MAS.632 Conversational Computer Systems
Fall 2008

12

Desktop Audio

Because of limitations in speech technology, its successful deployment has been limited to certain situations in which voice offers unique advantages over visual interfaces or in which no other interface modality was practical. This bias has been reflected in the case studies as well, with their emphasis on hands-and-eyes busy environments or tasks oriented around managing voice communication with other people. This chapter instead focuses on applications of voice in an ordinary desktop computing environment used for day-to-day office activities. To continue the theme from the end of the last chapter, once computers manage our telephones and take voice messages for us, what else can we do with this stored speech?

Although nearly every office is equipped with a computer as well as a telephone, desktop voice processing is only beginning to become a reality. This gap is due in large part to the lack of a convincing argument for desktop audio; much office work is handled effectively with the use of keyboard and mouse to access text-based applications. To appreciate the potential of voice applications in the office we must first understand how speech can benefit office work as well as the synergy of using voice across a range of applications instead of in an isolated niche.

For decades computers were used to perform one task at a time; now window systems and multiprocessing operating systems give the user the ability to interact with many applications simultaneously. What is true with text-based software will be even truer with voice: no single application will dominate computer usage or provide sufficient benefit to warrant the purchase of voice processing capabilities. Rather in the future we can expect families of voice-capable applica-

tions running on the desktop sharing data and interaction techniques. Equally important to the need to share data among voice applications are the means to interchange information between voice and text databases and utilize presentation methods that allow telephone access to the desktop and other portable devices.

This chapter describes the concept of **desktop audio,** an environment that supports multiple voice applications on a single desktop computer and remote voice access to office databases. These ideas have already been touched upon in previous chapters: Xspeak (see Chapter 8) used speech recognition at the desktop to coordinate window-based interaction with multiple applications, Phone Slave (see Chapter 11) included a variety of telephone-related functions in a single application, and Voiced Mail (see Chapter 6) provided an integrated remote interface to voice and text messages.

This chapter considers several aspects of desktop audio. It first explores strategies that can smooth the transition from text-only to multimedia computing in the office and then presents a variety of graphical user interfaces to stored voice, which can help overcome its slow and serial nature in the desktop environment. The text advocates a client-server based software architecture to support multiple voice applications simultaneously. The notion of voice as a means of capturing much of our normal office conversations for later retrieval is discussed. In closing, the chapter presents a number of case studies to illustrate these desktop audio concepts.

EFFECTIVE DEPLOYMENT OF DESKTOP AUDIO

Most new workstations[1] are now equipped with a speaker and a microphone and increasingly fast processors allow software-based implementations of speech recognition, text-to-speech synthesis, audio compression algorithms, and time-scale modification to intelligibly play voice back in less time than originally spoken. Yet deployment of voice applications has proceeded slowly and lags behind the underlying technology. This delay is due in part to the inability of software developers to target applications for which voice offers a benefit to the user, in part due to the difficulty of developing user interfaces that overcome the slow and serial nature of voice as data, and in part due to software architecture constraints interfering with the development of applications to be portable across a variety of hardware platforms.

From the desktop audio perspective, much of the difficulty lies in the lack of any single ideal voice application. To date the most prevalent speech application is voice mail, but current stand-alone implementations of voice mail are deployed and efficiently managed as part of the telephone system, not a networked com-

[1]The term "workstation" is used in this chapter to refer to any desktop computer. With ever-increasing microprocessor speeds, the distinction between "workstation" and "personal computer" is becoming increasingly blurred.

puter resource. The delay in proliferation of voice applications is exacerbated by the lack of integration between new voice-capable software and existing applications. If each speech application uses its own distinct user interface, it is harder still on the user to access multiple applications. The absence of a single "killer" desktop voice application does not mean speech will not be useful but it does suggest that it is essential to consider how speech applications will work with each other, how voice will enhance existing workstation applications, and what unique new capabilities voice will enable.

To succeed voice must offer supplemental value to the existing productivity tools on workstations rather than replace those tools. Users have not been willing to give up functionality or applications that they are already using for the sake of adding voice; voice capability must instead augment these applications as well as introduce additional new ways in which to use the computer. Augmenting instead of replacing functionality can be accomplished by maintaining compatibility with current text-based databases and applications wherever possible. For example, the multimedia mail system developed as part of Carnegie Mellon University's Andrew project [Morris *et al.* 1986] added the ability to include nontextual enclosures such as animations within Email messages. Users of a non-Andrew system could view the text portion of the message simply as conventional email without the animation, enabling Andrew users to continue to exchange messages with the non-Andrew Email users. But Andrew multimedia messaging never caught on in the larger non-Andrew world because it was so tightly embedded with the rest of the Andrew environment and hence not very portable. Recently, the MIME message format [Rose 1993] has been proposed for use in the Internet community to allow a broader exchange of multimedia messages in a heterogeneous operating environment. MIME is based on the desire to make multimedia capabilities available across the Internet without any modification of mail transport protocols or software and includes software to add minimal multimedia support to many mail reader programs across many operating systems. Rather than the all-or-nothing approach of Andrew (and similar multimedia mail systems), MIME attempts to add new media ability while minimizing disruption to traditional text-based mail. Desktop voice applications must do the same.

But even if voice can be added to the desktop with minimal disruption, what new capabilities will it provide? Chapter 4 already discussed the expressive richness of voice and its useful role as a document type in and of itself. With an appropriate graphical representation, voice can be mixed with media such as text and image as well and even moved between applications. Ordinary office conversations and meetings can be recorded for future use allowing this important record of decision-making activity to be archived and shared. Perhaps most powerful, speech user interfaces will allow computers to be accessed in new, nontraditional work environments.

Integrating telephone access with a unified voice application environment can result in a synergistic increase in the utility and need of voice as data. Speech synthesis can translate text into voice, enabling access to many text databases over voice telephone connections. This functionality will prove most valuable for retrieving timely personal information while traveling or working away from the

office; databases may include telephone numbers and addresses, one's personal calendar, and electronic mail. Although voice access will not eliminate the use of modems, it is an attractive alternative in that it provides rapid access from any telephone obviating the need for additional hardware.

Perhaps even more significant will be the resulting changes in existing applications for dynamic support of stored voice as a data type. When users can access databases over the telephone some will wish to update them concurrently. If a caller wishes to enter data such as an entry in a calendar or the reply to an email message, recording a voice snippet is much easier than trying to type text with the telephone keypad. However, since we cannot yet reliably translate voice into text, the new entry must remain a voice file, and the underlying database becomes multimedia. The combination of voice and text data, in turn, has repercussions on the graphical interfaces of desktop applications accessing that database. And screen-based voice mail with audio cut and paste allows all the user's telephone messages to become data for manipulation by other applications as well.

Voice interfaces allow the telephone to be used as a computer terminal, and with the advent of highly portable cellular telephones, users can access their desktop from almost anywhere. But other portable technologies will utilize voice, and, in turn, contribute to the role of voice as a data type on the desktop. As laptop computers shrink to palmtops and smaller, the display and keyboard become limiting factors to further size reduction. Voice interfaces require little space; hand-held computing appliances may soon be used to record voice notes for later inclusion in other desktop applications [Stifelman *et al.* 1993]. For example, one may use a hand-held digital recorder to gather thoughts for a presentation while taking a walk or driving to work and later organize the recorded ideas at the office. Once uploaded into a desktop computer, graphical tools could allow the user to further edit, organize, and annotate the spontaneously spoken ideas into a coherent outline or a multimedia document.

GRAPHICAL USER INTERFACES

Although the previous section pointed out that nonvisual user interfaces could enable computer use in novel situations, most applications of voice as data also benefit from visual interfaces when used on the desktop. Graphical representations of audio cue the user to the presence of voice data in an application as well as allow control of playback. Richer graphical interfaces allow direct manipulation of the stored voice by providing a means of positioning playback at random points throughout the sound as well as offering visual cues to a sound's overall length.

Simple "sound button" graphical interfaces (see Figure 12.1) use an icon such as a sketch of a speaker to alert the user to the presence of a sound. The user can click on the icon to play the sound and then click again to stop. Buttons are advantageous because they require little screen space, allowing many of them to be used by an application or to be mixed with text (see Figure 12.2). Buttons are

Figure 12.1. An assortment of "button" representations to indicate the presence of sound in a visual user interface. These representations are from (left to right) NeXT, Digital's Xmedia, and Sun's OpenWindows.

