. Computer Science
University of Michigan
Ann Arbor, Michigan
1987

SUBMITTED TO THE MEDIA ARTS AND SCIENCES SECTION, SCHOOL OF ARCHITECTURE AND PLANNING, IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE DEGREE OF

MASTER OF SCIENCE

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1992

Signature of Author:

.... [illegible]

Debby Hindus
Media Arts and Sciences Section
January 17, 1992

Certified by:

....

Chris Schmandt
Principal Research Scientist
Thesis Supervisor

Accepted by:

.... [illegible]

Stephen A. Benton
Chairman
Departmental Committee on Graduate Students

Semi-Structured Capture and Display of Telephone Conversations[1]

by

Debby Hindus

Submitted to the Media Arts and Sciences Section, School of Architecture and Planning on January 17, 1992, in partial fulfillment of the requirements for the degree of Master of Science at the Massachusetts Institute of Technology

Abstract

Speech dominates day-to-day communication, but despite technological advances, ordinary conversations have remained outside the realm of computer-supported work. This thesis addresses *semi-structured* audio, a framework for medium-specific information retrieval of stored voice that does not rely upon knowledge of the actual content. Instead, structure is derived from acoustical information inherent in the stored voice and augmented by user interaction. To demonstrate semi-structured audio, two software applications were constructed: the Listener, a tool for capturing structure while recording telephone conversations, and the Browser, for subsequent browsing of speech fragments.

The Listener segments the audio signal, using changes in who is speaking to identify conversational turns, and pause detection to identify phrase boundaries. This conversational structure is dynamically determined during the phone call, and presented in a retrospective display that provides flexible capabilities for marking segments of interest. The Listener and Browser make use of the ChatViewer widget, which supports the display of, and user interaction with, collections of sound and text items called *chats*. This semi-structured approach makes it practical to retain and access large amounts of recorded speech.

Thesis Supervisor: Chris Schmandt
Title: Principal Research Scientist

[1]This work was supported by Sun Microsystems, Inc.

Acknowledgements

I would like to thank my advisor, Chris Schmandt, for his guidance and his many good ideas.

Wendy Mackay has been a friend and source of inspiration throughout my career, and she and Kate Ehrlich helped me formulate the original concept behind this thesis. Mike Hawley and Janet Cahn have shared with me their enthusiasm and their insights into audio and conversation, for which I am grateful.

I am indebted to Mark Ackerman, for the hours he spent instructing me in the art of widget writing, for his technical expertise and keen perception of research issues, and for his friendship and understanding during the past three years. Joe Marks has also helped me retain a sense of humor and perspective on my work, and I feel fortunate to have known them both as collaborators and colleagues.

Thanks, too, to Barry Arons, who provided so much practical advice and was the developer of the Soundserver; to Chris Horner, for his contributions to the ChatViewer widget; and to Eric Ly, for his constructive feedback on displays and diagrams. I would like to acknowledge Don Jackson of Sun and Eric Hulteen of Apple, for their comments on the software over the past year.

I would also like to express my gratitude to Hal Abelson and David Reed, who initially fostered my pursuit of graduate studies.

My friends, Jo-Anne Wyndham, Leslie Rubenstein and Dave Gladstein, have given me their comfort and strength, and kept me in touch with life beyond MIT. Amy Bruckman, Michelle Fineblum, Lisa Neal, Sanjay Manandhar and Chi Wong were invaluable confidants and comrades.

To my immediate and extended families I extend my love and appreciation. especially to my mother, who has always encouraged me, and to my late father, who would not have been surprised.

Throughout everything, I have been sustained by the constant love and support of my husband, Bob Deloria. I could not have gotten this far without him.

Contents

1 Introduction 7

1.1 The Ubiquity of Speech . . . 8

1.2 Computer Workstations and Speech Data . . . 9

1.3 Overview of This Document . . . 10

2 Semi-Structured Audio for Telephone Calls 11

2.1 Background . . . 11

2.1.1 Structure in Applications of Stored Voice . . . 11

2.1.2 Visual Representations of Speech . . . 12

2.1.3 The Structure of Telephone Conversations . . . 14

2.1.4 Computers and Telephony . . . 15

2.1.5 Previous Work in Semi-Structured Media . . . 16

2.2 A Demonstrational Approach to Semi-Structured Audio . . . 17

3 The Capture and Display of Telephone Conversations 19

3.1 System Overview . . . 19

3.1.1 Software Components . . . 20

3.1.2 Software Environment . . . 22

3.2 The Listener Tool . . . 22

3.2.1 The Listener is Ubiquitous . . . 22

3.2.2 The Dynamic Display of Conversational Structure . . . 24

3.2.3 Identification of Interesting Segments . . . 24

3.2.4 Post-Call Review and Revision . . . 26

3.3 The Browser Tool . . . 27

3.3.1 Multiple Views . . . 27

3.3.2 Replaying Segments . . . 28

3.3.3 Editing Stored Conversations . . . 29

4 Turntaking and Segmentation 30

4.1 Workstation Audio . . . 30

4.1.1 Workstation Capabilities . . . 30

4.1.2 Workstation Audio Support . . . 31

4.2 How the Listener Determines Segments . . . 32

4.2.1 Acting on Energy Events . . . 33

4.2.2 Pause Detection . . . 35

4.2.3 Turn Detection . . . 36

4.3 Issues in Segmentation . . . 39

4.3.1 Delays in Updating the Display . . . 39

4.3.2 Pause Detection 40
4.3.3 Synchronization 40

5 The ChatViewer Widget **42**
5.1 The Nature of Widgets 43
5.2 Using a ChatViewer within an Application 43
5.2.1 Chatfiles 46
5.3 Implementation 47

6 Future Directions **51**
6.1 Contextual Structure and Strategies for Retrieval 51
6.1.1 Contextual Items 52
6.1.2 Browsing Through Multiple Chats 52
6.2 User Observations 53
6.2.1 Practical Considerations for a Pilot Study 54
6.3 Controlling Segmentation Sensitivity 55
6.4 A Handful of Audio Processing Ideas 55
6.5 ChatViewer Variations 57
6.6 Other Software Improvements 58

7 Discussion **62**
7.1 What Was Learned and Not Learned 62
7.2 Applications 63
7.3 Impact on Users and Organizations 64
7.4 A Final Word 65

A User Documentation **66**
A.1 Browser Man Page 66
A.2 Browser Widget Tree 67
A.3 Browser Resource File 67
A.4 Listener Man Page 70
A.5 Listener Widget Tree 71
A.6 Listener Resource File 72

B ChatViewer Reference **75**
B.1 Synopsis 75
B.2 Resources 75
B.3 Public Data Structures 77
B.4 Public Routines 77

Bibliography **81**

List of Figures

2.1 Three different visual representations of a 13-second long utterance 13
2.2 Apple's SoundBrowser representation, from [DMS92] 13
2.3 Etherphone's sound-and-silence representation, from [AS86] 14

3.1 Listener Software Architecture 20
3.2 Receiving a phone call 23
3.3 Sequence of segments during a phone call, with transcriptions 25
3.4 Browsing after a phone call 27
3.5 The Browser, displaying the saved portions of the sample conversation . . 28
3.6 A closeup of a SoundViewer, with a playback in progress 29

4.1 Processing energy events in the Segmenter 34
4.2 Example of a 300-msec sliding window used in pause detection 36
4.3 State transition diagram for determining segments 38
4.4 Silences are divided between segments 38

5.1 Examples of ChatViewer layout styles 44
5.2 Sample chatfile 47
5.3 Sample ChatViewer database record 48

Chapter 1

Introduction

Speech is second nature in our everyday lives and computing has become ubiquitous in our work lives within the last decade. This combination presents tremendous opportunities that have yet to be exploited. As Moran and Anderson [MA90] point out, technologies are mundane in the workaday world, and are successful when ordinary activities, such as telephone calls, interact seamlessly with technology. In the case of telephone calls, we have all wished at one time or another that we had tape-recorded something that was said. Imagine having a tool on your workstation that would ensure important conversations would not be lost, yet not burden you with hours of extraneous recorded audio to plow through just to find that crucial statement.

This thesis addresses that problem: the theoretical and practical issues of applying computer technology and the concept of semi-structured audio in the context of a specific domain, that of telephone conversations. It addresses creation of structure at the time of capture, the visual representation of free-form conversations, dynamic displays for real-time capture, and browsing and retrieval interactions. The work features the Listener, a telephone listening tool that allows users to identify and save the relevant portions of telephone calls as the phone conversation progresses, and a browser for reviewing and modifying previously recorded conversations.

The Listener captures structure from telephone calls, as described in the following scenario:

- You place a phone call. The Listener notices this, and displays a dialog window from which you can choose to record the conversation.
- While you're talking, a graphical representation of the conversation is constructed on your screen, showing the shifts in who is speaking and the relative length of

each turn.

You can click on a segment to indicate that it should be saved, or tie it to a text label.

- At the end of the phone call, you can listen to segments and decide which ones to save.

  If you want to, you can add a text summary of the conversation.

- Later on, the saved conversation can be reviewed and further pruned or annotated.

- At any time in the future, the conversation can be retrieved and played back.

In order to make that scenario real, I built a listening tool for telephone calls, referred to in this document as the Listener, and a browser (the Browser) for stored conversations. Listening tools capture inherent structure by taking advantage of situational constraints, capturing available contextual data, and analyzing the speech signal as appropriate.

When computers become more capable of participating in our spoken interactions, as intermediaries, note takers, or digital adjuncts to personal memories and recollections, they have the potential to become deeply involved in daily life. This project is but a step towards making audio data on the workstation as integral a part of the workaday world as text is now.

1.1 The Ubiquity of Speech

People spend much of their work day talking. In a recent study of the work activities of three groups of professionals within a Fortune 500 company, Reder and Schwab [RS90] found that phone calls comprised about 20% of the workday, and face-to-face meetings accounted for an additional 25-50%. Although text editing programs and desktop publishing systems are commonplace in offices, and messages and documents are increasingly communicated via electronic mail, the time spent talking has been out of reach of computer technology. Only formal meetings are candidates for being transformed into a computer-accessible form, as written minutes. The remaining verbal interactions, and many meetings, are ephemeral unless, after the conversation, a participant goes to the trouble of composing a summary online.

This loss of speech information is all the more striking in the light of studies of communication outcomes by Chapanis [OC74] and others that convincingly demonstrate the utility of voice communication. Furthermore, speech communication fulfills different communicative purposes than text communication; Chalafonte, Fish and Kraut [CFK91] concluded that speech is a rich, expressive medium that "is especially valuable for the more complex, controversial, and social aspects of a collaborative task."

1.2 Computer Workstations and Speech Data

Technological advances have enabled workstation manufacturers to provide support for audio digitization and playback, and these capabilities are now found on many workstations. But despite the combination of available technology and the power of speech as a communication medium, computer applications of stored voice are in their infancy. This is largely due to retrieval problems; unlike text, speech cannot yet be freely searched for keywords.

Stored voiced is bulky and difficult to access. Audio is a temporally-constrained medium; unlike text, it cannot be skimmed, and requires more time and attention to peruse than would a visual scan of a page of text. Speeding up the playback of the recorded speech is helpful, but limited; speech becomes significantly less comprehensible when it is compressed beyond 50% of the original time [BM76, FS69][1]. Listening is much slower than reading, though writing takes more effort than speaking. In sum, audio is biased towards the speaker, not the listener.

As a result, current applications of stored voice are limited to two distinct styles of audio data. The first consists of short message segments that are not meant to be listened to repeatedly, as in voice mail systems. The second is highly structured audio segments, such as in hypermedia systems, where the author applies links between data nodes and the application provides a user interface for navigation.

What about managing large quantities of more casual recorded conversations? Retrieval can be assisted by the application of structure in this domain as well, but without understanding the words actually spoken, such structure must be derived from acoustical information. With such an approach, the audio data can be automatically separated into meaningful pieces, or *segments*, and these segments can be classified without knowledge of their semantic content. The result is a framework for describing and manipulating the recorded conversation, and the framework, together with the audio data, can be characterized as *semi-structured audio.* Some semi-structuring is automatic, and some requires the cooperation of users. Structuring may occur when the audio is originally captured, that is, during the conversation; immediately following the

[1]Simply playing samples at a faster clock rate shifts the pitch of the sound so that it becomes squeaky, like the voice of the cartoon character Mickey Mouse. More sophisticated methods of time scale modification remove complete pitch periods during playback.

conversation, while recollection of it is still fresh; and during later retrieval and editing operations.

1.3 Overview of This Document

The rest of this document describes the background, implementation and results of this research. Chapter 2 discusses five areas of converging technologies and interfaces that have led to semi-structured audio in the context of telephone conversations. The nature of telephonic conversation is explored, along with the role of semi-structured audio in telephone interactions. Chapter 3 begins the material on implementation of the tool set with a system overview. It next describes the telephone listening tool and the browser. Chapter 4 discusses how the listener determines which participant is speaking, and how the conversation is broken up into segments. Chapter 5 describes the ChatViewer widget, an underlying software object created to support the other tools.

The thesis ends with brief descriptions, in Chapter 6, of the various ways that this work ought to be extended and the related issues investigated, and a discussion, in Chapter 7, of the results and impact of this project. The Appendices contain user documentation for the Listener and Browser applications, and technical documentation of the ChatViewer widget.

Chapter 2

Semi-Structured Audio for Telephone Calls

This chapter discusses the areas of converging technologies and interfaces that have led to semi-structured audio in the context of telephone conversations. These supporting areas include: applications employing voice snippets, structured hypermedia documents, visual representations of time-dependent media, computer-assisted telephony, the nature of telephone conversations, and semi-structured approaches to stored voice and other media. The remainder of this chapter describes the approach taken towards semi-structured audio in this project.

2.1 Background

2.1.1 Structure in Applications of Stored Voice

Voice Snippets

Many current applications of stored voice succeed because voice segments are short and the application itself provides the structural framework for presentation of sound to the user. Voice mail is such an application. Voice messages are short, and intended to be short-lived; this is enforced by expiration dates for messages on many voice mail systems. A user interface presents messages sequentially, and may also indicate the sender and when the message was sent. Another application that uses voice snippets is Caltalk [Sch90], which allows users to record voice entries into their calendars; the date and time of the entry key it for later retrieval. Voice can also be used to annotate text, as was done in Quilt [FKL88] and Slate [Cro91]. When used for annotation, voice segments are correlated with particular locations in the document, and are displayed appropriately. Similarly, products have appeared to allow voice annotation of spreadsheets, where a voice snippet is associated with a spreadsheet cell. The stored voice is not itself structured in these applications; the recorded speech is treated as a single unbroken entity, and there is an external pointer to the sound, such as a message number, calendar date, or position within the text or spreadsheet.

Hypermedia

The applications just described work with short, often transitory, voice snippets that are organized simply. Hypermedia is a prominent application area that has made meaningful use of longer sound segments that are organized in complex ways and are intended to be accessed repeatedly. Hypermedia documents employ large quantities of text, sound and other media, with links between the data nodes and a user interface to navigate among them. Although most hypermedia systems to date have concentrated on text augmented with images, Arons [Aro91b] describes a voice-only hypermedia system employing recorded speech at the data nodes and utilizing speech recognition for navigation.

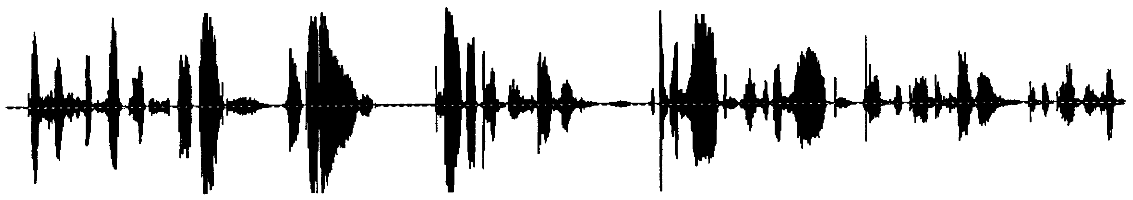
Hypermedia documents embody considerable structure, which is supplied during the authoring process. Muller [MD90] describes an architecture for voice documents, and Zellweger [Zel89] describes a means of authoring scripts or guided tours through hyperdocuments that can include voice. Both these examples illustrate the degree of structure that must be applied to make a medium such as audio useful when it is used in quantity.

2.1.2 Visual Representations of Speech

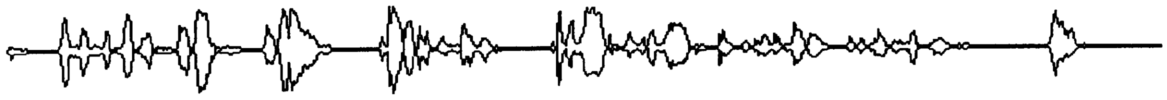
Speech is often represented as a waveform or average magnitude envelope, as shown in Fig. 2.1; although this intuitively looks like speech, it does not contain much useful information for typical users. This form treats speech as any other audio signal, and is required for fine-grained editing and other audio manipulations.

A somewhat different approach is employed by Apple's SoundBrowser; here, amplitude data is presented in a format designed to be understandable by non-expert users [DMS92]. An example is shown in Fig. 2.2.

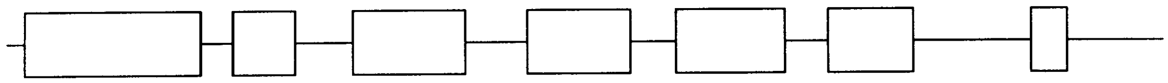
Speech consists of audible sounds interspersed with silences, such as the natural pauses at the end of sentences. Some visual representations make use of this aspect of



Waveform (from Mixview, on a Sparcstation)



Envelope (from SoundWorks, on a Next machine)



Speech and Silence (from Sun's Audio Tool)

Figure 2.1: Three different visual representations of a 13-second long utterance
Text: "This is an example of speech that shows how there can be spaces between words and within words, and at the end of phrases, and at the end of sentences. Period."

speech, and show speech as periods of sound and silence. An early example of such as representation was used in the Intelligent Ear [Sch81]. The Etherphone voice editor [AS86], illustrated in Fig. 2.3, used black and white segments to convey the silence information. Sun's AudioTool uses another sound-and-silence representation, as shown in Fig. 2.1. These forms deemphasize audio details, which are unlikely to be salient to users of speech data, and highlight the temporal aspect of sound. As will be discussed shortly, silences are not reliable indications of the structure of the speech's content, but do provide structure for display and graphical interaction.