Figure 12.2. Because they are small, buttons can be easily mixed with text. This figure shows voice embedded in a text email message, on a NEXT computer.

intuitive to operate for most users familiar with graphical interfaces. The chief disadvantages of buttons are that the user has no sense of how long a sound will last once it starts and must listen to the entire sound sequentially. The lack of any sense of the duration of a sound while it plays makes it difficult to quickly search a number of sounds for desired information and shows many difficulties associated with similar graphical interfaces [Myers 1985]. The inability to skip around within an audio segment renders buttons useful only when they are used to control very short sounds.

An alternate form of graphical user interface displays additional information about the sound file and allows greater playback control; such visual representa-

tions of sound map time to the horizontal dimension, showing a bar the length of which indicates sound duration. The user plays the sound by clicking on the bar, and as playback progresses a cursor moves in synchronization to indicate temporal position. With many of these interfaces the user can change the playback position by clicking to reposition the cursor. Chapter 4 illustrated a variety of time bar representations, many showing speech and silence intervals. The SoundViewer is a more recent example of such an interface; it is described below as a case study.

The time bar representation has several advantages over the simple sound button by indicating the duration of the sound and providing finer control over playback. Its main disadvantage is the amount of screen space it requires. This difficulty has led to implementations in which the presence of a sound within an application is indicated by a small icon; when this icon is clicked on or dragged to another location a larger control panel opens up for playback, sound editing, and other functions (see Figures 12.3 and 12.4). Such an interface affords many of the advantages of the time bar display while saving screen space (especially useful if multiple sounds are visible), although it does require additional effort on the part of the user, and more importantly requires the user to shift gaze to the location where the control panel appears.

AUDIO SERVER ARCHITECTURES

An appropriate software architecture is required to support access to desktop audio by multiple applications. Current audio servers support digitization and playback of voice, but future servers will also incorporate software implementations of the speech processing technologies discussed throughout this book. Software-based speech recognition, speech synthesis, audio data compression, time-scaling, and other signal manipulation algorithms are feasible on today's workstations; products to perform these operations are already becoming available. When recognition and synthesis required additional external hardware devices or add-in cards, they were costly and unwieldy. When available as software, these technologies can be considerably less expensive and more widely available and hence attractive to application developers.

Audio servers allow voice resources (speaker, microphone, voice processing algorithms) to be made available to multiple applications running simultaneously on one workstation. A server-based approach allows distributed audio processing and resource management among a variety of client applications. The relationship between an audio server and its clients is similar to that found in a server-based window system; instead of directly manipulating audio devices, clients make requests to the server, which controls devices on behalf of the clients. Because clients do not manipulate hardware directly, each client can operate without knowledge of the other clients. The server, or a separate policy agent operating in concert with the server, arbitrates conflicting requests (for a window system, the policy agent is the window manager). For example, if one client requests the server to play a sound while another client is already in the

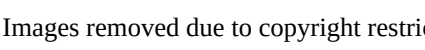


Figure 12.3. The NeXT audio control panel. In the top form sound is represented only as a bar; by clicking a button the user can invoke the waveform envelope display shown at the bottom.

midst of playing a different sound, various policies might be: make the new client wait, make the first client stop playing, or mix the sounds and play them simultaneously.

Because the server approach divides client and server into separate processes that communicate via a well-known protocol, several additional benefits are realized. An application need not be recompiled to execute with a different version of

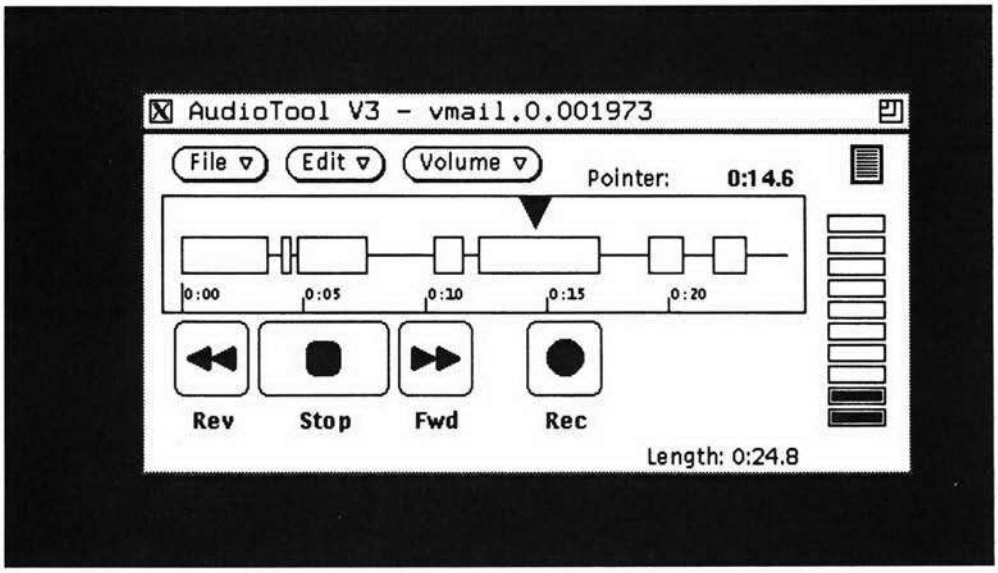


Figure 12.4. Sun's audiotool control panel displays speech and silence segments and provides editing capabilities.

Courtesy of Sun Microsystems, Inc. Used with permission.

the server; the same application binary file can run on different workstations with different audio hardware, simplifying software distribution. Additionally, because the server implies an interprocess communication path between client and server, a client can be made to communicate transparently with servers on its local or remote workstations across the network.

Another advantage of an audio server is that it frees the client application from the time-critical task of reading and writing data between the audio devices and disk files. The server also shelters the application from the detailed programming sequences that control the audio devices; the audio protocol is device independent. These characteristics lead to more modular and portable application software.

The client-server approach is evident in many of the projects described elsewhere in this book, such as Etherphone and MICE (see Chapter 11). An early audio server was developed at the M.I.T. Media Lab [Schmandt and McKenna 1988] in 1984; as with the two examples just mentioned, this server executed on separate hardware from the client. The availability of inexpensive add-in hardware and the real-time nature of the server's DOS operating system allowed the audio server to be implemented on a personal computer that communicated with a Unix host over a serial interface. This early server was motivated largely by the difficulties of managing the real-time audio data stream on early slow Unix workstations. Once playback of a sound begins, it must continue to completion without interruption; stops and starts in the audio medium are much more jarring to the user than an interrupted window refresh on a display.

All audio data was stored on a dedicated disk within the server hardware. The server incorporated a request **queue;** a client could ask the server to prepare for a series of play and record operations (e.g., the prompts and message-taking for an answering machine) and then let the server execute them without further intervention. By queuing requests, an application allowed the server to perform some time-consuming operations in advance such as creating and opening files and ensured that back-to-back play requests could be performed with no silence in between. The client received asynchronous events notifying it of completion of each task.

This server was later extended to include speech recognition, speech synthesis, and several audio compression algorithms, but never fully supported multiple clients' access to the server simultaneously. To do so would have required the server to associate requests with clients and maintain the correct state of devices for each client. For example, one client may set the output gain high, while another sets it low; the server would need to reset the gain of the audio device to the appropriate client-dependent level.

The Olivetti VOX audio server [Arons *et al.* 1989] provided more sophisticated protocols and implemented resource management for multiple simultaneous clients. It digitized and played sounds back from files managed by the server, but VOX also included external computer-controlled analog mixing and switching equipment to route audio to several speakers located in the same or different offices. To support multiple clients, VOX allowed clients to build hierarchical *CLAUDs* (Composite Logical Audio Devices) as server-side constructs. A CLAUD defined all the audio resources needed by a client, much as a window hierarchy specifies all of the graphical objects belonging to a client of a window system server. Again, as with window systems, clients could request that a CLAUD be **mapped** (activated) or **unmapped** (deactivated). Mapping would fail if resources were unavailable because they were currently utilized (mapped) by another client. A CLAUD contained a *Logical Audio Device* (LAUD) for each physical device it included; a LAUD stored the desired state of the physical devices for a particular client. When a CLAUD became active, the physical devices were set to the state appropriate for the client as contained in its LAUDs.

The Digital Equipment Corporation's Xmedia audio server [Angebranndt *et al.* 1991] incorporates the server-side client resource constructs of VOX (the composite audio device hierarchy), but the mixing and routing of audio paths are completely digital. The protocol for this server is more heavily influenced by that of the X window system, including a more sophisticated protocol than VOX for allowing a "media manager" to implement resource management policy. The Xmedia server also provides audio data paths to the client; in addition to recording or playing from a file, the client can provide or access audio data directly. Its internal logic for mixing and synchronizing multiple simultaneous data streams extends to control digital video as well as audio.

The Xmedia server was originally designed in a framework including an audio toolkit to free clients from the need to directly interface to the low-level audio server protocol, but only a minimal toolkit has been developed to date. A recently

published audio server architecture from Hewlett-Packard shows a design very similar to that of the Xmedia server [Billman *et al.* 1992].

Because of the limited availability of any of the audio servers described so far, a new server has recently been developed at the M.I.T. Media Lab; it is used by many of the applications described later in this chapter [Arons 1992b]. This server does not contain any of the server-side resource management constructs from VOX or the Xmedia server but does implement a simple priority-based resource management policy. Clients can send audio data to the server or have it play from files and receive asynchronous notification of audio events. This server includes software for time-scale modification (as well as touch tone detection) making it simple for any application to vary playback speed. Other servers, e.g., for speech recognition and synthesis, are designed to work in concert with the audio server by using its ability to route copies of audio data streams. For example, a recognition server will always get copies of audio input while recognition is enabled. If some other client requests the server to record a sound, the same audio data will be copied into a file. Finally a recording-level client (e.g., to display a graphical "VU meter") would request its own copy of audio data whenever another client activates audio recording.[2]

The Media Lab's audio server is motivated primarily by the need for effective resource management because many applications share the audio device as well as audio data itself. In the absence of a policy agent, or audio manager, this server implements a simple priority scheme. Clients may declare themselves to be of high (urgent), medium (normal), or low (background) priority; higher priority requests supersede lower priority ones, whereas lower or equal priority requests are queued until the server is idle. When a play operation is superseded by another of higher priority, playing pauses and the client is notified; when the new request finishes, the interrupted one resumes. An interrupted record operation does not resume without client intervention.

One component of a software architecture missing from the servers just discussed is an **audio toolkit.** Graphical user interfaces are usually implemented in conjunction with software toolkits consisting of a set of direct manipulation objects such as scroll bars, sliders, and dialog boxes.[3] Audio toolkit objects might include constructs such as a **menu,** which speaks the user's choices and awaits a selection via touch tone, or a **form,** which speaks, prompts and records replies. Resnick's Hyperspeech project implemented a comprehensive set of telephone interaction objects to facilitate rapid development of telephone-based community bulletin boards [Resnick 1992b, Resnick 1992a, Malone *et al.* 1987]. Another such toolkit is described in [Schmandt 1993]; it is heavily used by Phoneshell, described below.

[2]Actually the meter client would request audio level events whereby the server provides periodic updates as to the audio energy level and thereby minimizes the data being transmitted from server to client.

[3]These are the *widgets* of the X-windows community.

Arons describes another hypothetical set of "audgets"—audio widgets—in [Arons 1991a]. An audio toolkit should be distinct from a graphical toolkit as some audio applications have no graphical interfaces. Graphical audio controllers, such as the SoundViewer described later, are best implemented as a separate toolkit layer, as shown in Figure 12.5, so as to free nonvisual or nonaudio applications from the added baggage of the unused media.

UBIQUITOUS AUDIO

Most applications of desktop audio involve explicit invocation of recording activity; the user clicks the mouse or presses a touch tone key to digitize a short snippet of audio. Such recordings comprise but a small fraction of the speech we produce during a work day. We hold meetings, converse on the telephone, chat with visitors, and catch up on news at the water cooler. All of this talk has so far been beyond the reach of computer applications, but desktop audio configurations could allow recording and subsequent retrieval of much of this previously evanescent "ubiquitous audio."

Assuming 10:1 data compression of telephone-quality speech with half of the day spent in conversation of which 80% is speech and the remainder silence, a year's worth of office conversation would require slightly over 2 Gigabytes of storage. Although this quantity would have seemed outrageous as recently as several years ago, such storage capacities are rapidly becoming feasible. The combination

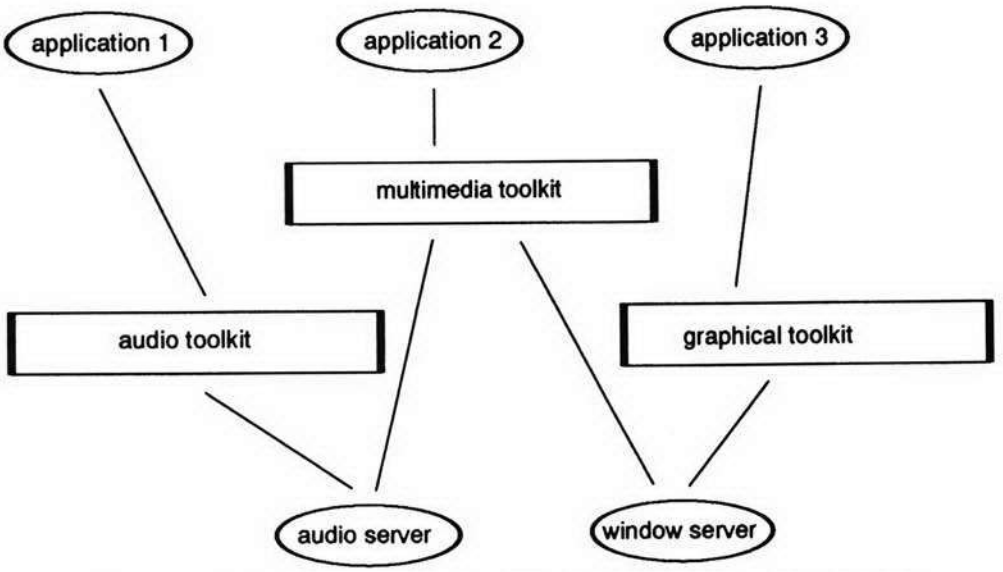


Figure 12.5. Audio and graphical toolkits shield applications from details of communication with their respective servers. Multimedia toolkits can be built on top of single-medium toolkits.

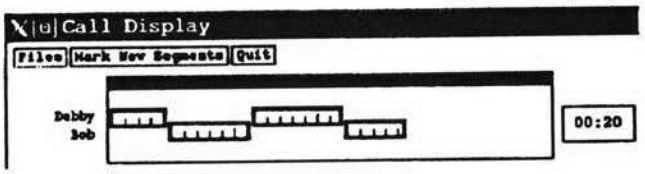
of audio-capable workstations, plentiful storage, and computer-mediated telephony hardware could allow us to record all the conversations that occur in our offices and meeting rooms. There is certainly no doubt that this audio record would contain a wealth of useful information and a valuable archive of business or engineering design decisions, but it would also include a plethora of social or otherwise irrelevant conversations of little archival value.

How could we access the important portions from such a vast quantity of stored voice, which is such a difficult medium to summarize or search? A variety of information sources come into play. Situational information specifies the circumstances in which a recording was made, but nothing about its contents; this includes dates, location, and participants in the conversation. It has been suggested that situational information may be adequate for retrieval of otherwise poorly structured multimedia records such as audio or video recordings [Lamming and Newman 1991]. If users are identified by wearing locator devices, by face recognition from a video record, or by initiating or receiving a call placed by a computer or the telephone network, the audio archive could store a list of talkers with each recording session. Applying situational constraints to form queries such as "Find all long telephone conversations with Eric last spring" can narrow the search immensely.