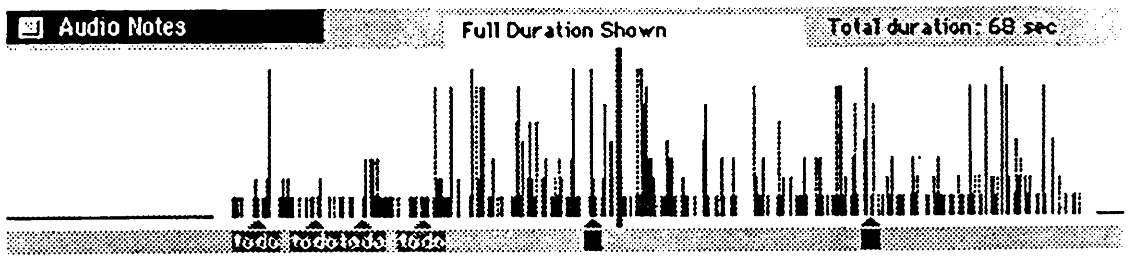


Figure 2.2: Apple's SoundBrowser representation, from [DMS92]

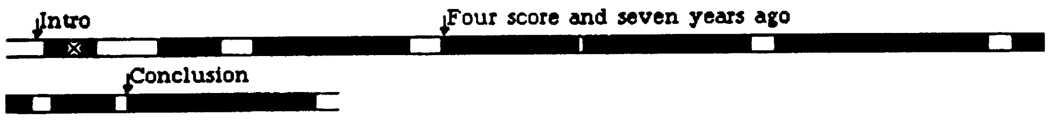


Figure 2.3: Etherphone's sound-and-silence representation, from [AS86]

2.1.3 The Structure of Telephone Conversations

In order to design tools for capturing, displaying and retrieving structure in an audio interaction, that interaction must be well understood. This section looks at research on telephone and audio communication for details on the structure inherent in telephone calls.

One useful parameter is the expected duration of business telephone calls. Three large-scale field studies of patterns of business contacts were conducted in Sweden and London, 1968-71, and showed that telephone calls were brief, with 82–87% of the calls lasting between 2 and 10 minutes (summarized by Thorngren [Tho77]). Reder and Schwab's recent study of professional work activities within a large company showed similar results [RS90]; calls lasted an average of 3–6 minutes.

A number of studies quantify other parameters, such as turntaking, pausing, and interruptions. These studies were designed to compare face-to-face conversations to audio-only interactions, as summarized by Rutter [Rut87], but their results are generally applicable to audio interaction. Butterworth, Hine and Brady [BHB77] looked specifically at grammatical, hesitation and turntaking unfilled pauses in three conditions: visual communication, audio-only (separated by a screen), and telephonic. Their study asked undergraduate subjects to converse in pairs about ethical situations, and 200-second long samples of the conversation were analyzed in depth.

The results for the telephone condition indicated that 30% of the total conversation time was taken up with pauses, that pauses during an utterance accounted for 20% of the utterance time, and that short utterances (such as attention signals, paraverbals, clarification questions like "Pardon?", laughter, and attempted interruptions) occurred 5 times per 200-second sample. Within-utterance pauses were approximately .7 seconds long, and occurred 26 times in 100 seconds.[1]

Another study, by Rutter and Stephenson [RS77], compared audio-only to face-to-face

[1]Because only measures that differed statistically were given, the durations and frequency of turntaking pauses are not available, and cannot be compared to these untypically long pause times.

conversations, and provides additional measures of audio communication. The subjects were students who had not previously met, and they spoke in pairs for 15 minutes about political matters, either sitting side-by-side, or, in the audio-only condition, conversing over intercoms. In this laboratory experiment, turns were observed an average of 25 times per 5 minutes; participants spoke at approximately 3 words per second, and filled pauses occurred about 3 times per 100 words, with men using more filled pauses than women. Utterances averaged 34 words long, with 28% of those shorter than 5 words, and an additional 15% between 5 and 10 words in length.

Two detailed analyses of actual telephone calls provide more robust parameters for designing segmentation tools. One study of the Watergate tapes showed that utterances averaged 15 words in length, and that approximately 10% of utterances were (presumably brief) acknowledgments [WW77]. Beattie and Barnard [BB79] focused on turntaking during inquiries to directory assistance, and drew upon 18 calls selected from a a corpus of 700 British telephone operator interactions with cusomters. They found that turntaking pauses averaged .5 seconds long, although 34% of turns were accomplished within .2 seconds. Filled pauses occurred about .6 times as often as unfilled pauses. Their results confirmed earlier work in that instances of simultaneous speech are less common in phone calls than in face-to-face conversations. Interruptions are less frequent in calls, but pauses are shorter and filled pauses are more common.

The above studies suggest that pausing in and of itself is insufficient for structuring a conversation, because turntaking pauses will not be distinguishable from other pauses by their length, not all pauses will be attributable to turntaking, and many turns happen with minimal pausing. Futhermore, by extrapolation, a telephone call consists of many short segments, interspersed with pauses, filled pauses and short utterances, that do not necessarily reflect the conversational structure very well.

2.1.4 Computers and Telephony

The appeal of telephone conversations as a candidate for everyday use of stored voice is bolstered by the variety of research projects supporting telephone management from computer workstations. Examples include Etherphone [ZTS88] from Xerox PARC, MICE from Bellcore [HORW87], and more recently, PX from Bell-Northern Research [KEE90] and Phonetool from MIT and USC-ISI [SC89]. Such applications can allow users to place calls from online address books; capture calling party information to assist with replying to voice mail messages [Sti91]; log telephone activity; and route calls based who is calling, the time of day and so on [Won91]. As a result, an increasing number of telephone conversations will be mediated by workstations, making it simple to associate situational information with a digitized telephone conversation.

2.1.5 Previous Work in Semi-Structured Media

Semi-structured media have not, to date, been extensively explored. In the text domain, Information Lens [MGL$^+$87] is an example of the power of a semi-structured approach. Information Lens users add extra fields to electronic mail messages; this extra information provides attributes to messages but is not part of the message body itself. Users can then write rules to route and sort received mail, based on these additional attributes [MMC$^+$89].

In the video domain, the Interactive Cinema Group at the Media Lab has been researching a variety of approaches to derived and annotational structure with respect to video clips [MD89, DASP91]. Fujikawa, et al., [FSM$^+$91] automatically derives structure from unedited video segments by identifying differences between scenes, and motion of either the camera or subject within a frame. This information is used to access video segments during editing.

Europarc has been exploring semi-structured ubiquitous computing with a system that continually videotapes lab members; situational information is used for retrieval of the stored voice and video [LN91]. Laboratory members wear "active badges" [Wei91] that can be tracked, allowing a recorded conversation to be correlated with the names of the conversants.

An early example of semi-structured audio was Phone Slave [SA85]; it used a conversational technique to take a telephone message, asking callers a series of questions and recording answers to each question in separate files. As the callers usually cooperated with the automated receptionist, sound segments could be highly correlated with structured information about the call. For example, the caller's response to "At what number can you be reached" usually contained the phone number in a short voice segment. Recently, structured data capture has been applied to a voice-driven form-filling application by Resnick and Virzi [RV92], who use the form-entry metaphor for a telephone-based voice bulletin board. In addition to recording their messages, contributors to the bulletin board are asked to supply information for additional fields, using appropriate mechanisms. For instance, the headline field is filled in with a brief recording, and expiration dates are given by touch tones.

The most closely related work is by Degen, et al., at Apple Computer, and has focused on user-supplied structuring of personal recordings. A handheld tape recorder was modified with two buttons, so that users could mark the beginning of interesting portions of a lengthy recording, such as a meeting, or separate shorter segments, such as items on a to-do list. A key point is that these annotations could be made in real time, as the sound was being recorded. The recordings were transferred to a computer-based application for review, and considerable effort was put into understanding users' needs and then providing a rich set of interactions [DMS92].

2.2 A Demonstrational Approach to Semi-Structured Audio

The project described in this thesis draws upon the work discussed above, and then extends it to two speakers and live conversation. There has not been, so far as I could ascertain, previous work that attempts to capture and display conversational structure in real time, nor has user annotation been explored in the context of dynamic conversation.

Semi-structured audio provides a framework for organizing and manipulating audio data. Information inherent in the speech signal and in the audio interaction can be combined with situational constraints and user annotations to create semi-structured audio data. Semi-structuredness provides enough structure for flexible access to the data later, without relying on the explicit creation of structure by the source or recipient of the audio.

The real-time nature of listening tools is critical—a user is not forced to decide ahead of time whether an utterance should be saved, and not forced to rehear the whole conversation in order to save utterances. Users *can* create structure as they see fit, but they do not *have* to create structure in order for the audio data to be manageable and accessible.

The approach taken in this research is to build working software tools that demonstrate how semi-structured audio can be applied to telephone conversations. The Listener incorporates robust underlying browsing and segmenting capabilities. A graphical sound-and-text display mechanism, the ChatViewer, provides grouping and manipulation of recorded speech segments during the phone call and afterwards. The Browser also allows fragments of recorded audio to be easily discarded or regrouped.

The other underlying capability, segmentation, is the automatic breaking up of speech—while it is being recorded—into small, meaningful pieces, so that it is manageable, and so that only segments of interest are retained. If it were feasible, speech recognition and discourse analysis techniques could convert an utterance to text and then automatically mark the text as being interesting or not. Doing these kinds of processing reliably and in real-time is not feasible for, perhaps, decades [Zue91].

Telephone conversations are a practical choice for demonstrating semi-structured audio. Very little equipment is required beyond audio-capable workstations, and it is possible to detect who is speaking during a phone call. Also, audio is the only channel for telephone communication; facial expressions and gestures go unseen. The saturation of the audio channel in this fashion means that capturing the audio is capturing the interaction. Another advantage of telephone calls is that the participants cannot see each other, so a user can operate a pointing device with minimal interference on the conversational flow. Furthermore, users can be assumed to be experienced and skilled in conducting telephone conversations, so training is not an issue; users will readily perceive the impact of the listening tool on their calls; and the context is understandable and immediate.

A working implementation is a "proof-of-concept". Thus, I can speak directly to the question of whether these concepts are currently workable, and what the limiting factors are in their implementation. Also, a live demonstration is compelling in ways that even a clever simulation is not, and videotaped demonstrations can be shown to far-flung audiences. Finally, having usuable tools is a necessary condition for conducting observational studies. Such studies are especially valuable for new technologies; see [SAH90, SHAM90] for an example of such a study by Schmandt, et al.

Chapter 3

The Capture and Display of Telephone Conversations

To demonstrate the capture and display of telephone conversations, this project concentrated on building the Listener, a telephone listening tool, and the Browser, which displays stored conversations. A *listening tool* is designed for a specific type of audio interaction, and provides an appropriate representation of the audio along with user interaction mechanisms. This chapter describes the design, implementation and behavior of the Listener and Browser.

3.1 System Overview

The Listener and the Browser are both X Window System client applications that communicate with several server processes, and both are built on top of a generic sound-and-text handling mechanism, called the ChatViewer.

Two microphones collect audio signals for the Listener. Ideally, digitized sound could be obtained directly from the ISDN phones; this capability will be available on workstations in the near future. For the time being, to get digitized sound from an ISDN telephone set into a Sun Sparcstation workstation's audio input, a separate device taps into the audio signals carried through the handset cord.[1] For detecting turntaking, that is, changes in which person is talking, an additional microphone is used. This microphone sits near the user and transmits audio only from the vicinity of the user's workstation. The data from both microphones are used by the Listener although, because the Sparcstation is

[1]This device is available from Skutch Electronics, Roseville, California.

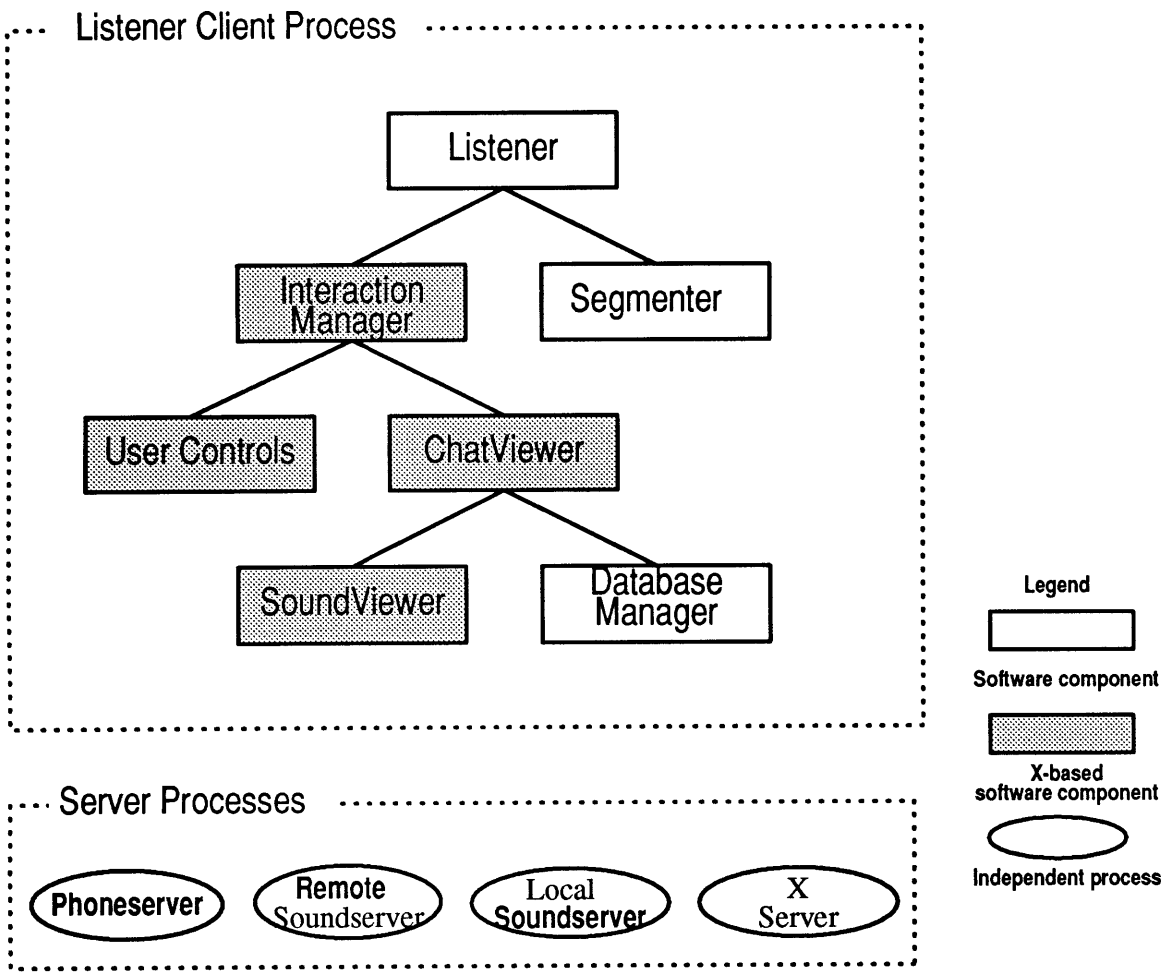


Figure 3.1: Listener Software Architecture

currently limited to monoaural audio input, this second microphone is actually connected to a second workstation.

The Listener builds a representation of the conversation, called a *chat*, and saves that in a database format or, in an abbreviated form, in a *chatfile*. It also saves the audio in a sound file. The Browser then uses these files to build its presentation of the chat, and can save whatever changes the user makes to the chat.

3.1.1 Software Components

Fig. 3.1 shows the components of the Listener. As shown in the diagram, the Listener relies upon independent servers to handle its interactions with various devices, including the user's display, telephone line and workstation audio. Each of the components and servers is briefly described below.

Listener software components:

- The **Segmenter** is the module that performs *segmentation*, that is, breaks the speech signal up into meaningful chunks. Because there are two speakers, segmentation is a significant issue and requires several levels of analysis (see Chapter 4).

- The **Interaction manager** is a logical component that coordinates the other display-related components. It is logical in the sense that, as is typical of an X client application, this coordination is scattered throughout the application code. The X Window System handles the lower levels of user interaction and displays.

- **User controls** refers to the portions of the Listener's user interface that the user can interact with, such as menus.

- The **ChatViewer** is an X Window System widget (a widget is an object that uses the X toolkit) that displays and keeps track of the sound segments generated by the Segmenter. It makes use of two other components, the SoundViewer and the database manager.

- The **SoundViewer** is another X Window System widget. It displays a sound file according to its length and allows a user to select and play any portion of the sound.

- The **database manager** is one of the Speech Group's software libraries, and provides applications with a simple record-oriented database, where fields are defined as key-value pairs.

Independent processes:

- The **Phoneserver** monitors the status of the Speech Group's ISDN telephone lines and can deliver phone-related events to other programs such as the Listener. The Phoneserver itself resides on a Sun 386i workstation; the ISDN phone lines are plugged into a Teleos card installed on an IBM PC. See [Won91] for a more complete description of the Phoneserver's operation and implementation.

- A **Soundserver** interfaces with the Sparcstation's built-in audio capabilities and allows applications to asynchronously record and play audio. Digitized sound is stored in sound files on a large-capacity disk within the Speech Group's local network.

  Two Soundservers are used by the Listener. The primary Soundserver runs on the user's workstation and records the conversations into a sound file. It also reports energy events from the telephone microphone to the Segmenter component of the

Listener application. The secondary Soundserver is used to report energy events from the other microphone that hears only the local user's speech.

- As an X Window System application, the Listener relies upon the workstation's **X server** to display its user interface and pass along user input events.

The Browser is similar and simpler; it does not need a Segmenter, or to communicate with the Phoneserver or remote Soundserver processes. Later chapters describe the Segmenter and ChatViewer in considerably more detail.

3.1.2 Software Environment

The Listener and Browser were developed in the C language on a Sun Sparcstation 2, under the SunOS 4.1.1 operating system. Release 4 of the X Window System toolkit was used, along with the Athena (Xaw) widget set. The SoundViewer, Phoneserver, Soundserver, and database manager components are part of the Speech Group software library.

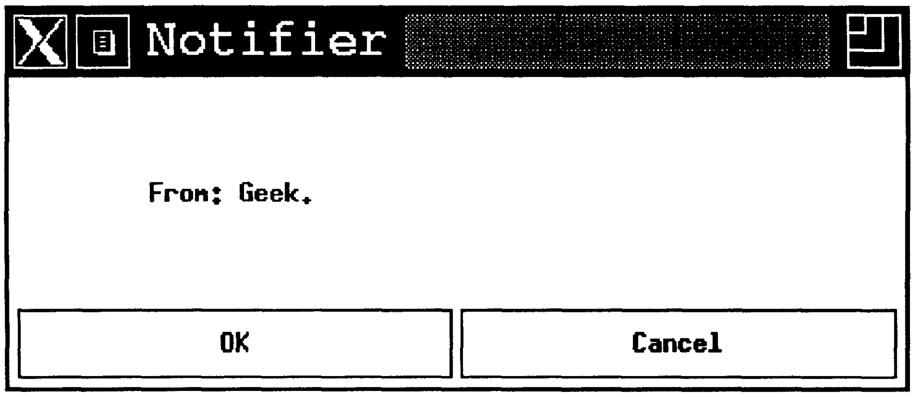
3.2 The Listener Tool

The Listener, the telephone listening tool, captures structure from telephone calls. Listening tools pose challenging user interaction issues. One issue is human cognitive constraints, such as short-term memory limitations and loss of comprehension while performing multiple tasks. Another challenge is how to visually present a new kind of dynamic, real-time phenomenon so that the display is readily understandable to the user. These challenges are discussed in the context of Listener capabilities that are available before, during and after a telephone conversation.

3.2.1 The Listener is Ubiquitous

Part of the conceptual framework underlying this project is that technology should be ubiquitous and integrated into routine activities. In the case of the Listener, placing or receiving a phone call should be sufficient to invoke it. This, indeed, is the case. Through the capabilities of the Phoneserver, the Listener is able to respond to changes in the status of a telephone line.

When, for example, the telephone rings, the Listener pops up a notification window in the middle of the user's display screen, as shown in Fig. 3.2. This notification provides whatever information it can; it makes use of a local extension-based database, so that



From: Geek.

Figure 3.2: Receiving a phone call

calls from members of the Speech Group appear with the name of the calling extension's owner (who is most likely the person placing the call.) For calls from within MIT's ISDN network but outside of the Speech Group, the extension of the calling party is displayed.

The notification window gives users the option to record the call; calls are not recorded unless the user explicitly requests recording. In fact, the window will disappear after a short period (currently five seconds), so no action on the part of the user is necessary if the call is not being recorded.

Once recording is selected, by pressing the $\boxed{\text{OK}}$ button, the call display is started up as shown in Fig. 3.3.

The Listener responds to additional phone line status conditions with the following actions:

- When the called party answers the placed call, the phone line status becomes ACTIVE, and if recording has been selected from the Notifier window, the recording begins at this point.
- When a call ends, the line status is IDLE and recording ends.
- When the handset is picked up and the user hears a dial tone, the phone line status becomes DIALING, and a Notifier window appears.

The Phoneserver can also detect when on-going calls are put on hold or transferred, though the Listener does not currently respond to those events.

The Listener is intended to run continuously on the user's workstation.

3.2.2 The Dynamic Display of Conversational Structure

While in a phone call, a person's attention is focused on the conversation, and cannot be distracted by an obtrusive listening tool. A listening tool that is not interactive would satisfy that constraint. However, the limited timespan of short-term memory mitigates against a non-interactive design; to the extent that interesting segments can be identified at all during a conversation, that identification must happen shortly after the segment is heard.

Numerous factors influence what is kept in short-term memory and for how long, but in very general terms short-term verbal memory can hold between four and seven items, for 2–30 seconds [Pot90, MN86]. Therefore, a visual representation of the previous 30 seconds or so of the call is displayed during the conversation, and a user can select segments of interest simply by clicking on them.

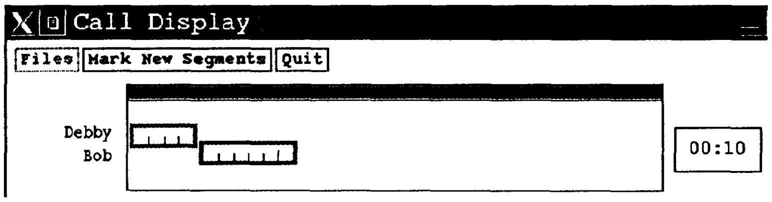
As the conversation proceeds, a visual representation is displayed retrospectively. This representation shows each conversational turn, reflecting the phrase-level utterances of the speakers. Each rectangular object is a representation of a segment, or portion of the audio signal, and each tick mark within a segment represents one second of audio. The Listener determines segment boundaries by analyzing the audio signal; the next chapter describes this segmentation process in detail.

Figure 3.3 shows the call display during part of the conversation. The transcription for each new segment is given below each picture of the call display window. Note that in the third picture, the $\boxed{\text{Mark New Segments}}$ button is highlighted. All the segments from then on, concerning the article in *Computational Linguistics*, were automatically visually highlighted on the screen, although the darker border color may not be distinguishable on paper.

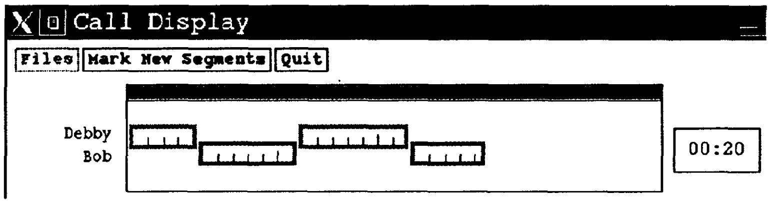
New segments appear at the right-hand side, and older segments scroll out of view to the left. Speakers are visually distinguishable by their vertical offsets, and by different border colors when unmarked. To draw the user's eyes to where new segments appear, there is an elapsed time indicator just to the right of the segment display. It is updated about once per second.

3.2.3 Identification of Interesting Segments

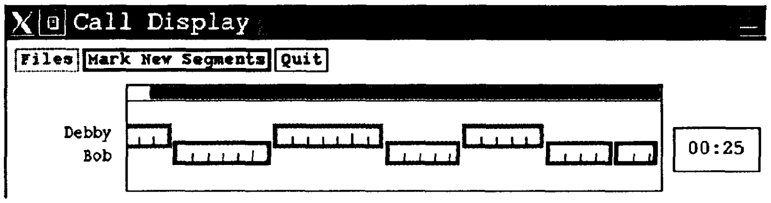
A central concept of semi-structured audio is that structure enables users to make better use of audio. In the case of telephone conversations, the conversational structure makes it possible to identify the segments containing information that ought to be saved for later



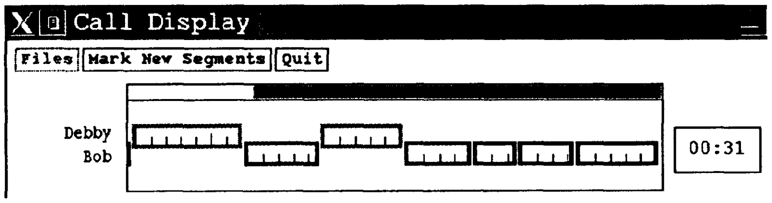
Debby: Hello, this is Debby Hindus speaking.
Bob: Hi Deb, it's Bob. I'm just getting out of work, I figured I'd call and see how late you're going to stay tonight.



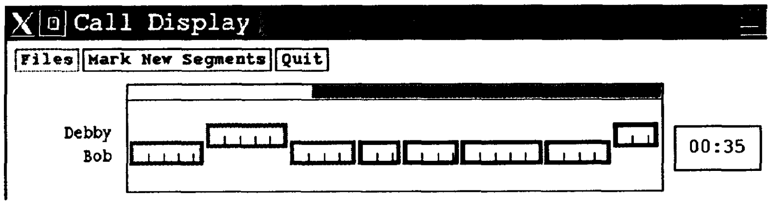
Debby: Well, I think it'll take me about another hour, hour and a half, to finish up the things I'm doing now.
Bob: OK, I'm just going to head on home, I'll probably do a little shopping on the way.



Debby: Well, if you think of it, maybe you could get some of that good ice cream that you got last week.
Bob: OK. By the way, somebody, uh...
Bob: mentioned an article you might be able to use



Bob: in your tutorial. Debby: Oh really? [Debby's very short turn is ignored.]
Bob: Yeah, it's by Graeme Hirst, in the June '91 Computational Linguistics.



Bob: It's a review of, uh, Computers and Conversation.
Debby: Oh, that sounds pretty interesting.

Figure 3.3: Sequence of segments during a phone call, with transcriptions

retrieval. The Listener provides mechanisms for *marking* these interesting segments. Marking a segment causes the segment to be visually highlighted, as shown in Fig. 3.4. Marking also determines whether a segment is saved or not.

Segments can be marked in several different ways so that marking can take place whenever the user realizes that the conversation has become worth saving. A phone call often begins with a social exchange that is probably not worth saving. Later, the conversation will turn to more substantial matters. At this point, the user may decide to invoke the primary marking mechanism, which marks all new segments until the user indicates otherwise. The $\boxed{\text{Mark New Segments}}$ button controls whether this mechanism is active or not.

Interesting statements may also occur unexpectedly during a conversation, when automatic marking is not in effect. The call display includes the just-completed utterances and recent history of the conversation, because the significance of an utterance may not be realized right away. At any time, a user can mark individual segments by clicking on them. Marking is a reversible state; marked segments can be unmarked simply by clicking on them. Therefore, uninteresting segments that occur during an otherwise interesting part of the call can be easily unmarked.

3.2.4 Post-Call Review and Revision

The Listener is notified by the Phoneserver when the telephone call has been terminated by either party. Recording is automatically terminated at that point. The user also can press the $\boxed{\text{Quit}}$ button at any time during the conversation.

The nature of the user interaction changes once the conversation is completed, from capture to review. Functions that were too obtrusive during a conversation are now made available, and the Listener essentially turns into a Browser.

The complete conversation is available for review at this point, as shown in Fig. 3.4. A user can replay all or part of the conversation, revise the choice of segments to store and save the chat for later retrieval. When the user finishes reviewing the chat and exits by pressing $\boxed{\text{Quit}}$, the portions of the recorded audio that correspond to marked segments are stored permanently and the rest of the audio file is discarded.

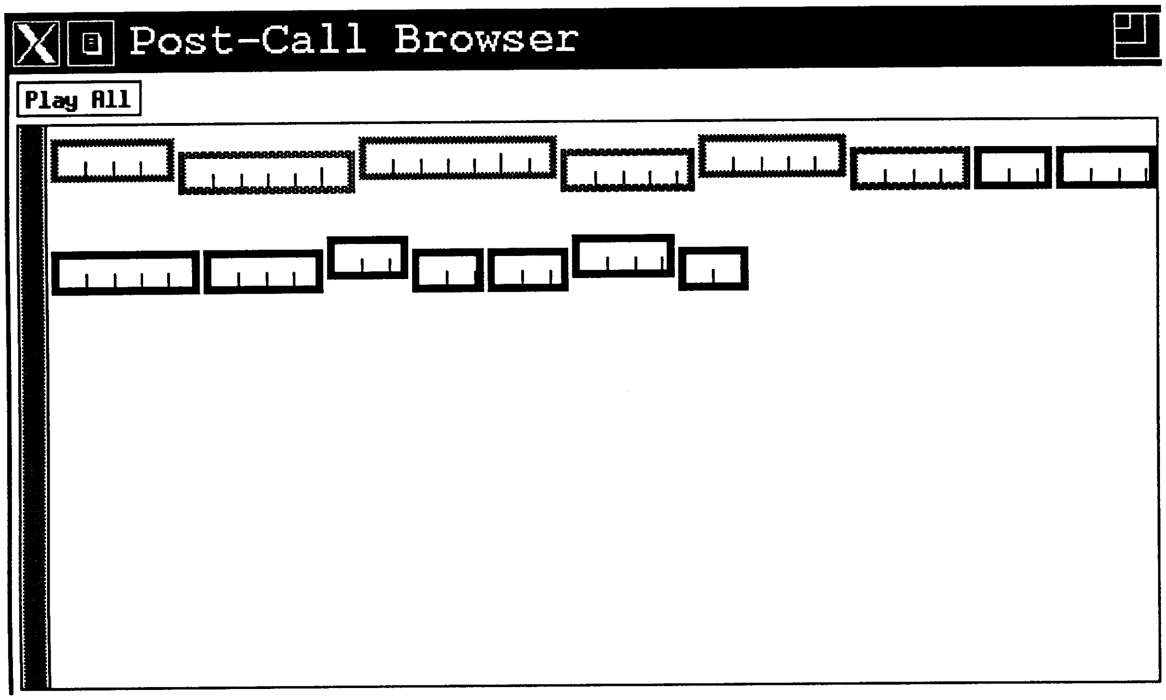


Figure 3.4: Browsing after a phone call

3.3 The Browser Tool

The Browser graphically displays *chats*, collections of stored sound segments and text. Within the Browser, the ChatViewer widget provides underlying display and sound handling functions. The Listener uses the ChatViewer to provide a simple browsing capability; the Browser application is intended to provide a richer set of chat editing functions, though few of those functions are currently implemented.

3.3.1 Multiple Views

The Listener's call display window is designed to show only the last 30 seconds of the conversation. Earlier portions can be seen by scrolling within the call display window, but that is an unwieldy mechanism, particularly for a lengthy conversation. To provide an overview of the conversation, the Browser creates a *miniature chat* along with an enlarged display of the same representation, as shown in Fig. 3.5.

The two views both display the conversation much like a sheet of music displays a tune, so that segments are "wrapped around" to another line when the window boundary is reached. Scroll bars provide the capability to scroll around in the enlarged display.

In the very near future, the miniature will be hooked into a Listener-style closeup view of

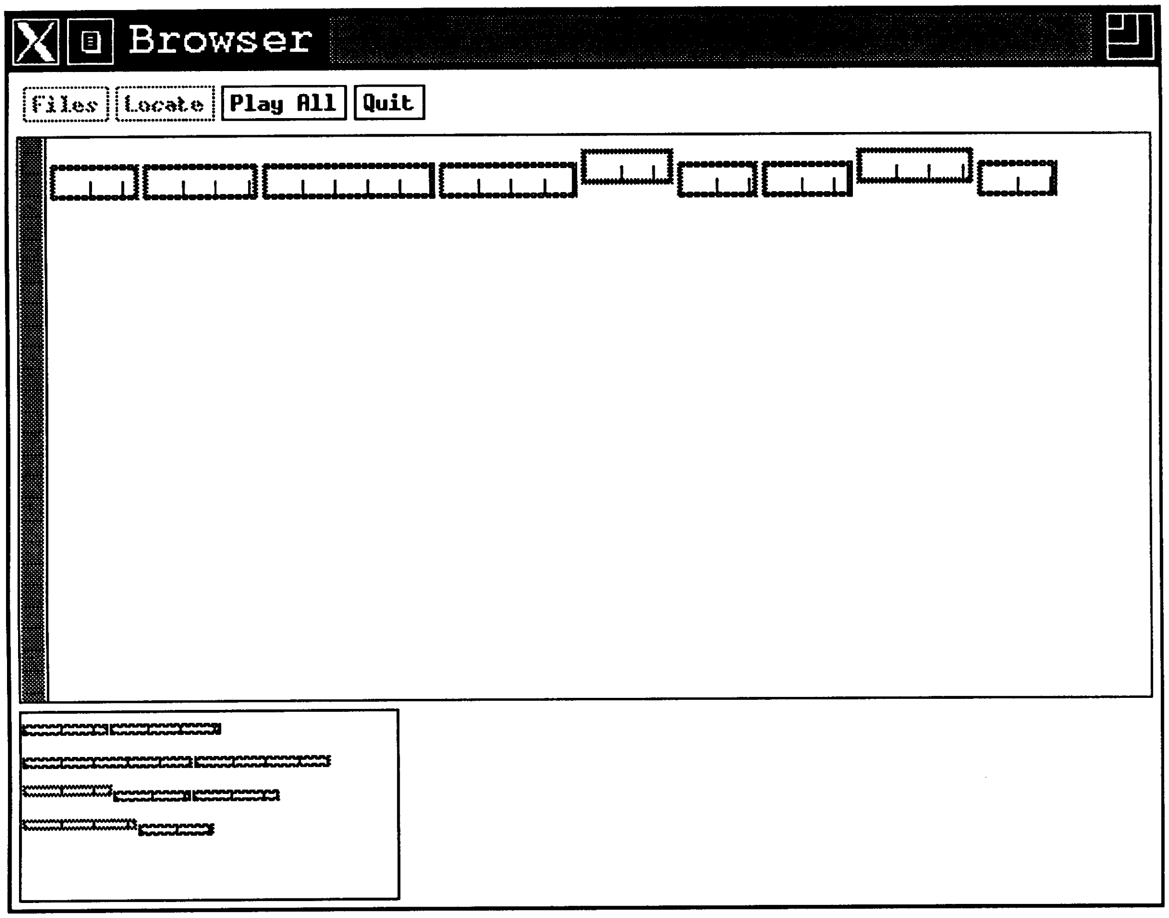


Figure 3.5: The Browser, displaying the saved portions of the sample conversation

the chat, so that the user can navigate through the complete conversation with a panner, and the portion under the panner will be expanded into detail in the closeup view.[2]

3.3.2 Replaying Segments

Segments are displayed as SoundViewer widgets, which provide users with considerable control over the replay of a segment. SoundViewers allow a sound to be started, stopped and restarted at any point, with just a press of a mouse button, and the replay status is indicated by an indicator bar that fills in as the replay progresses. Fig. 3.6 shows a SoundViewer playing a segment.

So, it is easy to replay just a portion of a segment. Segments can be played one at a time. The complete conversation can also be played continously, using the $\boxed{\text{Play All}}$

[2]In the R5 version of the X Window System, the Athena widget set includes the Panner and Porthole widgets, which are designed to implement exactly this kind of panning operation.

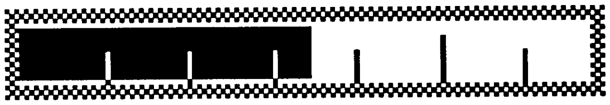


Figure 3.6: A closeup of a SoundViewer, with a playback in progress

button. These aggregations of segments are replayed using the individual SoundViewers, so any replay can be interrupted or restarted at any point.

3.3.3 Editing Stored Conversations

The Browser is the primary interface for manipulating stored conversations. However, the format of the stored conversations is ASCII and therefore editable; sound segments are represented as pointers to subsections of sound files.

Chapter 4

Turntaking and Segmentation

Monolithic, lengthy recorded audio does not allow users to identify and mark the interesting utterances. In order for the user to take advantage of a conversation's inherent structure to filter out those worthwhile segments, the Listener needs to derive the inherent structure from the audio signal during the conversation. This chapter discusses the workstation's software and hardware platform for audio, describes how the Listener breaks the audio signal into meaningful pieces, and addresses some of the related issues in segmentation.

4.1 Workstation Audio

Real-time segmentation relies upon on the workstation's underlying hardware and software support for audio. This section characterizes the workstation's audio capabilities and the Speech Group's audio software library and asynchronous audio server.