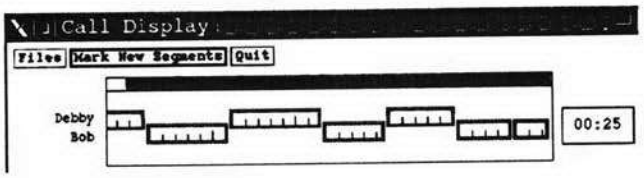
Because of the slow and serial nature of speech, playback of monolithic audio files is tedious regardless of their length. If structure can be added to the recording, it could provide a means of breaking the recording into segments, which indicate reference locations identifying conversational landmarks to assist random access techniques. Structure to aid audio retrieval can be created explicitly by user activity during recording or derived after the fact based on acoustic analysis.

For example, participants in a meeting may take notes on laptop computers; these text entries can be time-stamped and synchronized to the audio recording. Later a graphical conversation browser could tag the visual representation of the recording with the text from the notes or alternatively a text viewer could allow the user to click on a sentence and hear the associated audio. Acoustic analysis can break a conversation into segments delineated by pauses, and this can be incorporated into the visual browser as well. A more powerful method is to identify which speaker is talking at any moment, and visually differentiate the display of each speaker.

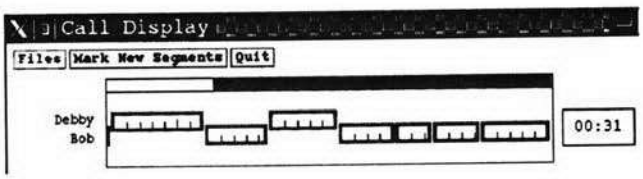
In some situations more explicit user activity could trigger recording, which would yield smaller units to archive, but this action must be unobtrusive. A Media Lab project shown in Figure 12.6 displays a graphical interface on the workstation while a user is engaged in a telephone call [Hindus 1992]. Speaker turns are detected and each scrolls across a small viewing panel showing the recent history of the conversation; each box represents the audio of a single turn. The user's hands and eyes are free during a phone call, and the mouse can be used to start or stop recording at any point. Several portions of the call might be recorded, and each is grouped into a "paragraph," or series of consecutive turn segments. Text annotation can also be added to the speech display. A similar graphical interface can be invoked at a later time for browsing completed telephone calls.



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



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



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

Figure 12.6. A telephone recording tool shows the recent history of a conversation. Each box represents one turn; position differentiates speakers. Reprinted with permission from proceedings of the *ACM 1992 Conference on Computer-Supported Cooperative Work*, © 1992, ACM.

Another Media Lab application, Xcapture, records ambient sound in an office into a circular buffer or "digital tape loop," which serves as a short-term auditory memory aid. The archetypical situation for its use is a collaborative writing session; one person suggests an alternate wording for a sentence while the other hurries to write it down, but neither can repeat the revision word-for-word and it is lost. Xcapture records in the background, displaying a small animated icon with a moving bar. When the user clicks on this icon, recording stops and a SoundViewer widget appears in a popup window (see Figure 12.7); the SoundViewer allows interactive playback of approximately the last five minutes of recording (the precise amount depends on available system memory). The user can review this recording, scan it at a faster speed, save it to a file, or cut and paste it into another application.

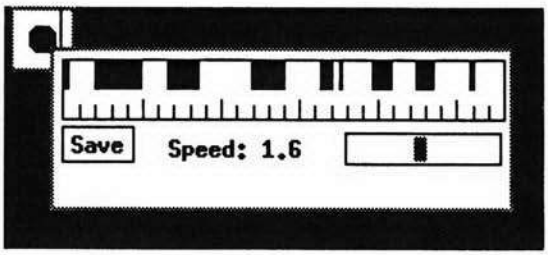


Figure 12.7. Xcapture displays a recording of recent conversation in a pop-up window.

Both these projects demonstrated potential but are limited in capability [Hindus and Schmandt 1992]. Xcapture is barely adequate for even short-term retrieval of a recent conversation because the SoundViewer lacks any cues of turns, pauses, or other structural landmarks in a conversation; even five minutes is a long duration of sound to navigate. Tools which are more sophisticated need to take advantage of multiple sources of structure and acoustic processing to enhance interactive retrieval of archived conversations. A requirement for the user is that it must take much less time to find the desired information than it would to listen to the entire recording sequentially. Playback speed is variable, using the time scale modification techniques described in Chapter 3. A recording can be scanned by playing small segments in sequence, skipping over larger intervening segments. Pauses in the conversation [O'Shaughnessy 1992] or emphasis detected using intonational cues [Chen and Withgott 1992] may suggest segments that are more likely to contain semantically significant utterances or mark the introduction of fresh topics. When coupled with an interactive user interface, such as one based on a touch pad, to scan through a recording at several levels of granularity [Arons 1993], none of these techniques need to work perfectly. Instead, they can act as an aid to the intelligent user, who may have even participated in the conversation being searched and have some memory of its overall structure.

Although recording the ubiquitous audio at work may be straightforward, retrieval from audio archives is difficult. Special capture applications unique to particular recording situations may be useful in the short term. In the long term, more research is needed into techniques to supply structure to recorded sound based on acoustic analysis. Additionally, development of interaction techniques, based in part on the acoustically derived structure, will facilitate retrieval from the audio archive. Whether such factors will impart real value to large audio archives remains to be demonstrated.

CASE STUDIES

This section presents four case studies emphasizing various aspects of desktop audio. The first case study describes the iterative design of a visual representa-

tion of stored voice. The second highlights Conversational Desktop, an eight-year-old project designed to provide functionality spanning many desktop audio application areas. The third case study is of a telephone interface to many common desktop utilities, which is in use today. The final study returns to visual interfaces by examining more recent work in screen interfaces to audio applications such as voice mail; these applications complement the telephone interface in the third case study and are also in use today.

Evolution of a Visual Interface

Visual interfaces to stored audio have been a recurring theme in this book. Chapter 4 pointed out the role of a visual user interface for an audio editor. Earlier in this chapter we returned to the utility and possible functionality of such an interface, presenting two extremes of graphical control of audio playback. A button-style interface requires very little screen space but offers minimal functionality; its small size allows it to be used repeatedly on the computer display by whichever applications contain audio data. These buttons were contrasted with more elaborate interfaces which offer visual representations of the stored voice, provide random access through voice files, and may even support limited editing functionality (but at the cost of size and complexity). In the control-panel style interface, sounds are represented iconically and the control panel appears when a user activates an icon.

This section considers a case study of the Media Lab's SoundViewer, a graphical interface which attempts to provide on-the-spot interactivity combined with small size so that the interface can be used in place by many applications [Hindus *et al.* 1993]. The SoundViewer uses horizontal size and time marks to convey the duration of a sound before a user decides to play it. It provides for direct manipulation at the place on the display where the user's attention is already focused, instead of in a separate control panel. Finally, a SoundViewer can operate in conjunction with other SoundViewers in the same or other applications to facilitate sharing voice data. Because the SoundViewer is an X widget, it can be controlled via X window resources and any change to the widget is inherited by all applications using it (after compilation). An application is required only to set the location of the SoundViewer and specify the sound file it is to control; all other graphical and audio interactions are managed by the widget.

The initial design of the SoundViewer was based heavily on earlier M.I.T. visual interfaces, including the Intelligent Ear (Chapter 4) and Phone Slave's graphical interface (Chapter 11). As with these earlier projects, the visual object's size indicates the length of the sound and its representation changes appearance in synchrony with audio playback. The basic SoundViewer action is shown in Figure 12.8; as the sound plays, a bar of contrasting color moves along left to right. The left mouse button is used to start and stop playback, while the middle button moves the position of the bar. Moving the bar during playback skips to a new location in the voice recording. Repeated clicking on the middle button repeats a segment over and over, which is particulary useful for tasks such as transcribing a phone number from a voice mail meassage. The right mouse but-

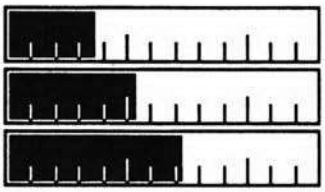


Figure 12.8. As a SoundViewer plays its associated audio file, a bar moves left to right in synchrony.

ton can be used to specify a portion of the SoundViewer, which then becomes the audio *selection* (see Figure 12.9); the selection can be moved to other applications using the standard X protocols.