4.1.1 Workstation Capabilities

The audio capabilities of the workstation used in this research are well-suited to handling telephonic speech. The Sun Sparcstation 2 comes with hardware for digitizing a single sound channel.[1] The sampling rate is 8000 samples per second, which, according to Nyquist's Theorem, corresponds to a 4000-Hertz bandwidth. Each sample contains 12 bits, though it is stored in a compressed, 8-bit μ-law format [Par86, Sun90].

[1]The Sparcstation contains an AM79C30A Digital Subscriber Controller chip, supplied by Advanced Micro Devices.

To put these numbers in perspective, consider that humans are capable of hearing sounds up to about 20,000 Hertz, and compact-disk-quality audio is sampled 44,100 times per second with 16-bit samples. However, telephone lines carry audio signals with only a bandwidth of 300Hz–3400Hz [Ron88], and so higher-quality digitizing would be of no particular benefit when working with audio from telephones.

Digitized audio is stored in files, referred to as *soundfiles* or *audio files.* Even in μ-law format, audio requires considerable storage space; a recording that is 1 minute long will take 480K bytes to store.

4.1.2 Workstation Audio Support

The workstation audio device, `/dev/audio`, supports low-level read and write operations. The audio control pseudo-device, `/dev/audioctl`, supports operations on the characteristics of `/dev/audio`, such as setting the volume levels. The Speech Group has built a library of device-independent routines that provide a level of abstraction above these devices, so that application programs need only be concerned with actions on sounds themselves and not with the details of interfacing to the audio devices. This library supports audio operations that are non-blocking, a feature required by interactive applications. The library includes a wide variety of audio-related routines in addition to the basic record and play routines.

The Soundserver is a networked server for managing asynchronous audio operations that is built on top of the sound library routines. It was written by Barry Arons of the Speech Group, and uses two other tools, the ByteStream Manager and the Socket Manager, for handling interprocess communication [Spe91]. The Soundserver's architecture emulates that of the X Window System in that it consists of a server process that executes on the workstation, and a client-side library of routines that are linked into an application program. The client-side routines enable the application to communicate with the server process through callbacks.

In its current implementation, a Soundserver process can interact with only one application at a time. This is normally sufficient, as an application connects to the local Soundserver just before it actually uses the audio device and disconnects immediately afterwards. Applications can, however, connect to multiple Soundservers as long as they run on different workstations, and this is what the Listener does to accommodate two audio input streams in the absence of stereo audio input.

The Soundserver's event-oriented software interface allows for intermixing audio and non-audio events, such as user input. This is important to an X Window System application, because X is entirely event driven. Also, a callback-driven programming style can lead to more modular code.

4.2 How the Listener Determines Segments

The inherent structure of a conversation is defined by the naturally-occuring pauses at the end of phrases and sentences, and the alternation of utterances between speakers. The audio data can be automatically segmented into understandable pieces, based on the boundaries between speakers and between phrases. This division of the audio signal is done by the Segmenter component.

As previously discussed, there are two sources of analog audio signals available to the Segmenter, dispatched through two Soundservers. One audio source comes from the telephone handset and carries speech from both ends of the phone call. The other audio source is a local microphone, placed near the user's workstation, to capture only the local user's speech.

To determine the boundaries of segments, the Segmenter receives audio data from both these sources, performs pause detection on each source, synchronizes the sources, and then locates changes of speaker between pauses. This multi-stage processing is presented in the rest of this section.

Before the Segmenter can receive audio data, it must establish communication with the Soundserver. The initialization involves the following steps:

- The Segmenter connects to the local and remote Soundservers when a phone call begins.
- The Segmenter then registers, with both Soundservers, callbacks for two kinds of sound-related events, *energy* events and *end-of-recording* events.
- Finally, the Segmenter tells the Soundservers the names and maximum lengths of the to-be-recorded soundfiles, and requests that the digitized audio be saved in a file in addition to causing energy events to be generated.

Once recording begins, the Soundservers receive input from the workstations' audio devices. The energy events generated by the Soundservers trigger the registered callback routine.

4.2.1 Acting on Energy Events

Energy events occur at fixed intervals, currently one event every 100 milliseconds, and contain data about the audio signal over that interval. This data consists of the following items:

> which Soundserver generated this event;
> the timestamp for the interval, starting at 0;
> the average and peak magnitude of the signal; and,
> average and peak energy values of the signal.

Note that the actual digitized audio is not part of the data returned by energy events.

The energy event callback routine performs the following functions, as shown in Fig. 4.1:

- First, the latest event, along with previous events from the same Soundserver, are considered to determine the current *mode* of that audio source. The mode refers to whether the audio is currently in a silent period, a non-silent period, or in transition between the two. The section on pause detection, below, describes more fully how the mode is determined.

- Then, the overall state of the conversation is determined—is someone speaking, and if so, who is it. The procedure decides who is speaking by looking at the current modes, speech or nonspeech, of the two sound sources.

  In order for the two modes to be considered together, however, they have to be referring to the same time interval. Events from the two Soundservers arrive intermixed, and may arrive in groups of several events from one Soundserver, followed by a similar group from the other Soundserver. Therefore, the two streams of events have to be synchronized. This is accomplished by keeping two queues, one for each audio stream. These first-in-first-out queues contain modes and their corresponding timestamps. Once the modes are synchronized, the conversational state for that time interval can be ascertained.

- The conversational state for each time interval is then fed into the *turn detector*, which takes the appropriate actions to find the boundaries of segments and then creates the segments. This process is complex enough that it calls for a state transition approach, implemented as a state table, and is described below.

- When a turn is detected, a segment is created and is immediately added to the call display, as described in Chapter 3.

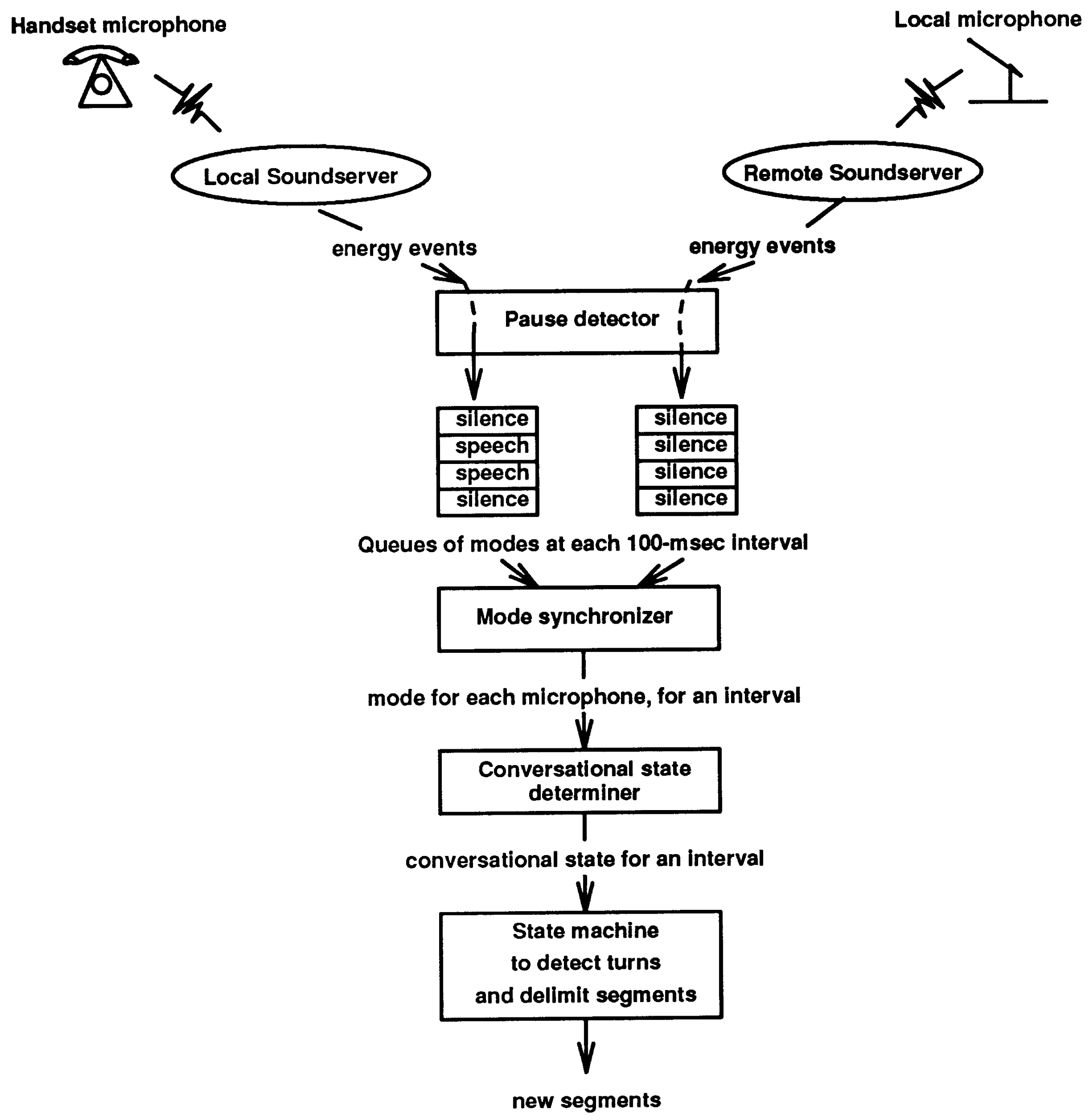

Handset microphone
Local microphone
Local Soundserver
Remote Soundserver
energy events
energy events
Pause detector
silence
speech
speech
silence
silence
silence
silence
silence
Queues of modes at each 100-msec interval
Mode synchronizer
mode for each microphone, for an interval
Conversational state determiner
conversational state for an interval
State machine to detect turns and delimit segments
new segments

Figure 4.1: Processing energy events in the Segmenter

4.2.2 Pause Detection

The preceeding section refers to determining the mode at a particular time. This determination procedure is actually doing *pause detection*, that is, dividing the audio signal into discrete periods of speech and nonspeech, where each piece of speech is separated by a pause, or a brief silent period.

A pause occurs when the magnitude[2] of the audio signal is low enough for a long enough period of time. Similarly, speech is detected when the magnitude is high enough for long enough. A pause detection procedure, then, specifies exactly how the signal is determined to be "low enough", and what "long enough" means.

The parameters that govern pause detection are as follows:

- Minimum duration of silence, in milliseconds
  Minimum duration of speech, in milliseconds

  These durations are currently both set to 300 milliseconds. Note that although durations are specified in milliseconds, the Segmenter needs to know how many energy events to examine, and so the durations are converted into counts of energy events when the Segmenter is initialized.

- Amplitude threshold for speech

  This threshold is specified separately for each microphone, and its value is related to the range of values typically produced by that microphone.

The amplitude of a signal containing speech is quite variable, especially at the beginning or end of an utterance. Even during a word, any single energy event may fall below the amplitude threshold for speech. Therefore, a pause detector cannot decide whether the current mode is speech or nonspeech by looking at only one energy event; it must take into account the amplitude over a period of time, as specified by the minimum duration times.

The Segmenter uses a sliding window when deciding whether the signal is currently speech or nonspeech, as shown in Fig. 4.2.

The pause detector looks at the average amplitude for the latest event, along with the consecutive preceeding events that constitute the mimimum duration, and only declares nonspeech or speech if the amplitudes from all the examined events meet the "low enough" or "high enough" criterion. Otherwise, the pause detector does not determine

[2]The magnitude of a signal at a point in time is just the absolute value of the signal's amplitude.

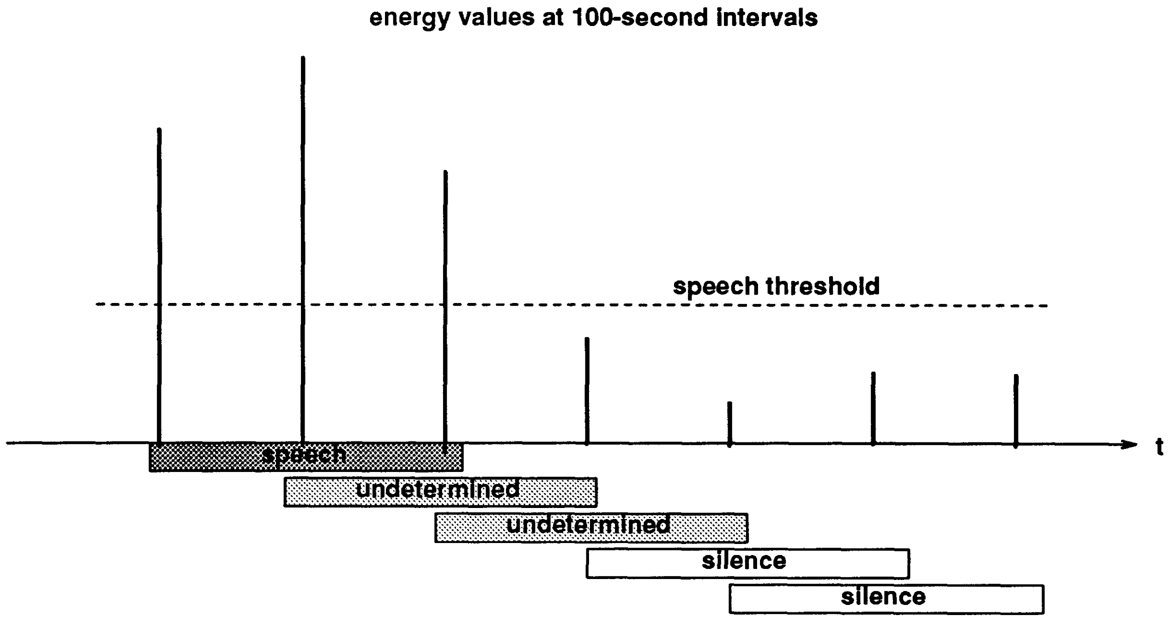


Figure 4.2: Example of a 300-msec sliding window used in pause detection

whether the mode is speech or not. If, for example, only one of the last three intervals is high enough to be speech, the pause detector declares that the signal's mode is undetermined at that time. This procedure is a simplification of the endpoint detection algorithm given in O'Shaughnessy, p. 434 [O'S87].

4.2.3 Turn Detection

Turntaking pauses, that is, the pauses between the end of one speaker's utterance and the beginning of the other speaker's utterance, may be quite short, so short that pause detection would not find them. But knowing who spoke which segment makes the structure of a conversation much more obvious, and so that information needs to be obtained by means other than pause detection.

As previously mentioned, the Listener uses two microphones to determine who is speaking. This speaker-based segmentation can be accomplished by detecting when one speaker's turn begins and ends, and matching this time window to the recorded audio. (The beginning and end times are the boundaries of the segment, and the boundaries are used in displaying and storing segments.)

The procedure in turn detection is to ascertain who is speaking at each interval, note when the speaker changes, and construct segments based on this information. Final segment determination must take into account constraints that ensure segments are as meaningful as possible.

The pause detector, described just previously, decides whether there is currently speech or nonspeech occurring on each of the two audio sources. Then, those modes are combined and the current conversational state is deduced, using the following simple rules:

- When there is speech in the telephone microphone, one or the other of the conversants is speaking. If there is also speech coming through the user's local microphone, then the user is speaking and that segment can be attributed to her.

  Note that the other person may also be speaking at the same time; there is not enough information available for that condition to be distinguished from only the user speaking.

- If there is speech in the telephone line, but the user's local microphone reports silence, then the other person is speaking.

- If both audio sources are silent, then neither person is speaking and the overall state is silence.

This overall state is then input to a *state transition* procedure, wherein the previous state and the new input together determine which transition to make, that is, the action to take and the new state. The state transitions for the turn detector are shown in Fig. 4.3.

The possible actions are:

Start the first segment—has to be special-cased
Start a segment, after a silence
Start a silent period
Switch between speakers, with no intervening silence
End a segment

The possible states are:

Unset—for starting out
Me talking
You talking
Silence

Note that silences are divided equally between segments, as shown in Fig. 4.4. This has attractive side effects. One is that no part of the audio signal is discarded, so the

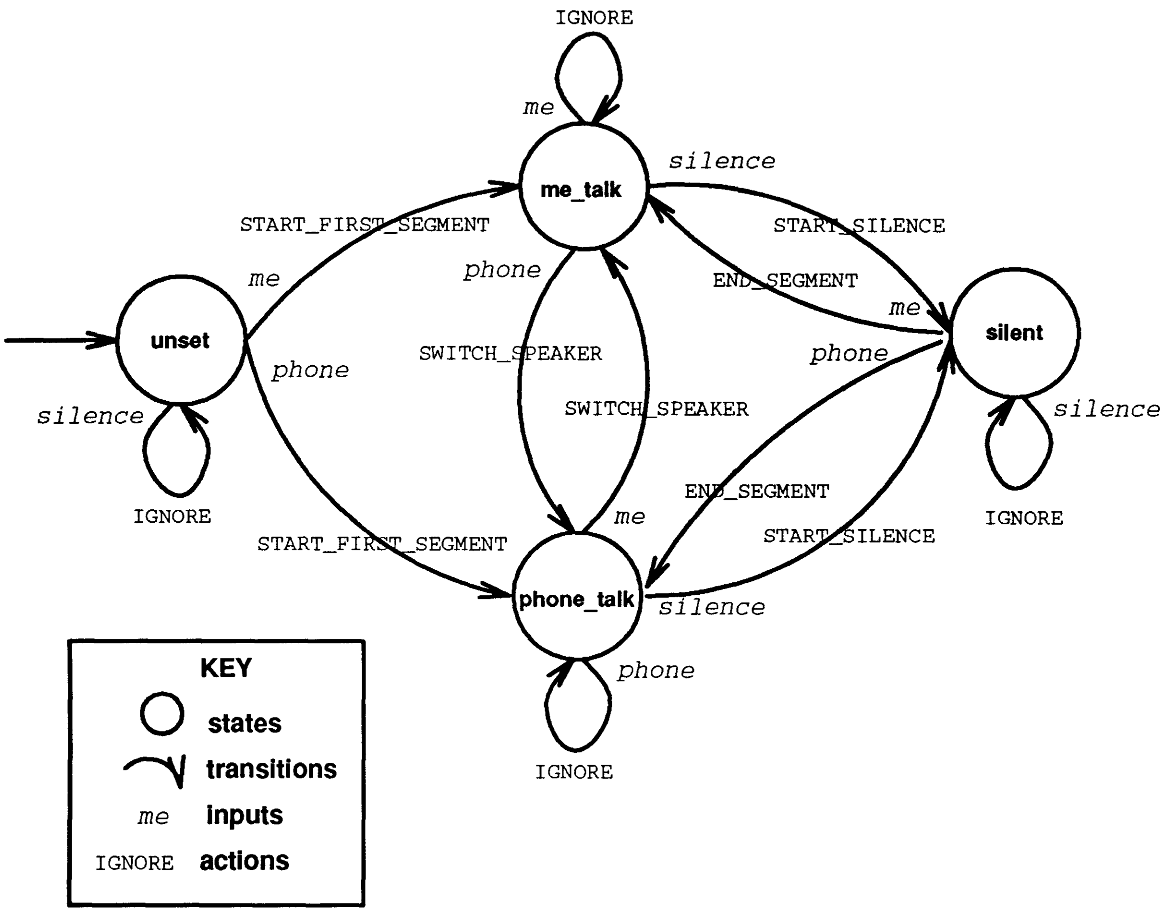


Figure 4.3: State transition diagram for determining segments

beginning and end of every utterance is preserved. During playback, the stored segments sound natural and complete when played individually as well as when played sequentially. Another benefit is that the pause detector does not inadvertently cut off parts of phonemes that have a low overall amplitude and energy in the higher frequencies; this can happen with silibants such as "s" sounds. The low bandwidth of a telephone line would exacerbate this problem in the case of the Listener.