One issue with visual display of audio data is the mapping between time and pixels, or width, which can provide both relative and absolute indications of sound length. The SoundViewer uses an internal algorithm (or one supplied by the application) to determine the interval at which to display tick marks; indicating time with too much temporal detail would just blur the marks. Although the heights of the tick marks are meant to differentiate them (e.g., those at every second are shorter than those at five-second intervals), no users have ever understood this without instruction. SoundViewer width is a very good indicator of relative length, however. But scale becomes a problem when short sounds are mixed with much longer sounds; the time-to-pixels ratio which just allows a 7 minute sound to fit on a screen would likely result in a 5 second sound being so short as to be rendered almost invisible. The SoundViewer therefore was made to support a mode in which sounds less than a specified threshold length are displayed at a constant scale to allow relative length comparisons, while longer sounds are "crunched" into allocated space by changing the time scale. Although this also changes the spacing of the tick marks appropriately, and the time bar moves more slowly as a result of greater time-to-width compressions, this duration cue is also poorly understood by users.

The tick marks provide useful navigational cues. For example, while listening to voice mail, users often watch to note the location of the time bar when the caller speaks a phone number; this makes it easy to return to the number at the conclusion of the message. But the ticks convey nothing of the *content* of the sound so an alternate representation was developed to display speech and silence intervals (see Figure 12.10) similar to Etherphone's visuals. More recently, "dog

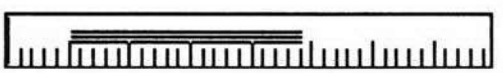


Figure 12.9. Horizontal streaks provide visual feedback when a portion of a SoundViewer's associated sound is selected with the mouse. This SoundViewer shows less temporal detail than those in Figure 12.8.

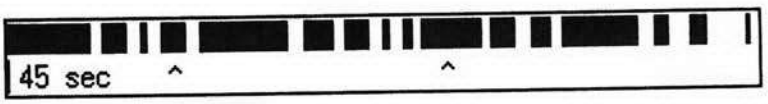


Figure 12.10. The addition of speech and silence marking and "dog ears" to a SoundViewer.

ear" markings were added to the SoundViewer. A spot can be marked (using the keyboard) during playback, and the location of the mark, which appears as a caret symbol, is saved with the sound and appears whenever it is displayed.

After speech and silence intervals had been added, the SoundViewer was made to allow the user to jump ahead to the next speech segment or back to repeat the current or previous segments also under keyboard control. Pauses in the speech correspond to breath groups and are usually deliberate, semantically meaningful speech segments. The SoundViewer also has for some time supported audio time-scaling to play back sounds faster or slower than they were recorded (time-scaling algorithms were discussed in Chapter 3). Because this is a widget "resource," users can set default time scales for all SoundViewers or for those of just a particular application; when users become accustomed to time-scaled speech they begin to prefer it over normal rate speech. A more recent addition provides improved random access; while moving the location bar by hand back and forth across the width of the SoundViewer, it plays small chunks of sound in synchrony. This feedback provides surprisingly useful navigational cues although the proper presentation style is the subject of further research.

The SoundViewer breaks down for very long sounds. When time is mapped to width so as to allow a 10 or 15 minute sound to fit into a reasonable portion of the screen, time is so compressed that the sound bar barely moves as it plays, and it can be positioned only very coarsely. Also, the speech and silence markings interfere with each other when most pauses are shorter than the audio duration represented by a single pixel. Although these effects are not noticed on short sound snippets characteristic of voice mail or calendar entries, they have become problematic for longer sounds such as recordings of meetings or newscasts. A new variant of the SoundViewer provides a global and a detailed view of the sound as shown in Figure 12.11. The lower representation presents a close-up view and is a window into the entire portion of the sound,which appears above. The user can start playback, set a position, or make a selection from either view; as the sound plays, both views get updated simultaneously.

The SoundViewer has been used in all the recent Media Lab desktop speech applications, some of which are described later in this chapter. It has been effective at providing a direct manipulation interface to stored voice files while consuming minimal screen space. Its visual representations have evolved to better facilitate navigation, and additional audio processing enhances the SoundViewer with time-scaling and auditory feedback during random access. Maintaining consistent visual time scales has been a problem from the beginning, and an auxiliary view seems essential for managing very long sounds.

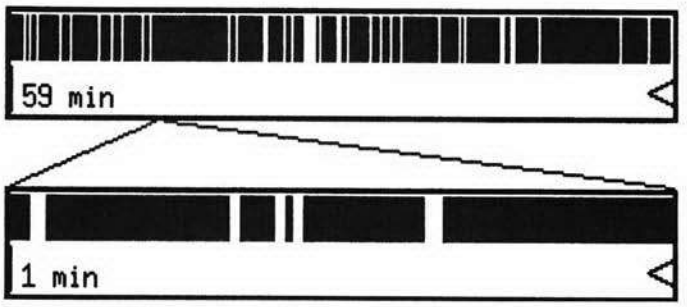


Figure 12.11. For very long sounds two representations are provided. The lower view is a close up of the entire sound shown above.

Conversational Desktop

Conversational Desktop was an early (1985) effort at the Media Lab to present a vision of the possible range of desktop audio applications [Schmandt and Arons 1986]. It emphasized integrated telecommunication management combined with extensive use of voice both as a data type and as an element in the user interface. In addition to spoken language, Conversational Desktop included a touch-sensitive graphical interface which, in anticipation of window systems, allowed the user to quickly switch between displays of the calendar or voice messages. Conversational Desktop's dialogue system and parsing techniques that cope with fragmentary and error-prone speech recognition were described in Chapter 9; this case study focuses on its functionality and integration of multiple application areas.

Conversational Desktop operated across multiple workstations. Each workstation included speech recognition, speech synthesis, audio digitization hardware (implemented on separate dedicated computers acting as servers), and a telephone interface. This project spanned telephone management, voice mail, scheduling, remote database access, and audio reminders triggered by external events. From the standpoint of this chapter, Conversational Desktop's most interesting aspect was the synergy arising from the interaction among these functions.

Conversational Desktop emphasized the role of the workstation in managing both remote data access as well as voice communication through local and wide area networks. The workstation acted as a communication agent, contacting external databases (a traffic information service and simulated access to airline reservation systems) as well as negotiating with agents of other users (scheduling meetings and call setup negotiation). Although services were implemented over analog telephone lines and Ethernet, this project was designed in anticipation of ISDN; calling another Conversational Desktop user established a data connection as well as a voice circuit so the users' agents could communicate while users were conversing.

Conversational Desktop placed telephone calls through voice dialing and incorporated the conversational answering machine approach first implemented in Phone Slave (see Chapter 11). Calls between Conversational Desktop work-

stations were set up using the local computer network before an analog telephone circuit was established; as the calling party was therefore known, the answering machine could play personalized messages and inform callers of the status of earlier messages just as Phone Slave had.[5] Because the Conversational Desktop maintained the user's schedule, outgoing messages could be automatically selected to account for current activity (in a meeting, out to lunch, out of town, etc.). In addition, microphones in the office monitored audio levels to determine when visitors were present, and could automatically take messages from unknown callers without interrupting a conversation in progress.

Because it utilized early connected speech recognition, Conversational Desktop users were required to wear a head-mounted noise-canceling microphone. This microphone was also used during telephone conversations; its noise cancellation allowed use of full-duplex speakerphones. Additional microphones mounted by the walls were used to detect when other people were speaking; speech present at the background microphones but not at the noise-canceling microphone indicated that a visitor was speaking. When the user spoke, a comparison of audio levels in the background microphones could determine the direction he was facing; direction of speaking was used to model user attention and enable recognition. While the user was facing away from the monitor, speech was ignored, but when the user faced the workstation, audio from the microphone was switched to the recognizer. Turning towards the workstation also muted audio to the telephone, so the user could have a private conversation with the computer during a phone call.[6]

Conversational Desktop also allowed its users to record short audio reminders for later playback. When issuing the command to record a reminder, the user specified the situations or activities that would prompt playback such as "when I come in tomorrow." Playback of a single reminder could be triggered by multiple events: a "when I talk to Barry" reminder would be played when Barry called, when the user placed a call to Barry, or when the application announced that it was time for a scheduled meeting with Barry. Reminders were not played, however, when another person was detected in the office as the reminder might contain private information.