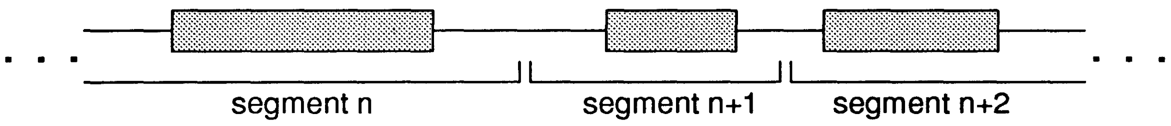


Figure 4.4: Silences are divided between segments

When the recording is terminated, either because the telephone call has ended or because the user has stopped the recording, the final energy events are processed and the last segment is created.

Constraints on Segmentation

As predicted by the studies of telephone conversations described in section 2.1.3, utterances are short, and very short turns (VSTs) are common. These VSTs are attention signals, attempted interruptions, paraverbals and so on, and should not, in general, be treated as a true turn. VSTs should either be treated as part of the current segment, or dropped altogether, or visually indicated but still contained in a larger segment.

Sometimes a single utterance will include a short segment, due to the natural pauses used by speakers in place of commas. Even within an utterance, short segments are not particularly meaningful or useful, and so segments should be at least a minimum length. There is a Segmenter parameter that specifies this minimum segment duration; it is currently set to 2 seconds, based on some experimentation.

The minimum segment duration is not sufficient for handling VSTs properly, however, because the segment interrupted by the VST should be treated as a single segment. Just forcing segments to be a minimum duration will cause an interrupted utterance to be broken into two segments. (It would be three segments if the VST were treated as a separate segment, but that is precluded by the minimum segment duration.)

4.3 Issues in Segmentation

While implementing segmentation and observing how well it worked within the Listener, I came across several issues.

4.3.1 Delays in Updating the Display

The turn detector does not establish the ending point of a segment until after the end of the segment; because silences are divided equally between segments, the end of a segment cannot be known until the next segment has begun. An unfortunate consequence of this technique is that this delays the creation, and therefore the display, of the latest segment. The user does not see the segment until at least .5 second after it has ended.

In practice the delay is closer to 2 seconds, and is quite noticable. As a user, I found it disorienting to not see something happen when I spoke, and the delay caused me to occasionally lose track of which segment corresponded to which utterance. This is a significant drawback for a listening tool, needless to say.

A better approach would be to show a lengthening segment-like indicator while the utterance is being spoken, and then turn it into a real segment when the endpoint has been determined. This approach would be useful in another context as well: when a VST interrupts a segment, the endpoints of the VST and the preceeding segment will need to be revised. This revision will not be as visually unsettling as it might be otherwise, because the user will be familiar with segments and endpoints being mutable.

4.3.2 Pause Detection

The issue here is ascertaining whether the microphones are transmitting useful information or not. The Segmenter does have parameters that specify minimum thresholds for deciding whether audio is coming through each microphone connection, and these thresholds are intended to enable the Segmenter to tell if one of its audio sources is not available and, if so, adapt its processing accordingly.

4.3.3 Synchronization

The Segmenter currently uses two microphones, each connected to a different Soundserver on different workstations. The use of two machines has introduced a synchronization issue: does the event from one audio source at time *n* really correspond to the event from the other source that is also labeled time *n*? Or, does it correspond to time *n - 1*, or time *n + 1*? In other words, is there a *skew* between the data from two audio sources? Also, the skew might not be consistent throughout a phone call; does the data from one of the workstations *drift* in relation to the other?

Skew and drift greatly affect how well the Segmenter works, in part because the size of the interval between audio events, 100 milliseconds, is large enough that even a one or two-event drift can cause a pause or turn to be missed. Such mistakes cause further errors; a state transition procedure is strongly influenced by one mistaken transition, and in practice several succeeding segments will be mis-segmented. Also, users are aware of their own speech, and quickly notice if the segmentation errs, for example, by erroneously attributing their utterances to the other speaker.

So, I added an explicit synchronization step at the beginning to detect the skew. When the user is placing the phone call, the initial synchronization works by asking the user to say the word "Go", after dialing a digit to eliminate the dial tone. (When the user is receiving a phone call, the user is extremely likely to speak first, and the synchronization could use that initial utterance in place of the word "Go".) The word "Go" works well for these purposes; it has a distinct beginning point and is just long enough, 300-400 milliseconds. The synchronization procedure looks over the audio events from the two microphones, and compares the starting point and lengths of the speech segment to determine if there is a skew.

The Segmenter can take the skew, if any, into account when synchronizing events. If it seemed warranted, another synchronization at the end of a phone call could verify that there was no drift during the conversation. However, since I added the explicit synchronization the skew has almost always been zero!

Chapter 5

The ChatViewer Widget

This thesis is premised upon the idea that speech is ever-present in everyday life, and computers can capture that speech for later reference. However, the nature of conversation suggests that these recordings may be lengthy; soundfiles many minutes long will not be unusual. This raises the issue of how to make these long recordings usable. Part of the answer is structuring the audio data; this topic was the subject of Chapter 4. Another part is displaying the structured audio in an understandable way, and providing users with the requisite level of control and interactivity with the audio. What kinds of software abstractions are needed to support interaction with large amounts of audio?

The ChatViewer is an attempt at such an abstraction. It is designed to manage and display *chats*: collections of recorded speech segments and short text strings. Using the ChatViewer, applications can provide features to create new chats, display chats in several different ways, review and edit chats, and store chats in two kinds of files, *chatfiles* and database files. ChatViewer widgets maintain information about the chat's sound and text *items* in a simple database. ChatViewers also support a range of user interactions.

This chapter presents the capabilities of the ChatViewer and briefly describes its implementation and interfaces.

5.1 The Nature of Widgets

The ChatViewer is implemented as a *widget* and is built using the X Window System, Xt Toolkit, and Athena widget set supplied by MIT and the X Consortium[1]. A widget is a user interaction component. It is also an abstract data type, and an object within a single-inheritance class hierarchy abstraction [AS90]. Widgets are defined by their attributes, or *resources*, and their operations, or *methods*. Resources are settable by users and by application programs.

Writing a non-trivial widget is not a task to be undertaken lightly. Contrasted with application programming, it requires a much deeper understanding of Toolkit concepts and implementation particulars. Furthermore, widget writing is somewhat of an arcane art, and there is not much documentation available about widget intricacies.

However, widgets are powerful tools for managing user abstractions, especially abstractions for which the details of appearance and arrangement are critical. Both the Listener and the Browser need to display multiple sound segments, though their displays serve very different purposes and look quite different. The ChatViewer, as a widget, can handle these different display needs and hide the details from the applications. Because widgets are objects, they can be reused by different applications, and there can be multiple ChatViewers within a single application, each with different attributes.

One major motivation behind building the ChatViewer as a widget is that a widget can readily be employed by other applications. The ChatViewer has been used in the Listener and Browser, of course, and also in the `todo` application, which supports lists of sound and text items[2]. The ChatViewer can also be adapted for use in transcription tools, such as manual speech-to-text transcriptions[3].

5.2 Using a ChatViewer within an Application

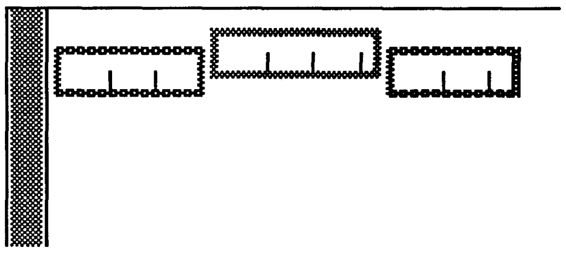
Within an application, a ChatViewer widget is created and used like any other widget. There are a number of resources defined for ChatViewers; the most important ones, the XtNchatName resource and the XtNlayoutStyle resource, are discussed here. The remaining resources are listed in Appendix B.

[1]For more information about these software libraries and the X Window System in general, see the excellent book by Asente & Swick [AS90] or the documentation supplied by the Consortium [MIT88].

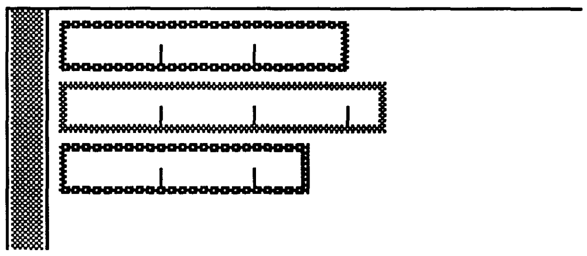
[2]The `todo` application was developed by Chris Horner of the Speech Group.

[3]Using ChatViewers for transcriptions was suggested by Barry Arons.

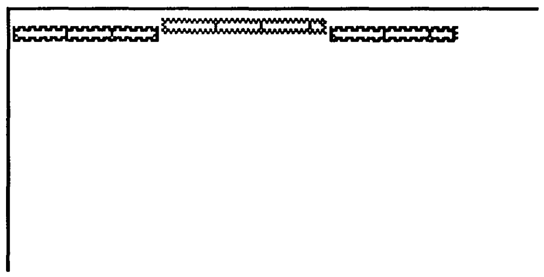
The XtNchatName resource is the name of the input datafile, either in chatfile or database format. The XtNlayoutStyle resource specifies how the ChatViewer visually presents the chat items in its display. The available layout styles are: XawlayoutStyleLeft, XawlayoutStyleTime, and XawlayoutStylePan, as shown in Fig. 5.1.



XawLayoutStyleTime



XawlayoutStyleLeft



XawlayoutStylePan

Figure 5.1: Examples of ChatViewer layout styles

The XawlayoutStyleLeft style places each item on its own line; it is well-suited for displaying lists of unrelated items, such as a list of things to do. The XawlayoutStyleTime style is used by the Listener, and treats sound segments as related and in an infinitely long horizontal display. The XawlayoutStylePan style creates the miniaturized displays of conversations that are used in the Browser.

The ChatViewer includes public routines that provide the following functions to the application:

- Dynamically modify chats, by adding and deleting items.

These functions are used by the Listener, which creates a new sound item for each segment that is detected by the Segmenter (see Chapter 4 for details).

> void XawCVAddItem (*widget, itemid, item_info*);
> void XawCVDeleteItem (*widget, itemid*)
> void XawCVMoveItem (*widget, from_itemid, to_itemid*)
> void XawCVAddItemAtEnd (*widget, item_info*)

- Mark and unmark items, both visually and in the database, and enable or disable marking.

  These operations are required by the Listener, so that users can identify to the program (and to themselves) the portions of the telephone conversation worth saving.

  > void XawCVMarkItemInDB (*widget, itemid*)
  > void XawCVUnmarkItemInDB (*widget, itemid*)
  > void XawCVVisualMarkItem (*widget, itemid*)
  > void XawCVVisualUnmarkItem (*widget, itemid*)

- Provide relative access to next and previous items; provide direct access to items and widgets, with item-to-widget conversions and *vice versa*.

  Although the ChatViewer itself provides the most commonly-needed functions, an application may wish to extend these. Therefore, the widget cooperates by giving programs access to the items at the lowest levels.

  > Widget XawCVItemToWidget (*widget, itemid*)
  > int XawCVWidgetToItem (*widget, item_widget*)
  > void XawCVGetCurrentItem (*widget, itemid*)
  > void XawCVGetPreviousItem (*widget, itemid*)
  > void XawCVGetNextItem (*widget, itemid*)

- Play all sound segments under application control.

  This function allows applications to add audio playback mechanisms beyond the single-segment playback automatically provided by the SoundViewer that the ChatViewer creates for each segment.

  > void XawCVPlayAllItems (*widget*)

- Get information about a chat item, and modify it. Get the number of items in the chat.

In addition to adding and creating items, some applications may wish to change items. The ChatViewer stores items in a database, but the application has no direct access to that database and must use these routines to retrieve and store data about items.

> void XawCVGetItemInfo (*widget, itemid, item_info*)
> void XawCVSetItemInfo (*widget, itemid, item_info*)
> int XawCVNumItems (*widget*)

- Write out chats in either a database format, or as a chatfile, and set the default path to look for sound files.

  The widget reads in chats when it is created; these routines enable the application to save chats whenever the user requests it.

> void XawCVWriteDB (*widget, fileptr*)
> void XawCVWriteChat (*widget, fileptr*)
> void XawCVDefaultSoundPath (*widget, sound_path*)

The calling sequences for each of the public routines and details about the programmatic interface to the ChatViewer are given in Appendix B. One point worth noting is that within an application, items are numbered consecutively and can be referred to by an *item sequence number.*

Widgets can define actions, and map user input events such as keystrokes or mouse button presses to those actions. The current implementation of the ChatViewer does not define user interactions, although the interactions defined by the sound and text widgets are available to users. The ChatViewer widget itself relies upon the application to handle ChatViewer-related user interactions, and provides the necessary public routines to be called in response to user input.

5.2.1 Chatfiles

The ChatViewer can accept either a chatfile or a database file as input, and can parse chatfiles to build a database. A *chatfile* contains sound and text items, in a concise and editable format. A database file is simply an ASCII representation of the contents of a database; it includes each record and each field. Fig. 5.2 shows a short chatfile.

The syntax for specifying sound segments is as follows:

`<sound file:` *sound-filename* [*partial-segment-specification*]>

```
<sound file: /sound/hindus/browse/listentest:::2950>
<sound file: /sound/hindus/browse/listentest::2950:6300>
<sound file: /sound/hindus/browse/listentest::6300:8832>
```

Figure 5.2: Sample chatfile
This chatfile describes the chat shown earlier in this chapter.

where *partial-segment-specification* is:

:: [*start-time*] [: *end-time*]

Start-time and *end-time* are given in milliseconds. If the *start-time* is not given, it is assumed to be the beginning of the sound file. Similarly, if the *end-time* is missing, the end of the sound file is used. So, to refer to the complete sound file, all that is needed is the *sound-filename*, preceeded by the standard prefix.

The chatfile representation does not currently include a specification for marked segments or for the speaker's name.

5.3 Implementation

A widget written using the Xt Toolkit follows a particular canonical form for how resources are defined and the widget code is organized. This section describes the aspects of ChatViewer widget implementation that transcend that canonical form.

Class Hieraracahy

The ChatViewer is subclassed from the Athena widget set; its immediate superclass is the Layout widget[4]. The ChatViewer uses the SoundViewer widget to represent sound items, and the Athena AsciiText widget for text items [MIT89].

Data Structures

The ChatViewer needs to maintain two kinds of information about its subsidiary sound and text items. One kind of information pertains to the items as widgets. This information includes the *widget id*, or pointer to the SoundViewer or AsciiText widget's own data structure. Because widgets are defined as reusable objects, the item widget's data

[4]The Layout widget was written by Russ Sasnett and Mark Ackerman.

structure includes all other information about that widget, including its position, size, attributes, and so on. The ChatViewer maintains a list of widget ids, ordered by item sequence number, so that both the widget id and the item sequence number can be used to reference an item and its corresponding widget.

```
{Record
{{Contents} {/sound/hindus/browse/listentest\00}
{SoundSpeaker} {\00}
{SoundStart} {0\00}
{SeqNum} {1\00}
{Type} {SoundItem\00}
{SoundLen} {2950\00}
{SoundStop} {2950\00}
}
```

Figure 5.3: Sample ChatViewer database record
This record describes the first segment in the sample chatfile in Fig. 5.2.

The other kind of information describes the item as it pertains to the chat. These data are stored in a database and managed by the database library routines. Fig. 5.3 shows sample records from a chat database. For both text and sound items, there is a field for the type of item, along with one for the item sequence number. For sound items, the record includes the soundfile name; the speaker's name, if known; and the start and stop times within the soundfile. If the item has been marked by the user, a field named Marked is created. For text items, the record includes the textual contents of the item.

Procedures

The ChatViewer does most of its work during initialization and when a new item is added. During initialization, the specified chatfile, if any, is parsed and the initial database is created. If a database file is specified, the database manager performs the parsing and database creation operations. Once the database exists, the initial display is created by adding to the display each item from the database, ordered by the item sequence number.

The procedures that add new items create the SoundViewer or AsciiText widget as appropriate and lay out, or position, the newly-created widget according to the specified layout style. Laying out the items requires taking into account the position of the previous item, constraints such as layout style and vertical offsets for speakers, and the relative position of the item in its window. In some layout styles, the items are "wrapped around" and presented in rows, so the layout procedure has to check for whether the new widget would fall beyond the window boundary, and create another row if necessary. Changing the display after adding and deleting an item is done very simply, by deleting and then re-laying out the widgets with sequence numbers greater than that of the affected widget.

There are also a number of very short routines to support marking and playback operations.

Use of SoundViewer Widget

The ChatViewer represents a sound segment with a SoundViewer widget. SoundViewer widgets display a sound as a rectangular object, as illustrated in Fig. 5.1. The length of the SoundViewer corresponds to the length of the sound segment so that, for example, short segments appear short. For use in the Listener, the SoundViewer was modified and now has resources for the starting and ending point of the segment within a soundfile.

Users can play all or part of the soundfile segment associated with the SoundViewer. While the sound is being played, an indicator bar within the SoundViewer fills in, to show how much of the sound has been heard so far.

The SoundViewer is subclassed from the Ind (short for Indicator) widget[5]. There are a number of display-related resources supported by the SoundViewer, including several different ways of mapping sound segments to rectangular objects. For more information about the SoundViewer, see [Spe91].