This project was an early example of applications of desktop audio. It used speech recognition and a dialogue system to address multiple application functions simultaneously and allowed each function access to stored voice. It explored issues in dynamic routing of audio paths among applications and combinations of speakers, microphones, and voice processing hardware. By incorporating a range of functions, Conversational Desktop demonstrated the synergistic interactions among disparate computer applications dealing with what might appear to be a single operation by the user, e.g., being reminded of something when placing a

[5]Where available, telephone network provided calling party identification and it could serve the same function today.

[6]An unfortunate disadvantage of this arrangement was that the user could not simultaneously read the computer screen and talk on the telephone.

phone call or automatically updating the outgoing voice mail message depending on one's schedule. Finally, Conversational Desktop demonstrated the personalization of call setup, depending on factors such as the calling party and activity in one's office.

Conversational Desktop also had a number of limitations. It was written as a monolithic application, which eliminated the need for interprocess communication but interfered with its modularity and hampered its extensibility. It also side-stepped some important considerations in managing the focus of speech recognition input across multiple applications; if the user is running several programs at once, which is being addressed at any moment? This project utilized custom-built or expensive peripherals, and a good deal of the computing resources of a laboratory group, which relegated it to the role of a demonstration system instead of one to be used in daily work. The audio switching based on direction of speech worked well in a single, sound-treated room, but may not extend well to noisy offices or open cubicles.

Despite these limitations, Conversational Desktop was valuable as an early, visionary system that later motivated much of the work described in several other projects described as case studies in this book; a remaining case study is described in this chapter. Although desktop audio required many exotic peripherals in 1985, all the functionality of Conversational Desktop can be implemented without additional hardware on workstations recently introduced by several vendors.

Phoneshell

Phoneshell [Schmandt 1993] is a family of applications that allows telephone and facsimile access to common desktop utilities including voice mail, electronic text mail, name and address database, and calendar. Phoneshell illustrates the utility of the telephone as a remote link to the desktop, and the enhanced role voice can play as a data type motivated in large part by the difficulty of entering text with a telephone. The telephone interface stimulates the use of voice as a data type and places new requirements on the screen-based applications that must provide access to the stored voice at the desk.

Phoneshell consists of a set of applications loosely linked together under a top-level menu, which allows the user to invoke applications sequentially during a session. The applications share many aspects of their user interfaces, for consistency, and depend on common lower-level utilities, but each application has been developed independently. Figure 12.12 summarizes the functions embedded in Phoneshell.

The voice mail application is similar to many commercial voice mail products with several additional features. Voice messages can originate as telephone messages, from other voice mail users sending a voice message from their workstations, or via digitized voice attachments encapsulated in email arriving over the computer network. In addition to replying to a voice message or recording a message for another user, a caller can record "memo" messages. A memo is intended for screen access only; instead of cluttering the voice mailbox during telephone

Figure 12.12. Phoneshell's menus provide access to a variety of personal information management functions.

access, a separate graphical things-to-do application absorbs the memo messages into its own database. Finally, in addition to being able to transfer a voice message to another user, a Phoneshell user can save an entire message to her calendar by specifying a date; this is the nongraphical equivalent of audio cut and paste.

The email-reading application is similar in many ways to Voiced Mail described in Chapter 6 in that it uses touch tone input to control the reading of email messages using speech synthesis. As with Voiced Mail, Phoneshell carefully pre-

```
Date: Tue, 30 Mar 93 21:01:55 PST
From: Ben.Stoltz@Eng.Sun.COM (Ben Stoltz)
Message-Id: <9303310501.AA13979@denwa.Eng.Sun.COM>
To: geek@media.mit.edu
Subject: Re: your schedule
Status: RO

> From geek@media.mit.edu  Tue Mar 30 15:17:50 1993
> Delivery-Date: Tue, 30 Mar 1993 15:17:53 -0800
> Date: Tue, 30 Mar 93 18:17:44 -0500
> From: Chris Schmandt <geek@media.mit.edu>
> To: stoltz@denwa.Eng.Sun.COM, stoltz@Eng
> Subject: your schedule
>
> What's the possibility of getting together Tuesday?
>


Dinner would be fun. Tuesday evening.
```

Figure 12.13. An email message that contains parts of other messages.

processes text such as email addresses so that they are pronounced correctly and breaks a message into sentences so the caller can skip forward and backward in the message and repeat the sentence currently being spoken. In addition, while reading a message included messages and their mail headers (see Figure 12.13) are detected; headers are summarized ("Header of message from 'Chris Schmandt' in reply to 'your schedule'") and included text can be skipped over by a "jump ahead" button. A message can be faxed as well as read; this is especially useful if reviewing mail about subjects such as an agenda or schedule from a remote telephone.

The user can also send replies using either voice or text. Voice replies are recorded and then returned to the original sender as a voice attachment to an email message using formats from a number of vendors. With voice being supported on a growing number of workstations, such attachments have become a convenient means of reaching many correspondents directly; this had been impossible a decade earlier under Voiced Mail. If the user does not know what message format to use or has no reason to believe the sender uses an audio-capable workstation, text replies can be typed with touch tones. Two keypresses specify each

character; the first indicates the group of three letters and the second selects one of the three depending on whether it is from the left, middle, or right column of keys. Punctuation and digits are also available. Whenever the user completes a word ("*" is the space key), it is echoed. When the user pauses, the current word is spelled out. The most recent word can always be deleted. The message includes an automatically generated addendum explaining that it was entered using touch-tones and the author's "signature" information.

Mail reading is facilitated by the addition of **filtering.** Messages can be sorted into categories such as "urgent," "important," "personal," and "other" based on keywords found in the subject line or on the basis of the author of the message. Phoneshell users can specify arbitrary filtering categories and determine their presentation order. A user who receives large quantities of mail is likely to "read" only the more important categories using speech synthesis simply because this method is so much more time consuming than reading text on a terminal.

Some aspects of Voiced Mail were not carried over into Phoneshell. Because users now get many more email messages, they are less likely to want to hear all their messages so Phoneshell speaks the sender and subject of each message but does not recite the message itself unless requested. The repetition strategy of Voiced Mail (slow down, spell mode) was also abandoned.

The calendar application, Caltalk, lets users scan their calendars and add new entries. A date can be specified with touchtones, and users can hear calendar entries for that day item-by-item; text entries are synthesized and voice entries are played. New entries are recorded and stored as voice annotations. Portions of the calendar can also be faxed if requested.

Although reciting each entry for a day is effective in describing a particular day, in many ways using an auditory calendar interface is more difficult than a graphical interface. Earlier versions of the application provided a "week-at-a-glance" function that merely recited each entry day by day; this ineffectively conveys the overview available from scanning a graphical representation: "The first part of the week is rather free, Wednesday and Thursday have some appointments, and Friday is packed." A more recent version of Caltalk includes new week-at-a-glance and month-at-a-glance functions that attempt to better summarize the calendar and recognize keywords such as "important" in calendar entries as well as entries that span multiple days or are regularly scheduled each week. Caltalk might say, e.g., "Nothing scheduled Monday or Wednesday, important meeting with British Telecom and the usual meetings on Tuesday, you are in Palo Alto Thursday and Friday." Terse summarization is difficult to do well and illustrates some of the problems in converting a tool which usually is accessed visually to a voice-only interface.

A fourth application, Rolotalk, provides access to a personal name and address database. The caller spells a name using touch tones, one tone per letter, to select a "card" from the database; the user can also specify alternate search criteria, such as company name. Once selected, the user can request telephone numbers, postal addresses, electronic mail addresses, and additional information about the selected person. Most useful is Rolotalk's ability to communicate

with the selected person; it can place a phone call to either the home or work number or send a voice or text message. When placing a call, Rolotalk dials the destination party and creates a three-way conference call including itself, the user, and the called party. Rolotalk remains on the line for several minutes; if it hears a "#" tone, it drops the onward call and returns to normal user interaction. This allows the user to remain connected to Rolotalk after dialing a busy number or leaving a message on an answering machine. Message transmission, either voice or text, is accomplished by the same mechanisms employed to respond to electronic mail.