Class Methods

The ChatViewer has its own Initialize and SetValues routines, a resource converter for converting a string to an LayoutStyle resource, and a simple GeometryManager. The ChatViewer is a composite widget that creates one child widget, a Form widget, and the Form performs geometry management for the ChatViewer's sound and text children. The ChatViewer's own geometry manager does not perform layout; it just increases the height and width of the ChatViewer itself, to match the size of its Form child. This basic geometry manager was required because the ChatViewer's superclass, the Layout widget, is explicitly designed to not resize itself as its children grow.

[5]The Ind and SoundViewer widgets were created by Mark Ackerman and Chris Schmandt

Source Files

By convention, widgets consist most often of a single source file of C code, along with the customary widget.h and widgetP.h include files. The ChatViewer code is broken up into two source files, ChatViewer.c, the main source file, and chatutil.c. Chatutil.c contains the routines that make use of the database manager; these routines were grouped together so that a different database manager could be substituted without massive code changes. There is also a chatutil.h file.

Chapter 6

Future Directions

This chapter explores the directions in which this research could proceed from the work described in this thesis. The concepts under investigation turned out to be a rich source of inspiration, and generated many ideas, both theoretical and practical, that were beyond the scope of this project.

The first section addresses additional uses of structure. The second section discusses the questions that a small user study might address, and strategies for gathering data from pilot users. Next, issues of segmentation sensitivity and advanced audio techniques are discussed. The remaining sections are concerned with more pragmatic software issues, starting with a revised design for the ChatViewer widget. The chapter ends with a list of recommended software revisions and extensions.

6.1 Contextual Structure and Strategies for Retrieval

In the Introduction to this document, the semi-structured approach to audio was described as encompassing inherent conversational structure, contextual structure, and supplemental structure. The current Listener and Browser focus on conversational structure, however, and do not yet make full use of the other kinds of structure. This is the primary area for additional work, and involves several different issues.

One direction is to exploit the structure of conversations above the discourse level. This means taking into account the major phases of a particular audio interaction. For telephone calls, there is always a period of social intercourse, establishing the speakers and so on, before the caller pursues her reasons for making the call [Sch77]. This meta-structure can be used to inform semi-structured audio.

On a much more pragmatic level, the Listener and Browser need to be extended so that chatfiles and soundfiles can be saved and given names by the user; filenames are currently hardwired. When the Listener is used on an everyday basis numerous files will be created, and if the user does not wish to explicitly name chats, random filenames will be assigned. How should these many files be organized? Should chats be stored in a separate directory? How long should the (sizable) soundfiles be retained? Will users want to group chats logically together, as in folders? If so, on what basis?

6.1.1 Contextual Items

Another mandatory feature is for the Listener to collect and store contextual, or situational, structure. For telephone calls, contextual data includes the other party's name and phone number, if known; the time and date of the call; and the location and extension of the user. The location data can be derived from the location of the telephone extension, and by information from active badges [Wei91]. This context information should be stored automatically with the saved segments. The user could provide an optional descriptive tag for each conversation, though the timestamp is sufficient identification for retrieving a conversation.

For manipulating context items, I envision a popup window that prompts for missing items. Because only on-campus callers will be identifiable by their phone numbers, the popup could include a list of frequent outside callers, ordered by frequency or recency, so that the same names do not have to typed in over and over. It should eventually be possible to link chats to related documents such as files, electronic mail, and so forth.

Context items must be stored in the chat database and, later, written out in conjunction with the chatfile. In order to do the former, the ChatViewer widget routines to get and modify context items will have to be fully implemented. This is straightforward to accomplish using the database manager software. Storing context items in chatfiles is more complicated, because a new syntax of key-value pairs will need to be defined and supported. In a sense, what is needed is an audio markup language, akin to the Office Document Architecture [Hor85] that has been defined for text documents. Version numbers within chatfiles will then be helpful to accommodate the inevitable syntax variations.

6.1.2 Browsing Through Multiple Chats

Once chats are saved regularly and contextual items are available, strategies for retrieving specific chats will become important to users. The Browser will need to be extended to provide access to chats and audio segments of interest, using selection criteria similar to those used by electronic mail programs. All chats would be accessible by date and time, by called-party, and by length. Depending on user-supplied context, chats could also be selected by subject, by type of conversation (phone call, in-person meeting, and so on), and by where the conversation took place.

Collections of chats could be quickly browsed, in two display formats. One representation would be modelled after directory listings, with a text line per chat showing the conversants' names along with the date and the duration of the conversation. The other format would build upon the miniature chat display used by the Browser, and show one miniature for each chat in a collection or directory. If colors were used to represent speakers, then a user could readily identify which chats were with a particular person. This, along with the date of the chat, used as a label, and the shape of the miniature, should greatly assist the recall of specific conversations.

Much more sophisticated kinds of retrieval could be performed by creating retrieval agents. These agents would analyze chats for speech signal characteristics such as emotion and gender. For example, as part of narrowing in on the segment of interest, users could specify that the just-played segment is too low-pitched and so would next be presented with slightly higher-pitched segments to choose among. Another agent could search for segments with a particular prosody, for example, questions. One way to implement these independent agents would be using the Unix *inetd* facility, so that these retrieval agents could execute on the most appropriate machines.

6.2 User Observations

The Listener and Browser tools were implemented for demonstrational use, and I was the only user. While observers pointed out weak points and made helpful suggestions, there was no opportunity to explore user interaction issues as could be done in the course of a pilot study. In addition to providing feedback for improving the user interface, observations of pilot users can address a variety of questions. Some questions of interest include:

- How will using the Listener affect a phone call and the participants?
- How will the resulting chats be used?
- Are segments marked more often during the conversation or afterward?
- After the call ends, how much do participants revise their choice of which segments to save?

Observations of pairs of conversants making a series of calls will enable longitudinal data to be gathered. Some of the questions to be studied are as follows:

- Does the nature of the calls change over time?
- Are users more or less likely to create context items ahead of time as they gain experience with the tools?
- Do participants refer to earlier conversations more frequently?
- Do the conversations become more structured?
- Are participants' perceptions of the success and satisfactoriness of calls affected by having these tools, and if so, how?

Additional questions can be addressed if both participants in the conversation are using Listeners:

- Do the participants agree on segments of mutual interests?
- Are segments spoken by one person more or less likely to be saved by that person?

6.2.1 Practical Considerations for a Pilot Study

One approach is as follows: several pairs of frequent phone conversants will be asked to participate. Sample phone conversations will be recorded beforehand. During the study, these same participants will speak again, this time with the Listener available to at least one participant in each pair. The study should also include occasional videotaping and interviewing of the participants.

It would be best to simply give the Listener and Browser to the pilot users and allow them to incorporate the tools into their daily routine. This may not be realistic, however, because of the specialized equipment required. Another approach is to invite pilot users to make their phone calls from a properly equipped office, though the users would not be in a familiar environment. With this scenario, it would make sense to provide users with the Browser, and their chats, on their home systems.

A note on implementation—the Listener and Browser will need to be instrumented so that usage data can be tabulated automatically. The logged data could include the length and frequency of phone calls; the proportion of calls that are recorded; and segment marking by users, as a percentage of the total call duration and as a count of segments. Retrieval, editing, and playback operations are further candidates for automatic logging.

6.3 Controlling Segmentation Sensitivity

As discussed previously, the Segmenter demarcates segments based on detecting pauses and turns, and its decisions are governed by a set of parameters and constraints with preset values. The values were carefully chosen so that segments would correspond to phrases or sentences by one speaker. In other words, the preset values result in a segmentation that is *sensitive* to certain aspects of conversational structure, such as phrases, and insensitive to others, such as individual words.

That particular level of sensitivity may not be correct, however, for purposes other than segmenting telephone calls, or even for other users during calls. For example, some speakers may pause more often than others; pausing during speech can be affected by situational and individual factors such as anxiety, topic, gender, and personality [Roc73, Rut87].

Therefore, the user ought to be able to control the sensitivity of the segmentation. One way to provide this control would be to make the numerous segmentation parameters separately adjustable. This would be useful to a programmer working on the segmentation procedure, but not very understandable to a person who just wants to find portions of phone calls.

Instead, I propose a different mechanism, one that has a single, linear adjustment for setting the segmentation to be more or less sensitive. The implementation of this adjustment would have to concern itself with changing various parameters as appropriate, but the user would have a simple way to control the granularity of the segmentation; it could vary from effectively no segmentation (a monolithic block of sound), to every turn and pause resulting in a separate segment. The granularity of the segmentation could then be controlled using, for example, a Sensitivity slider mechanism.

6.4 A Handful of Audio Processing Ideas

ISDN dual-channel audio In order to segment based on speaker, two microphones were used. This dual microphone setup is something of a hack, and not generally practical. In the near future, ISDN technology will be integrated with workstations, and then two microphones will no longer be required. In ISDN telephone lines, the audio signal from each speaker is carried separately; once the hardware exists to tap into these digital signals from within a workstation, speaker-based segmentation will be easy and reliable.

Microphone calibration The pause detector is dependent on accurate settings for the recording levels and the speech thresholds, so that speech is distinguished from non-speech correctly. The current Listener relies upon the user to check and adjust the recording levels with the `vumeter` and `xgaintool` programs.[1] However, this calibration could be performed automatically by the software itself, perhaps with an explicit calibration step built into the Listener. Automatic gain control may be another option for accommodating changes in volume.

Non-telephonic conversations An obvious avenue for extending the Listener is to support non-telephonic audio interactions. These could involve more than two people, such as in meetings, or even in-person discussions between just two people. Both of these situations are much more acoustically complex than telephonic calls. Miking each person is often impractical (though eventually microphones may be built into active badges.) Instead, the meeting places could be wired with *environmental audio* capabilities. For large groups, the auto-directional microphone devised by Flanagan and others, though not commercially available yet, presents an attractive solution to the problem of capturing audio from meetings [FBE90]. A capture tool may well be useful in group meetings as part of a note-taking system.

Partial speech recognition Although real-time spoken natural language understanding systems are a number of years away, non-real-time speech-to-text automatic transcriptions could be made available as an overnight service; the segments of interest from a day's conversations could be transcribed by a specialized server connected through a network [Der91]. This is a particularly attractive use of natural language systems, as even partial transcriptions would be of value for later retrieval and recall of conversatons.

Partial recognition could also be incorporated into the real-time segmentation, by limiting the recognition to paraverbals and pre-defined audio flags. For instance, the words "first" and "second" could be recognized as phrase delimiters.

Controlling replay speed For audio scanning of lengthy stored chats, it would be helpful to control the speed of the audio playback. The faster replay may be even more effective, given the chunking provided by segmentation [BM76]. This speed control could be initially implemented using timescale modification of the speech signal, and controlled with a slider-style interaction mechanism. Experience with scanning may indicate that

[1]`Vumeter` and `xgaintool` are Speech Group application programs for monitoring and controlling audio device-related parameters, including audio signal levels.

other approaches to speeding up replay should be used, including automatic elimination of silent periods [Aro92].

Advanced signal processing Several sophisticated speech signal processing techniques could be used to advantage in a Listener. One is speaker *differentiation*; this would obviate the need for microphone-based speaker segmentation, as the program could examine the audio signal and determine which parts of the signal belong to which speaker. Another technique is speaker *identification*, or figuring out who is speaking by matching the speech signal characteristics with stored templates from known speakers. The Listener could even automatically create a template for frequent callers.

Another technique would use spectral filtering to refine pause detection. For an audio signal with a greater bandwidth than telephonic speech (that is, greater than 4000Hz), frequency-based analyses can be used to determine the zero-crossing rate, for example. A medium zero-crossing rate is associated with noise, while a high zero-crossing rate can detect fricatives with low energy values. [O'S87]. Finally, a Listener could classify (and perhaps segment) segments by correlating the prosody of the segments with templates of kinds of utterances. For example, questions can be identified by the ending rising pitch [HP86].

These techniques are not now appropriate for a telephone Listener, due to the lack of speech signal bandwidth, and might not be feasible for real-time use in any case, due to the intensive calculations required. However, advances in telephone technology and signal processing will eliminate these obstacles in the near future.

6.5 ChatViewer Variations

The ChatViewer has two major functions, managing the chat database and displaying the chat to the user. The current ChatViewer widget performs both of these functions, and supports several different kinds of displays.

The future of the ChatViewer is in breaking it up into a number of smaller, more maintainable widgets. The motivations behind this strategy are, first, the code for the ChatViewer now consists of about 2600 lines, which is quite large for a special-purpose widget. Second, it has become obvious that the different modes, or ways of displaying chats, is the area where the ChatViewer is most likely to change. Right now, any change has to be made so that the other modes are not adversely affected.

The existing ChatViewer will be split up into a family of widgets, with one widget as the base class, called the OmniViewer, and several widgets subclassed from that widget, each with its own display semantics. The subclasses would include:

- A ListViewer, for lists of brief sound and text items, akin to the Left mode currently used by the `todo` application;

  This widget could well include a resource to request that the list items be automatically numbered.

- A CallViewer, for dynamically displaying consecutive segments of a continuous conversation with two participants, akin to the Time mode currently used by the Listener during phone calls.

- A MiniViewer, for creating the miniaturized display of a conversation, and based on the current Pan mode used by the Browser.

- A new ChatViewer, for displaying stored conversations; it is closely related to the MiniViewer.

- A ScribeViewer, for transcriptions, that is, text items closely associated with sound items and connected by arrows.

Along with separating out the different display styles, the ChatViewer can be improved in other areas. In its current version, the ChatViewer does not respond correctly to many of application's SetValues requests on the widget's resources. Also, the ChatViewer does not support the selection mechanism available to X applications; selections enable users to cut and paste sound and text items across applications. User interaction is another area requiring work; the application-based approach now taken is overly limiting. Finally, the ChatViewer's resources and callbacks need to be trimmed and the ones worth keeping fully implemented.

Most importantly, the ChatViewer's geometry management will be restructured. Using a Form widget to avoid doing layout semantics was expedient, but is too constraining in the long run. The ChatViewer should be implemented as a true Composite widget, and perform its own layout. This change is anticipated in the plan to break the ChatViewer into multiple widgets; each current layout style will require independent layout semantics, and each layout-specific widget can have just the semantics it needs.

6.6 Other Software Improvements

Representations of Conversations Alternative approaches to displaying conversations should be explored. The current approach concentrates on segments rather than elasped time, and so the representation is not time-based. This is partly due to having one SoundViewer widget per segment; each SoundViewer is independent and shows a sound as if the sound started at time zero. Other representation strategies might preserve time information better. A related issue is that time should be more understandable in the call display, perhaps with an explicit timeline shown below the segments.

Also worth investigating are different visual representations for who is speaking, especially for future Listeners that display segments from three, four or even more speakers. Colors could be used, or the speaker's name could be put inside of each segment.

Minor Improvements The remainder of this chapter consists of the miscellaneous ideas and suggestions that have arisen during the course of this project, loosely grouped by topic. Deciding which of these items is worth pursuing should, of course, be based on more extensive experience and user feedback.

The telephone Listener could be smarter about handling telephone-related events, such as:

- On-going calls being put on hold.
- Initiating recording after the call has started.
- Working with other workstation tools like `xphone` and `xrolo`.[2]
- Phone calls arriving when on a second call appearance when the Listener is already recording a call on the first call appearance.

User interface improvements and affordances for the Listener and Browser:

- The Listener is intended to run in the background, but there should be a visual indicator when it is running—two at once is disastrous, as the audio signal events get intermixed. The indicator could provide additional user semantics, such as starting recording by clicking on it.
- Reinforce the user's focus of attention during a call to the right side of the call display (where new segments appear). One way is to maintain some whitespace at the right end, which is non-trivial to implement for widget-related reasons.

[2]`Xphone` and `xrolo` are Speech Group applications that provide visual interfaces to dialing telephones and looking up phone numbers [SC89].

- Marking a segment could be more approximate than it is now, and mean that the previous *n* seconds of conversation be saved, along with the marked segment itself.
- Marking a group of segments could be done in a single action by sweeping over them with the pointing device.
- To aid in marking, highlight SoundViewers when the cursor moves into them.
- Add playback to the Listener call display.
- In cases of very brief transitions between speakers, the Segmenter could pad the segments with short silences to improve the replay of each segment.
- Display an icon or scanned-image of the other conversant.
- Change cursor shape to provide affordances for various kinds of interactions.
- Improve playback interaction semantics, such as providing an obvious way to interrupt a PlayAll operation. Three ideas: make the PlayAll button act as a toggle, interpret a click anywhere to mean stop the playback or implement VCR-style controls.
- Segments should be playable in arbitrary groups of selected segments, and marked segments should be playable as a group.
- Add a message line to the Listener and Browser applications, perhaps with an on-line help option.
- Provide a resegmentation capability. This will require saving soundfiles of the signals from both microphones.

Miscellaneous missing features and improvements:

- Text annotations should be well-supported.
- Support multiline text items.
- Allow more generalized editing, such as moving items anywhere in 2-D space using a drag-and-drop interaction mechanism.
- Support pasting of sounds across applications, and recording of new sound items within the Browser.
- User configuration settings are needed, such as:
  - Preferred minimum and maximum segment durations.
  - Maximum call length to be recorded.

  - Resources for names and colors for speakers.
  - Maximum number of speakers.
- There are two planned SoundServer-related improvements:
  - Move the pause detection logic into the SoundServer and define events that occur only on transitions between speech and nonspeech.
  - Allow the time interval between energy events to be set by the application program, so that it is not just 100 milliseconds.

Chapter 7

Discussion

In this final chapter, the work described previously is placed into perspective with a summary of the knowledge gained and a discussion of potential applications and organizational impact.

7.1 What Was Learned and Not Learned

This project successfully demonstrated that semi-structured audio is an interesting and realistic methodology for integrating the world of speech with the electronic world. Furthermore, real-time segmentation can be accomplished on general-purpose workstations and with little (none, in the near future) special equipment. With respect to automatic derivation of conversational structure, the approach taken in this project is simple and effective, and does not rely upon sophisticated audio hardware or computation-intensive software methods for determining segments and turns. Futhermore, by dividing the silences between segments, a clean-sounding segmentation is produced.

The Listener and Browser tools allow conversational structure to be displayed and manipulated in a fashion that is original and thought-provoking. The ChatViewer widget itself is a general, reusable implementation for displaying sound and text items, and will be a base for continuing to experiment with different ways of displaying large amounts of continuous and discontinuous audio.