In addition to information stored in explicit fields, the underlying Rolotalk database allows users to specify **links** to other files. Typical links contain maps to the homes or offices of people in the database or driving directions as text. When a user calls in, Rolotalk can try to recite any text file links using speech synthesis or can fax all the links to a nearby fax machine. A user can also fax email messages; this is especially useful for long formatted messages such as a meeting agenda or conference schedule.

Phoneshell also provides several simple communication utilities. A dial-by-name directory enables a caller to spell out the name of a Media Lab staff member and transfer the call to that person's number. Phoneshell can also report the locations of Speech Group members using the activity information described in Chapter 11. The caller can find out who is currently in the lab or logged in from home, call on site users at the nearest telephone, or inquire when a specific user was most recently detected. A Phoneshell user can forward either stored fax messages or an item selected from a small archive of frequently faxed documents (e.g., a map to one's office) to another number.

Although it is currently considered to be in developmental stages as a research project, Phoneshell has been deployed for several years at two locations with as many as 20 users. Its design has benefited from iterative improvements to its user interface due to users' varying requirements and levels of expertise. It has demonstrated the effectiveness of speech as a means of remote access for the mobile user by turning any telephone into a terminal. Although some of the Phoneshell databases are more easily accessed by a laptop computer and a modem, it is often inconvenient to hook up such hardware at a pay phone, to a cellular phone, or in a host's office or home.

Key to the success of Phoneshell has been its integration of multiple communication applications. Reading email over the telephone without any ability to reply would be very frustrating; the recent surge of support for voice encapsulated in email messages facilitates on-the-spot voice replies. It is sometimes useful to send a copy of an email message to someone who appears in one's rolodex. The necessity of calling separate voice mail and text mail applications to retrieve incoming messages would be inconvenient. The ability to query one's calendar while hearing messages about upcoming meeting dates makes it possible to coordinate schedules more effectively. In short, Phoneshell presents a work environment wherein a user can easily switch between as many applications as are required to complete a particular task.

Visual User Interfaces to Desktop Audio

Earlier in this chapter visual user interfaces were extolled as the means of providing the user with random access to voice as data, thereby overcoming some of its slow and serial nature. The visual user interface also acts as a spatial representation, providing a navigational aid for applications presenting multiple snippets of audio. In the desktop audio context, some applications are intrinsically linked to stored voice, such as a graphical interface to voice mail messages. But with the increased availability of audio recording hardware on workstations plus remote telephone access through interfaces such as Phoneshell, many other applications may benefit from stored voice snippets. Both classes of applications require graphical user interfaces to stored speech.

This section describes a family of applications developed in recent years at the Media Lab which operate in concert with Phoneshell as well as some of the telephone management utilities described in Chapter 11. The intent of this case study is to illustrate this chapter's claims about visual interfaces and interapplication communication with actual examples. The SoundViewer widget described earlier in this chapter is used across all of these applications.

The first application to consider is a visual user interface to voice mail, seen in Figure 12.14. When not in use, this application is reduced to a small icon; when new voice mail arrives, the icon blinks, serving as a "message waiting" light. If the user clicks a mouse button on the icon, the message window opens, displaying messages as a column of SoundViewers with indication of the calling party's name or number if known. Vmail also accepts email messages containing voice attachments in several vendor formats; for these the sender's email address is displayed instead of a phone number. When the mouse is moved into the label identifying the caller, text in the upper-right corner of the window shows the date and time at which the message was recorded.

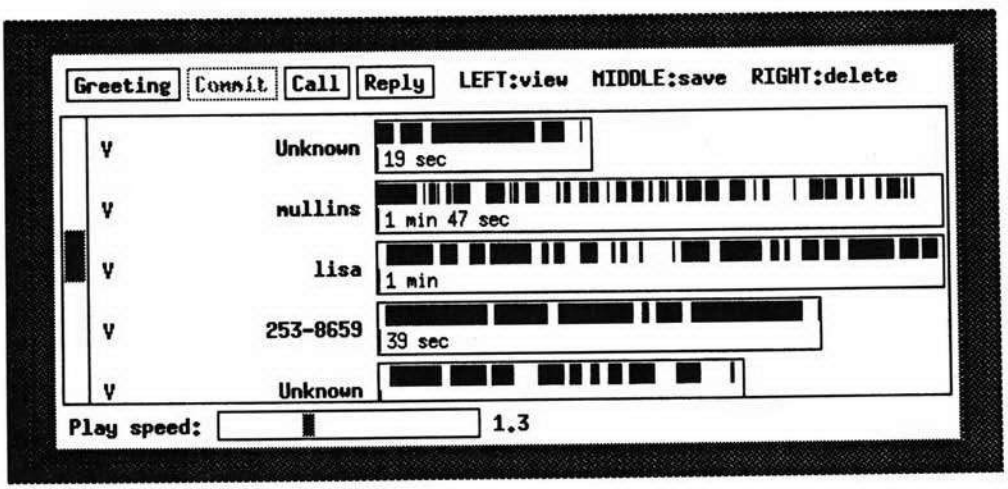


Figure 12.14. A visual interface to voice mail. Each row represents a message, and displays the caller's name or number, if known.

Clicking on a message causes it to play, with the SoundViewer bar moving left-to-right synchronously. While playing, the bar can be manipulated to move around within the message. If a second message is selected, the current message stops and the new message begins playback; this makes it easier to scan through a number of messages looking for one in particular, as messages typically can be identified by playing their first three or four seconds. Playback speed can be changed dynamically by a speed slider at the bottom of the window, and the user can specify a default playback speed for all messages.

After playback, a **V** appears to the left of the message indicating that it has been viewed; messages remain until they are explicitly deleted. Messages can also be saved to a file, in which case the user specifies a file name. If the caller's phone number is known, a return call can be placed by clicking on the "call" button; this sends a call request message to Xphone (described in Chapter 11). If the caller's name or email address is known, the "reply" button sends a request to Xmemotool (see below) to record a voice message in response.

This visual user interface to voice mail has proven very popular among the small community of users at the Media Lab. The direct manipulation SoundViewer interface makes it easier to play portions of the sound repeatedly, specifically while writing down a telephone number. The ability to increase playback speed lets users save time listening to messages, and some of them take advantage of this.[7] Because the application allows multiple users to access the same voice mailbox, a group secretary manages four mailboxes with 20 to 30 messages a day. Although many users delete messages immediately after they have been read, some leave 10 or 20 messages in their mailbox because the visual interface makes it easy to navigate between them and quickly find a particular old message; this is cumbersome with a tone-based telephone interface.

Users can also move messages or portions of messages to other applications. Holding down a mouse button and dragging the cursor across a SoundViewer selects a portion of sound, which can then be pasted into one's calendar, for example, or into the Sedit editor (described in Chapter 4) for further annotation. After editing, the resulting sound file can be forwarded to another user by pasting it into Xmemotool.

Xmemotool (see Figure 12.15) is a window-based utility for recording voice messages. Messages can be composed for other local voice mail subscribers or sent to remote users as email messages, in which case one of several sound file formats must be selected. Xmemotool can also receive the current audio selection instead of taking a new recording; this allows a segment to be cut from a SoundViewer in any application and pasted in as a voice message to be sent.

Xcal (see Figure 12.16) is a visual user interface to a calendar database. In many respects it is similar to a variety of calendar applications with the difference being that Xcal also supports voice annotations. Voice annotations may be recorded directly into Xcal but are more likely to be recorded over the telephone via Phoneshell. Voice entries can also be made by cutting and pasting from other

[7] A playback speed of 1.4 times faster than the original seems popular.

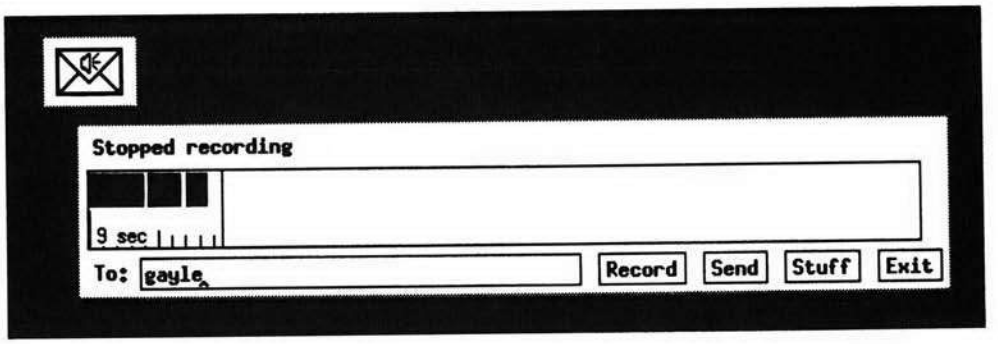


Figure 12.15. Xmemotool, an on-screen voice message taker.

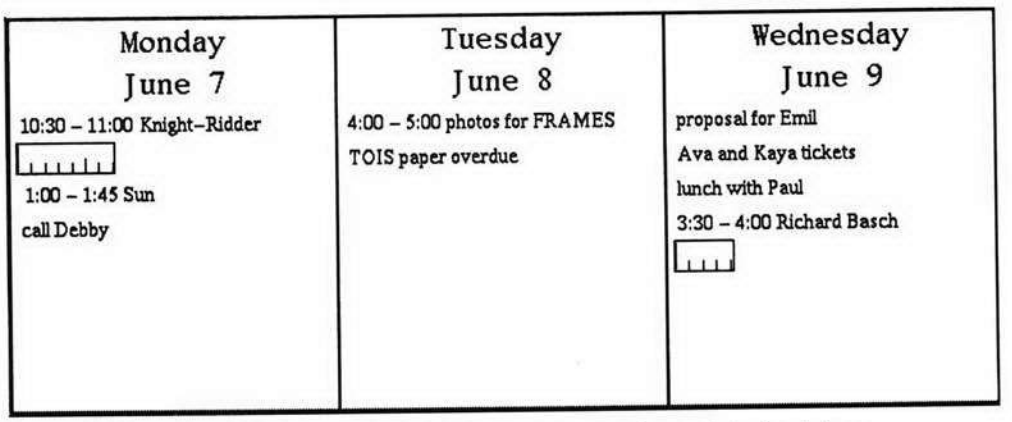


Figure 12.16. A personal calender containing voice and text entries.

applications; e.g., a user may select a portion of a voice message from a caller who will visit in a few days and paste that into the calendar as a reminder of the visit or to suggest topics for discussion during the visit.

Another application, ToDo, shown in Figure 12.7, combines voice and text in a things-to-do list. A user can enter voice or text notes while at the workstation, but voice notes more frequently originate from telephone interactions with Phoneshell. While accessing voice mail in PhoneShell, a user records a "memo"; this sound file is then incorporated into the ToDo database.

While each of these applications is fairly basic in isolation, it is their combination on a single workstation screen that renders them most powerful. In practice, little of the audio data used in these applications is recorded at the workstation; text is usually preferred simply because it is easier to retrieve at a later date. Instead, voice annotations arrive over the telephone into the databases used by these applications, either in the user's own voice via Phoneshell or as portions of voice messages from other people through audio cut and paste. Although none of

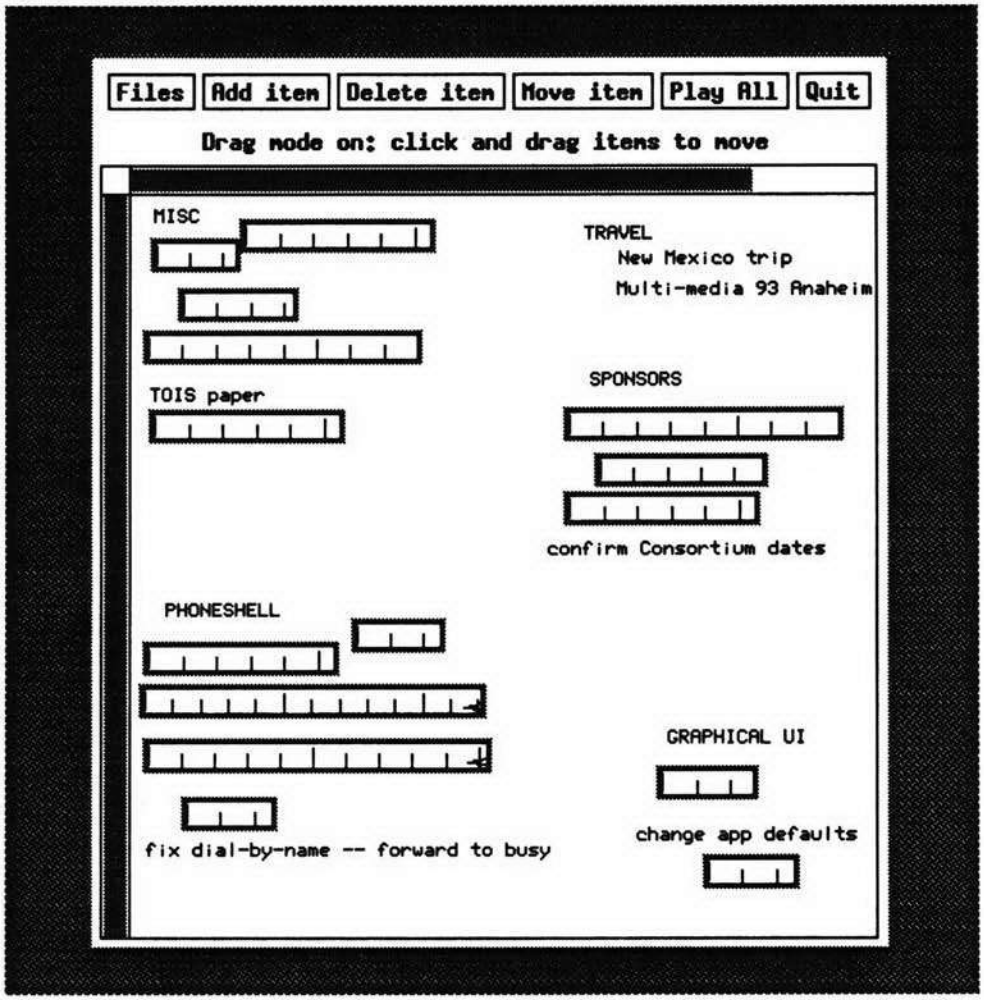


Figure 12.17. A personal project manager containing voice as well as text entries. Voice entries are most often recorded over the telephone using Phoneshell.

these applications is overwhelming in isolation, the appropriateness and ease with which they can be used in combination hints at the ultimate utility of desktop audio.

SUMMARY

This chapter builds on the previous chapter's point of view of unifying telephones and computers by presenting the concept of Desktop Audio, the integration of voice processing into screen-based desktop computing. It started by arguing that

the successful deployment of voice applications depends on a number of factors. The most important factor for success is that voice must add value to existing work practices, as it is unlikely that any single new application is going to radically change the ways in which office computers are being used. But it can enhance current productivity tools, e.g., by providing better user interfaces to current voice systems such as voice mail. Desktop voice processing can enable remote access over the telephone to desktop applications and their databases as was described in the case study on Phoneshell. The case study on Conversational Desktop was meant to illustrate this broad range of potential Desktop Audio functionality, and how it benefits from the integration of multiple voice processing technologies.

Desktop Audio requires graphical user interfaces to indicate the presence of stored voice within an application and to control its playback. Graphical interfaces can also support cut-and-paste of stored voice between applications. Common interfaces are the "button," which implements click-to-play while requiring little screen space, and more elaborate audio playback control panels which may also provide a temporal representation of of the stored voice. The SoundViewer, described as a case study, attempts to provide the enhanced functionality of control panels while minimizing its screen space requirements. The SoundViewer was shown in use in an array of applications discussed in the final case study.

Desktop Audio also benefits from a client-server architecture, which enables multiple applications to make use of digital audio resources simultaneously without interfering with each other. Audio toolkits should insulate clients from details of the audio server's protocol and provide convenient user interface building blocks to enable rapid application development. A number of server architectures were discussed in detail.

Desktop Audio has become a reality only very recently, driven primarily by the increased speeds of workstations. Issues with appropriateness of speech applications as well as the techniques to employ voice processing technologies effectively all come to bear on this integrated approach. With proper integration, voice can be used seamlessly across a range of tasks we perform in our daily work lives.