As can be inferred from the length of the prior chapter, Future Directions, I had hoped to implement more aspects of the semi-structured audio concept. I wound up concentrating on deriving and displaying the inherent conversational structure of telephone calls—now, the work needs to move beyond that. The key step is to incorporate contextual structure and annotation, and to provide retrieval mechanisms, as described in section 6.1. At that point, the tools can be used on a routine basis, and users can gain experience with real-time segment-of-interest identification, data retention and retrieval. Researchers can make informal observations of usage, if not go so far as to conduct a small pilot study.

Here are my personal observations on using the Listener. I found it to be comfortable and easy to use during a phone call. The structure shown in the call display did indeed make the conversation approachable from the point of view of picking out the important utterances, and I could readily mark segments of interest. It was interesting, too, to see aspects of conversing that are not normally made apparent, such as the lengths of utterances and patterns of turntaking. I did note that the accuracy (or inaccuracy) of the segmentation often went unnoticed by visitors, presumably because the dynamic display was so engaging. I remained sensitive to segmentation errors with respect to my own utterances.

The most awkward situations occurred due to unanticipated telephone circumstances. Credit card calls are a case in point; the telephone line status switches from Dialing to Active as soon as the local 950-xxxx number answers, and so the Listener winds up recording and segmenting the sound of numerous touch tones. This is annoying and something of a security issue, as the tones for my PIN code were recorded as well. I also encountered problems when I had multiple phone calls at the same time, when I put a caller on hold and when there was Muzak. Extraneous sound of any kind, such as music, the movements of my officemate, or even relatively quiet visitors, interfered with turntaking detection.

A minor but amusing weakness—my demonstrational setup, a Skutch box and a PZM mike, was not very well suited to demos, at least not ones where the visitor was the other party in the telephone conversation. The PZM microphone was too sensitive to sound anywhere in my office, and I wound up having to telephone a third person in another room.

7.2 Applications

The contributions of this project can be applied to a variety of other domains and uses. The telephone Listener is clearly an effective tool for people who conduct business over the phone, such as reporters or writers doing interviews, and support personnel dealing with customer inquiry and assistance calls. Imagine being able to listen to the customer's own words when trying to resolve a problem! The software would have to be tailored for the specific situation, of course.

Another way that this work could be built upon is building listening tools for other types of audio interactions, such as informal discussions in offices and formal meetings. Multimedia computer conferencing systems that make use of audio are a good opportunity for capturing group interactions. (See section 6.4 for a brief discussion of related audio issues.)

The ability to structure and display speech can be applied to previously recorded material as well. Some of the potential audio sources are voice mail, and a new category of professional tool, the personal audio agent. With personal agents, users could record segments while away from their workstation; they could, for example, speak into a microcassette recorder and later plug it into the computer [DMS92].

A related application area is transcribing speech to text, a task that is currently awkward and error-prone. Arons had to use a laborious procedure in creating his hyperspeech application, for instance; an appropriate listening tool could have greatly simplified that task, by providing automatic segmentation, playback on a per-segment basis, and simple entry of closely-associated text. [Aro91a]. This application would call for a somewhat different display representation, as mentioned in section 6.5. Another opportunity is tools for researchers who study conversation, such as in the fields of conversational analysis and sociolinguistics.

One final point—although this project is not specifically directed towards meeting the needs of visually-impaired computer users, the tools produced are potentially of interest to that user community.

7.3 Impact on Users and Organizations

Introducing new technologies can affect users in many ways. If the technology offers enough utility, then it will be used, even if the interface is awkward. If a new technology is not useful, then it is very difficult to convince or coerce users to adopt it. Moreover, a useful innovation may not be used as its designers intended, the technology's use may change over time, and it may affect other aspects of a user's worklife. For all these reasons and more, it is worthwhile to evaluate the impact of an innovation like semi-structured audio.

The simple text-audio database used in this project has the potential to be quite useful in groupware applications, particularly in being a repository of "group memory", that is, information that group members need to collectively know. Group memory often resides in the minds of the group members, not in written form, is transmitted informally by explanation or example, and can be irretrievably lost when a group member leaves.

One risk of a recording tool like this is inciting privacy concerns. To avoid this, no one should be recorded without their knowledge. Restricting recording to those who have previously given their permission is one way to accomplish this. Having the listening tool audibly warn conversants is another possibility. With ISDN telephones, it may well be feasible for users to have their telephone sets communicate a privacy setting, indicating whether it is OK to record or not. Intelligent agents could even use user-defined rules to figure out what the appropriate setting is for a particular call.

There is a societal implication to this work, too. Technologists foresee a time in the not-too-distant future when continuous recording of audio will be the norm, infinite storage will be very low-cost and computer capability will be available for processing all that audio signal data. One motivation for this scenario is that interesting and important statements are being lost to our descendents. In the past, many transactions were conducted via the written word, as letters. Technology in the form of telephones has eliminated that vast source of history, and continuous recording will rectify that lack.

7.4 A Final Word

Semi-structured audio is a promising concept, and the Listener and Browser tools are early steps in an evolving implementation of that concept. The semi-structured approach provides a framework for supporting large quantities of otherwise ill-structured sound data. This work illustrates a new perspective on using computers to support human conversations, and is a novel application of audio techniques. To the extent that computers can share in conversations, their role as partners in the workplace will be enhanced. I hope and believe that my work will change the way that people think about audio and workstations.

Appendix A

User Documentation

A.1 Browser Man Page

Name
browse—Displays chats, collections of sound and text items.

Synopsis
browse [*chatname*]

Description
Browse is a companion tool to *Listen*, and displays the chats created by *Listen*.

Options

[chatname]	This is the name of the chatfile or chat database file that the Browser reads and displays. Also settable as a ChatViewer resource.

Environment
None

Files
chatdb—the default name of the input file

See Also
listen, ChatViewer, Soundviewer

A.2 Browser Widget Tree

```
ApplicationShell xbrowser
      Form top_form
         Form menubox
            MenuButton files
               SimpleMenu filesmenu
                  SmeBSBObject filesmenu1...5, for menu items
                  TransientShell filenameDialogShell
                     DialogWidget filenameDialog
                        Command filenameDialogOK
                        Command filenameDialogCancel
            MenuButton locate
               SimpleMenu locatemenu
                  SmeBSBObject locatemenu1...5, for menu items
            Command playall
            Command quit

         ChatViewer mini_widget
            Form ...
               SoundViewer ...
               AsciiText   ...
         Viewport datavp
            ChatViewer browser_widget
               Form ...
                  SoundViewer ...
                  AsciiText   ...
```

A.3 Browser Resource File

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!  xbrowser -- resources for browser application
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
*bitmapFilePath: /usr/local/include/X11/bitmaps

xbrowser*.title:                Browser
xbrowser*menubox.borderWidth:   0

xbrowser*files.label:           Files
xbrowser*files.menuName:        filesmenu
xbrowser*files.sensitive:       False
xbrowser*locate.label:          Locate
xbrowser*locate.menuName:       locatemenu
xbrowser*locate.fromHoriz:      files
xbrowser*locate.sensitive:      False
xbrowser*playall.label:         Play All
xbrowser*playall.sensitive:     True
```

```
xbrowser*playall.fromHoriz:     locate
xbrowser*quit.label:            Quit
xbrowser*quit.fromHoriz:        playall

xbrowser*filesmenu1.label:      Load chat
xbrowser*filesmenu2.label:      Load text
xbrowser*filesmenu3.label:      Save chat
xbrowser*filesmenu4.label:      Save text
xbrowser*filesmenu5.label:      Directory

xbrowser*filenameDialog.label:  Filename dialog box
xbrowser*filenameDialog.value:  Initial value
xbrowser*filenameDialogOK.accelerators: #override \
        <Key>Return:         set() notify() \n\
        <Key>Linefeed:       set() notify()

xbrowser*filenameDialogOK.label:          OK
xbrowser*filenameDialogCancel.label:      Cancel
xbrowser*filenameDialog*Text*width:       200
xbrowser*filenameDialog*Text*borderWidth: 1
xbrowser*filenameDialog.value*resize:     never

xbrowser*filesmenu1.sensitive:  False
xbrowser*filesmenu2.sensitive:  True
xbrowser*filesmenu3.sensitive:  True
xbrowser*filesmenu4.sensitive:  False
xbrowser*filesmenu5.sensitive:  False

xbrowser*locatemenu1.label:     All
xbrowser*locatemenu2.label:     Date
xbrowser*locatemenu3.label:     Speaker
xbrowser*locatemenu4.label:     Type
xbrowser*locatemenu5.label:     Place

xbrowser*locatemenu1.sensitive: False
xbrowser*locatemenu2.sensitive: False
xbrowser*locatemenu3.sensitive: False
xbrowser*locatemenu4.sensitive: False
xbrowser*locatemenu5.sensitive: False

*ChatDir:      browse

xbrowser*mini_widget.layoutStyle:     XawlayoutStylePan
xbrowser*mini_widget.fromVert:        datavp
xbrowser*mini_widget.height:          100
xbrowser*mini_widget.width:           200
xbrowser*mini_widget.topMargin:       4
xbrowser*mini_widget.speakHorizDist:  2
xbrowser*mini_widget.speakVertDist:   10

xbrowser*mini_widget*SoundViewer*height:       2
xbrowser*mini_widget*SoundViewer*borderWidth:  2

xbrowser*datavp*resizable:      False
```

```
xbrowser*datavp.fromVert:       menubox
xbrowser*datavp.forceBars:      True
xbrowser*datavp.allowHoriz:     False
xbrowser*datavp.allowVert:      True

xbrowser*browser_widget.layoutStyle:  XawlayoutStyleTime
xbrowser*browser_widget.height:       300
xbrowser*browser_widget.width:        600
xbrowser*browser_widget.borderWidth:  0
xbrowser*browser_widget.playable:     True
xbrowser*browser_widget.editable:     True

xbrowser*ChatViewer.Form.borderWidth:     0
xbrowser*ChatViewer*Text*type:            string
xbrowser*ChatViewer*Text*displayCaret:    False
xbrowser*ChatViewer*Text*borderWidth:     0
xbrowser*ChatViewer*AsciiText*resize:     both
xbrowser*ChatViewer*Text*wrap:            word
xbrowser*ChatViewer*Text*height:          20
xbrowser*ChatViewer*Text*width:           600
xbrowser*ChatViewer*AsciiText*autoFill:   True
xbrowser*ChatViewer*Text*editType:        edit
xbrowser*ChatViewer*Text*translations:        #override \
        <Enter>:        display-caret(on) \n\
        <Leave>:        display-caret(off)

xbrowser*SoundViewer*height:          13
xbrowser*SoundViewer*width:           350
xbrowser*SoundViewer*widgetDuration:  20000
xbrowser*SoundViewer*indMode:         XtIndModeTruncated
xbrowser*SoundViewer*borderWidth:     3
```

A.4 Listener Man Page

Name
listen—dynamically structure and record a telephone conversation

Synopsis
listen [-debug] [-synch]

Description
Listen is an X Window System application that implements semi-structured audio for telephone conversations. It creates a dynamic display of the conversation and allows users to indicate which parts of the conversation should be retained. After the phone call is over, the conversation can be replayed and further filtering performed. A description of the saved portions of the conversation is saved in the chat database file named chatdb. The recorded audio is saved in a file called listentest.

Options
-debug If this option is present, a great deal of output will be written to stdout, showing for each energy event (every 100 milliseconds or less!) the data values and how the pause detector and turn detector process it. Also settable as an application resource.

-synch If this option is present, an explicit synchronization step will take place when the handset is picked up. You will be prompted to say "Go", and when that window is dismissed, you can proceed. Also settable as an application resource.

Environment
MY_DN—Specifies the user's own extension. This is where the Phoneserver looks for phone-related events.

Files
chatdb—This file is the chat database file that is created.
testchatfile—This file is the chatfile format file that is created.
/sound/hindus/browse/listentest—This file is the soundfile that is created.

See Also
browser, ChatViewer, SoundViewer

Bugs
Listen must be stopped and restarted after each phone call.
File handling is dreadful.
This is experimental software.

Notes
The audio from the telephone must be fed into the user's Sparcstation audio input jack. To do this from an ISDN phone line requires a device like the Skutch box used when this software was created.

Turntaking detection currently relies on a second microphone connected to another workstation.

A.5 Listener Widget Tree

```
ApplicationShell xlisten
   TopLevelShell browserShell
      Form top_form
         Form menubox
            MenuButton files
               SimpleMenu filesmenu
                  SmeBSBObject filesmenu1...5, for menu items
                  TransientShell filenameDialogShell
                     DialogWidget filenameDialog
                        Command filenameDialogOK
                        Command filenameDialogCancel
            Command marknew
            Command quit

         Form databox
            Label speaker_label_1
            Label speaker_label_2
            Viewport chatvp
               ChatViewer browser_widget
                  Form ...
                     SoundViewer ...
                     AsciiText   ...
            Label clock

   TransientShell call_alert
      DialogWidget call_alertDialog
         Command call_alertDialogOK
         Command call_alertDialogCancel

   TransientShell browser2_shell
      Form top_form2
      Command playall2
      Viewport datavp
         ChatViewer browser_widget2
            Form ...
               SoundViewer ...
               AsciiText   ...

   TransientShell synch
      DialogWidget synchDialog
         Command synchDialogOK
         Command synchDialogCancel
```

A.6 Listener Resource File

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!  xlisten-- resources for listener
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!*debug: True
!*synch: True

xlisten*call_alert.title:                 Notifier
xlisten*call_alertDialog.label:           Listening Agent
xlisten*call_alertDialog*width:           200
xlisten*call_alertDialog*height:          100
xlisten*call_alertDialog*Command.height: 30
xlisten*call_alertDialogCancel.label:     Cancel
xlisten*call_alertDialogOK.label:         OK
xlisten*call_alertDialogOK.accelerators: #override \
        <Key>Return:         notify()

xlisten*synchDialog*AsciiText*width:     200
xlisten*synchDialog.label: Synchronizer\n\n\nDial until no dialtone\n\n
Then press Synch
xlisten*synchDialogOK.accelerators: #override \
        <Key>Return:         notify()
xlisten*synchDialogOK.label:         Synch
xlisten*synchDialogCancel.label:     Done
xlisten*synchDialog*width:           200
xlisten*synchDialog*height:          200
xlisten*synchDialog*Command.height:  30

xlisten*browserShell.title:      Call Display

xlisten*menubox.borderWidth:     0
xlisten*menubox*font:            sun_cour16b

xlisten*files.label:             Files
xlisten*files.menuName:          filesmenu
xlisten*files.sensitive:         False
xlisten*marknew.label:           Mark New Segments
xlisten*marknew.sensitive:       True
xlisten*marknew.fromHoriz:       files
xlisten*quit.label:              Quit
xlisten*quit.fromHoriz:          marknew

xlisten*filesmenu1.label:        Load chat
xlisten*filesmenu2.label:        Load text
xlisten*filesmenu3.label:        Save chat
xlisten*filesmenu4.label:        Save text
xlisten*filesmenu5.label:        Directory

xlisten*filenameDialog.label:    Filename dialog box
xlisten*filenameDialog.value:    Initial value
```

```
xlisten*filenameDialogOK.accelerators: #override \
        <Key>Return:         set() notify() \n\
        <Key>Linefeed:        set() notify()

xlisten*filenameDialogOK.label:          OK
xlisten*filenameDialogCancel.label:      Cancel
xlisten*filenameDialog*Text*width:       200
xlisten*filenameDialog*Text*borderWidth: 1
xlisten*filenameDialog.value*resize:     never

xlisten*filesmenu1.sensitive:   False
xlisten*filesmenu2.sensitive:   True
xlisten*filesmenu3.sensitive:   True
xlisten*filesmenu4.sensitive:   False
xlisten*filesmenu5.sensitive:   False

xlisten*databox.borderWidth:    0
xlisten*databox.fromVert:       menubox

xlisten*speaker_label_1.label:           Debby
xlisten*speaker_label_1.font:            sun_cour16
xlisten*speaker_label_1.borderWidth:     0
xlisten*speaker_label_1.justify:         right
xlisten*speaker_label_1.width:           100
xlisten*speaker_label_1.internalWidth:   10
xlisten*speaker_label_1.horizDistance:   5
xlisten*speaker_label_1.vertDistance:    41

xlisten*speaker_label_2.label:           Callee
xlisten*speaker_label_2.font:            sun_cour16
xlisten*speaker_label_2.borderWidth:     0
xlisten*speaker_label_2.justify:         right
xlisten*speaker_label_2.width:           100
xlisten*speaker_label_2.internalWidth:   10
xlisten*speaker_label_2.horizDistance:   5
xlisten*speaker_label_2.vertDistance:    2
xlisten*speaker_label_2.fromVert:        speaker_label_1

xlisten*clock.label:           00:00
xlisten*clock.font:            sun_cour18
xlisten*clock.width:           80
xlisten*clock.horizDistance:   10
xlisten*clock.height:          40
xlisten*clock.vertDistance:    45
xlisten*clock.fromHoriz:       chatvp

xlisten*chatvp.resizable:      False
xlisten*chatvp.forceBars:      True
xlisten*chatvp.allowHoriz:     True
xlisten*chatvp.allowVert:      False
xlisten*chatvp.fromHoriz:      speaker_label_1
xlisten*chatvp.height:         100
xlisten*chatvp.width:          510
```

```
xlisten*browser_widget.height:       75
xlisten*browser_widget.width:        500
xlisten*browser_widget.layoutStyle:  XawlayoutStyleTime
xlisten*browser_widget.playable:     False
xlisten*browser_widget.editable:     False

*ChatViewer.ChatName:  listenchat
*ChatDir:              browse

xlisten*browser2_shell.title:             Post-Call Browser
xlisten*browser2_shell.x:                 250
xlisten*browser2_shell.y:                 450
xlisten*browser2_shell*playall2.label:    Play All
xlisten*browser2_shell*datavp.fromVert:   playall2
xlisten*browser2_shell*datavp.height:     300
xlisten*browser2_shell*datavp.width:      600
xlisten*browser2_shell*datavp*resizable:  False
xlisten*browser2_shell*datavp.forceBars:  True
xlisten*browser2_shell*datavp.allowHoriz: False
xlisten*browser2_shell*datavp.allowVert:  True

xlisten*browser_widget2.layoutStyle:      XawlayoutStyleTime
xlisten*browser_widget2.playable:         True
xlisten*browser_widget2.editable:         False

xlisten*ChatViewer.Form.borderWidth:     0
xlisten*ChatViewer*Text*type:            string
xlisten*ChatViewer*Text*displayCaret:    False
xlisten*ChatViewer*Text*borderWidth:     0
xlisten*ChatViewer*AsciiText*resize:     both
xlisten*ChatViewer*AsciiText*wrap:       XawtextWrapLine
xlisten*ChatViewer*Text*height:          20
xlisten*ChatViewer*Text*width:           600
xlisten*ChatViewer*AsciiText*autoFill:   True
xlisten*ChatViewer*Text*editType:        edit
xlisten*ChatViewer*Text*translations:    #override \
        <Enter>:        display-caret(on) \n\
        <Leave>:        display-caret(off)

xlisten*browser_widget.topMargin:         22
xlisten*browser_widget.speakHorizDist:    2
xlisten*browser_widget.speakVertDist:     16

xlisten*SoundViewer*height:          15
xlisten*SoundViewer*borderWidth:     4
xlisten*SoundViewer*markHeight:      6
xlisten*SoundViewer*width:           100
xlisten*SoundViewer*widgetDuration:  7000
xlisten*SoundViewer*indMode:         XtIndModeTruncated
xlisten*SoundViewer*record:          False
```

Appendix B

ChatViewer Reference

B.1 Synopsis

The ChatViewer is intended to be a general-purpose widget for handling simple audio and text objects, called chats. It is a subclass of Core, Composite and Layout (the latter provides a "just say yes" geometry manager for composite widgets).

Audio items are represented as SoundViewer widgets. Text items are represented as AsciiText widgets. The ChatViewer supports items as being sequenced.

Within the ChatViewer, a chat is implemented as a db-style database.

B.2 Resources

Name	Default	Type	Access	Class
XtNautoLayout	True	Boolean	CG	XtCAutoLayout
XtNchatDir	NULL	String	CG	XtCChatDir
XtNchatName	NULL	String	CG	XtCChatName
XtNeditable	True	Boolean	CG	XtCEditable
XtNfileDescriptor	-1	Int	CG	XtCFileDescriptor
XtNfinishedPlayCallback	NULL	String	C	XtCCallback
XtNlayoutStyle	XawLayoutStyleLeft	LayoutStyle	CG	XtCLayoutStyle
XtNnotifyMarkCallback	NULL	String	C	XtCCallback
XtNplayable	True	Boolean	CG	XtCPlayable
XtNrowVertDist	30	Position	CG	XtCSowVertDist
XtNspeakHorizDist	2	Position	CG	XtCSpeakHorizDist
XtNspeakVertDist	7	Position	CG	XtCSpeakVertDist
XtNstartPlayCallback	NULL	String	C	XtCCallback
XtNtopMargin	7	Position	CG	XtCTopMargin
XtNwhenToScale	10000	Int	CG	XtCWhenToScale

Access refers to how the resource can be acted on. C means, set at creation time. G means, can be retrieved via GetValues. S means, settable via SetValues. The ChatViewer does not currently support SetValues reliably.

autoLayout Supposed to indicate whether the widget automatically lays out the chat items or not. Not currently supported.

chatDir Specifies where to look for the sound files referred to in the chatfile. Assumed to be a sub-directory within the user's /sound directory.

chatName The filename containing the text or ASCII database version of the chat. If the file looks like a db-style database, it is treated as such. Otherwise, the file is assumed to be a chatfile.

editable Supposed to indicate whether items can be edited or not. Not currently supported.

fileDescriptor This is the Unix fd for the soundfile. It is passed to the Soundserver. If this is negative, sound items should be insensitive.

finishedPlayCallback Supposed to call functions when a Play operation completes. Not currently supported.

layoutStyle Specifies which layout style the widget should use. It is an enumerated type.

XawlayoutStyleLeft Each item is left-justified on its own line. Text and sound items are both laid out properly in this style.

XawlayoutStyleTime Sound segments are laid out horizontally, in order of their appearance in the chatfile or their sequence number in the chat database file. In database files, one speaker's name can be included, in which case the segments with a non-null speaker field are placed vertically offset from segments with a null speaker field.

The widget will "wrap" the displayed segments to another row if a segment would be drawn past the right-hand side of the widget.

Text items are not well-handled in this layout style.

XawlayoutStylePan This style is similar to the Time style, only it's intended to make a miniature display.

A converter is defined, so that this resource can be specfied as Left, Time or Pan in a resource file.

notifyMarkCallback Functions on this list are called when the user clicks on a widget in marking mode.

playable Supposed to indicates whether sound items can be played or not. Not currently supported.

rowVertDist The number of pixels to separate each row by vertically. Applies only to the Time style.

speakHorizDist The number of pixels to separate each segment by horizontally. Applies only to the Time style.

speakVertDist The number of pixels to vertically offset segments from different speakers. Applies only to the Time style.

startMarkCallback Supposed to call functions when a Play operation begins. Not currently supported.

topMargin The number of pixels to vertically separate the first row from the upper edge of the widget. Applies only to the Time style.

whenToScale A segment duration past which the SoundViewer is displayed in Scaled mode. Prevents lengthy segments from being truncated.

B.3 Public Data Structures

```
typedef struct Context {
    int nspeakers;
    char *speakers[MAX_SPEAKERS];
    char *datetime;
    char *host;
    char *phone;
    char *text;
} Context;
typedef struct Item {
    int type;
    char *contents;
    long sound_start;
    long sound_len;
    long sound_stop;
    char *speaker;
    Boolean marked;
} Item;
#define ITEM_ID int
```

B.4 Public Routines

In all these routines, the first argument, *widget*, specifies the ChatViewer widget on which the routine will perform its actions.

Itemid refers to a sound or text item in the chat, and is the item's sequence number. Sequence numbers are one-based.

void XawCVAddItem (*widget, itemid, item_info*)
    Widget *widget*;
    ITEM_ID *itemid*;

Item **item_info*;

Adds a new item just before item *itemid*, using data in *item_info.*

void XawCVAddItemAtEnd (*widget, item_info*)
Widget *widget*;
Item **item_info*;

Adds a new item after all other items, using the data in *item_info.*

void XawCVDeleteItem (*widget, itemid*)
Widget *widget*;
ITEM_ID *itemid*;

Deletes an item from the display and from the database.

void XawCVMoveItem (*widget, from_itemid, to_itemid*)
Widget *widget*;
ITEM_ID *from_itemid*; ITEM_ID *to_itemid*;

Repositions an item within the item sequence.

void XawCVGetItemInfo (*widget, itemid, item_info*)
Widget *widget*;
ITEM_ID *itemid*;
Item **item_info*;

void XawCVSetItemInfo (*widget, itemid, item_info*)
Widget *widget*;
ITEM_ID *itemid*;
Item **item_info*;

Returns or sets data for an item.

void XawCVPlayAllItems (*widget*)
Widget *widget*;

Plays all sound items in the widget in order.

void XawCVWriteDB (*widget, fileptr*)
Widget *widget*;
FILE *fileptr;

Writes ASCII db file to file in fp.

void XawCVWriteChat (*widget, fileptr*)
Widget *widget*;
FILE *fileptr;

Writes ASCII chatfile to file in fp.

void XawCVMarkItemInDB (*widget, itemid*)
Widget *widget*;
ITEM_ID *itemid*;

Creates a database field that indicates marking in the item's database record.

void XawCVUnmarkItemInDB (*widget, itemid*)
  Widget *widget*;
  ITEM_ID *itemid*;

  Removes the database field that indicates marking from the item's database record.

void XawCVVisualMarkItem (*widget, itemid*)
  Widget *widget*;
  ITEM_ID *itemid*;

  Displays the visual indicator of marking on an item.

void XawCVVisualUnmarkItem (*widget, itemid*)
  Widget *widget*;
  ITEM_ID *itemid*;

  Removes the visual indicator of marking from an item.

Widget XawCVItemToWidget (*widget, itemid*)
  Widget *widget*;
  ITEM_ID *itemid*;

  Returns the widget id of an item, given its item sequence number.

int XawCVWidgetToItem (*widget, item_widget*)
  Widget *widget*;
  Widget *item_widget*;

  Returns the item sequence number of an item, given its widget id.

int XawCVNumItems (*widget*)
  Widget *widget*;
  Returns the total number of items in the widget.

void XawCVDefaultSoundPath (*widget, sound_path*)
  Widget *widget*;
  char **sound_path*;

  Returns the widget's default sound path into *sound_path*.

void XawCVGetCurrentItem (*widget, itemid*)
  Widget *widget*;
  ITEM_ID id;

  Returns the item sequence number of the current item, but the concept of current item has not been implemented.

void XawCVGetContextInfo (*widget, itemid, context_info*)
  Widget *widget*;
  ITEM_ID *itemid*;
  Context **context_info*;

  Returns data for context. Not fully implemented.

void XawCVSetContextInfo (*widget, itemid, context_info*)
  Widget *widget*;

ITEM_ID *itemid*;
Context **context_info*;

Sets data for context. Not fully implemented.

void XawCVGetPreviousItem (*widget, itemid*)
Widget *widget*;
ITEM_ID *itemid*;

void XawCVGetNextItem (*widget, itemid*)
Widget *widget*;
ITEM_ID *itemid*;

These routines allow an app to proceed forwards or backwards through the items.

Bibliography

[Aro91a] B. Arons. Authoring and transcription tools for speech-based hypermedia systems. In *Proceedings of 1991 Conference*. American Voice I/O Society, September 1991.

[Aro91b] B. Arons. Hyperspeech: Navigating in speech-only hypermedia. In *Hypertext '91*, pages 133–146. ACM, 1991.

[Aro92] Barry Arons. A review of time-compressed speech and preliminary thoughts on speech scanning. Technical report, MIT Media Lab, January 1992.

[AS86] Stephen Ades and Daniel C. Swinehart. Voice annotation and editing in a workstation environment. Technical Report CSL-86-3, Xerox Palo Alto Research Center, Sept 1986.

[AS90] P. J. Asente and R. R. Swick. *X Window System Toolkit*. Digital Press, 1990.

[BB79] Geoffrey W. Beattie and P. J. Barnard. The temporal structure of natural telephone conversations (directory enquiry calls). *Linguistics*, 17:213–229, 1979.

[BHB77] Brian Butterworth, R. R. Hine, and K. D. Brady. Speech and interaction in sound-only communciation channels. *Semiotica*, 20:81–99, 1977.

[BM76] D. S. Beasley and J. E. Maki. Time- and frequency-altered speech. In N. J. Lass, editor, *Contemporary Issues in Experimental Phonetics*, chapter 12, pages 419–458. Academic Press, 1976.

[CFK91] Barbara L. Chalfonte, Robert S. Fish, and Robert E. Kraut. Expressive richness: A comparison of speech and text as media for revision. In *Proceedings of the Conference on Computer Human Interaction*, pages 21–26, New York, Apr 1991. ACM.

[Cro91] Terrence Crowley. Voice in a multimedia document system. *Journal of The American Voice I/O Society*, 9:21–26, Mar 1991.

[DASP91] Glorianna Davenport, Thomas G. Aguierre Smith, and Natalio Pincever. Cinematic primitives for multimedia. *IEEE Computer Graphics and Applications*, 11(4):67–74, 1991.

[Der91] Michael L. Dertouzos. Building the information marketplace. *Technology Review*, pages 29–40, 1991.

[DMS92] Leo Degen, Richard Mander, and Gitta Salomon. Working with audio: Integrating personal tape recorders and desktop computers. To appear in Human Factors in Computer Systems – CHI'92 Conference Proceedings, 1992.

[FBE90] J. L. Flanagan, D. A. Berkley, and G. W. Elko. Autodirective microphone systems. Preprint of invited paper for a special issue of Acustica honoring Professor G. M. Sessler, 1990.

[FKL88] Robert S. Fish, Robert E. Kraut, and Mary D.P. Leland. Quilt: a collaborative tool for cooperative writing. In *Conference on office information systems*, pages 30–37, Palo Alto, CA, 1988. ACM.

[FS69] W. Foulke and T. G. Sticht. Review of research on the intelligibility and comprehension of accelerated speech. *Psychological Bulletin*, 72:50–62, 1969.

[FSM+91] K. Fujikawa, S. Shimojo, T. Matsuura, S. Nishio, and H. Miyahara. Multimedia presentation system "Harmony" with temporal and active media. In *USENIX Conference Proceedings*, pages 75–93, 1991.

[Hor85] W. Horak. Office document architecture and office document interchange formats: Current status of international standardization. *IEEE Computer*, pages 50–60, October 1985.

[HORW87] G. Herman, M. Ordun, C. Riley, and L. Woodbury. The modular integrated communications environment (MICE): A system for prototyping and evaluating communications services. In *Proceedings of the 1987 International Switching Symposium*, pages 442–447, 1987.

[HP86] Julia Hirschberg and Janet Pierrehumbert. The intonational structuring of discourse. In *Proceedings of the Association for Computational Linguistics*, pages 136–144, July 1986.

[KEE90] Ragui Kamel, Kamyar Emami, and Robert Eckert. PX: supporting voice in workstations. *IEEE Computer*, 23(8):73–80, Aug 1990.

[LN91] Mik Lamming and William Newman. Activity-based information retrieval: Technology in support of human memory. Technical Report 91-03, Rank Xerox EuroPARC, Feb 1991.

[MA90] Thomas P. Moran and R. J. Anderson. The workaday world as a paradigm for CSCW design. In *Computer Supported Cooperative Work – CSCW'90 Conference Proceedings*, pages 381–393, 1990.

[MD89] Wendy E. Mackay and Glorianna Davenport. Virtual video editing in interactive multimedia applications. *Communications of the ACM*, 32(7):802–810, 1989.

[MD90] M. J. Muller and J. E. Daniel. Toward a definition of voice documents. In *Proceedings of COIS '90*, 1990.

[MGL+87] T. W. Malone, K. R. Grant, K-Y. Lai, R. Rao, and D. Rosenblitt. Semi-structured messages are surprisingly useful for computer-supported coordination. *ACM Transactions on Office Information Systems*, 5(2):115–131, 1987.

[MIT88] MIT X Consortium. *X Toolkit Intrinsics—C Language Interface.* 1988.

[MIT89] MIT X Consortium. *Athena Widget Set—C Language Interface.* 1989.

[MMC+89] Wendy E. Mackay, Thomas W. Malone, Kevin Crowston, Ramana Rao, David Rosenblitt, and Stuart K. Card. How do experienced information lens users use rules? In *Human Factors in Computing Systems, CHI 89 Proceedings*, pages 211–216. ACM, 1989.

[MN86] Yoshiro Miyata and Donald A. Norman. Psychological issues in support of multiple activities. In Donald A. Norman and Stephen W. Draper, editors, *User Centered System Design*, chapter 13. Lawrence Erlbaum and Associates, 1986.

[OC74] R. B. Oschman and A. Chapanis. The effects of ten communication modes on the behavior of teams during co-operative problem solving. *International Journal of Man/Machine Systems*, 6:579–619, 1974.

[O'S87] D. O'Shaughnessy. *Speech Communication: Human and Machine.* Addison-Wesley, 1987.

[Par86] Thomas Parsons. *Voice and Speech Processing.* McGraw Hill, 1986.

[Pot90] Mary C. Potter. Remembering. In Daniel N. Osherson and Edward E. Smith, editors, *An Invitation to Cognitive Science: Thinking*, volume 3, chapter 1. MIT Press, 1990.

[Roc73] S. R. Rochester. The significance of pauses in spontaneous speech. *Journal of Psycholinguistic Research*, 2:51–81, 1973.

[Ron88] John Ronayne. *The Integrated Services Digital Network: From concept to application.* Pitman/Wiley, 1988.

[RS77] Derek R. Rutter and G. M. Stephenson. The role of visual communication in synchronising conversation. *European Journal of Social Psychology*, 7:29–37, 1977.

[RS90] Stephen Reder and Robert G. Schwab. The temporal structure of cooperative activity. In *Computer Supported Cooperative Work – CSCW'90 Conference Proceedings*, pages 303–316, 1990.

[Rut87] Derek R. Rutter. *Communicating by Telephone.* Pergamon Press, 1987.

[RV92] Paul Resnick and Robert A. Virzi. Skip and scan: Cleaning up telephone interfaces. To appear in Human Factors in Computer Systems – CHI'92 Conference Proceedings, 1992.

[SA85] C. Schmandt and B. Arons. Phone Slave: A graphical telecommunications interface. *Proceedings of the Society for Information Display*, 26(1):79–82, 1985.

[SAH90] Christopher Schmandt, Mark S. Ackerman, and Debby Hindus. Augmenting a window system with speech input. *IEEE Computer*, 23(8):50–56, Aug 1990.

[SC89] C. Schmandt and S. Casner. Phonetool: Integrating telephones and workstations. In *Proceedings of GLOBECOM '89*. IEEE Communications Society, November 1989.

[Sch77] Emanuel A. Schlegoff. Identification and recognition in interactional openings. In Ithiel de Sola Pool, editor, *The Social Impact of the Telephone*, chapter 19. MIT Press, 1977.

[Sch81] C. Schmandt. The Intelligent Ear: A graphical interface to digital audio. In *Proceedings IEEE Conference on Cybernetics and Society*, pages 393–397, October 1981.

[Sch90] C. Schmandt. Caltalk: a multi-media calendar. In *Proceedings of 1990 Conference*. American Voice I/O Society, 1990.

[SHAM90] Christopher Schmandt, Debby Hindus, Mark Ackerman, and Sanjay Manandhar. Observations on using speech input for window navigation. In *Proceedings of the IFIP TC 13 Third International Conference on Human-Computer Interaction*, pages 787–793, Aug 1990.

[Spe91] Speech Research Group. Software documentation, 1991.

[Sti91] L. J. Stifelman. Not just another voice mail system. In *Proceedings of 1991 Conference*, pages 21–26. American Voice I/O Society, 1991.

[Sun90] Sun Microsystems. *SunOS 4.1 Operating System Manual*. 1990.

[Tho77] Bertil Thorngren. Silent actors: Communication networks for development. In Ithiel de Sola Pool, editor, *The Social Impact of the Telephone*, chapter 17. MIT Press, 1977.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific American*, 265:66–75, September 1991.

[Won91] Chi Chong Wong. Personal communications. Master's thesis, MIT, 1991.

[WW77] Chris Wilson and Ederlyn Williams. Watergate words: A naturalistic study of media and communications. *Communications Research*, 4(2):169–178, 1977.

[Zel89] Polle T. Zellweger. Scripted documents: a hypermedia path mechanism. In *Hypertext 89 Proceedings*, pages 1–14, Pittsburgh, PA, 1989. ACMSIGCHI.

[ZTS88] P. Zellweger, D. Terry, and D. Swinehart. An overview of the etherphone system and its applications. In *Proceedings of the 2nd IEEE Conference on Computer Workstations*, pages 160–168, Santa Clara, CA, March 1988.

[Zue91] Victor W. Zue. From signals to symbols to meaning: On machine understanding of spoken language. In *Proceedings of the International Phonetics Congress*, 1991.