An Audio and Telephone Server for Multi-media Workstations

Chris Schmandt — Michael A. McKenna

Media Laboratory — Massachusetts Institute of Technology

Abstract

Workstation utility may be enhanced by the addition of *audio* and *telephony* functions for a number of applications, such as voice mail, multi-media documents, and computer conferencing. We outline the requirements of these applications, and present an *audio server* architecture employing a personal computer with a disk and speech-processing board as a server dedicated to the workstation. The functional interface between client and server is described, as well as our motivation for this specific server approach with particular emphasis on user interface considerations. We discuss our experience with this server architecture and contrast it with alternate architectures.

1 Introduction

This paper describes the design principles and architecture for a dedicated audio and telephone server to a general-purpose workstation. This work was done as part of a larger research agenda of exploring uses of audio, with particular emphasis on user interface, in applications such as voice mail and telephone messaging systems, conferencing, dictation, and audio document editing.

Audio and telephone services can be used for a variety of functions on the workstation. *Telephony* functions are the natural extensions of tools such as on-line directories of frequently called numbers. *Voice mail* and *telephone messaging* complement text-based electronic mail systems and may allow retrieval of messages from remote locations using ordinary telephones. *Remote access* to on-line databases over voice lines can be provided by DTMF (Touch-Tone) decoding and text-to-speech synthesis. Audio may be used for *documents*, including both dictation as well as multi-media documents; editing, transmission, and perusal capabilities are all required. *Conferencing* scenarios, involving multiple users on multiple workstations cooperating on a task, necessitate transmission and switching of audio. *Human-computer interfaces* may be improved by the use of natural dialog, achieved through the use of speech recognition and synthesis.

Our goal was to build a flexible and moderately inexpensive means of adding audio and telephone functions to our workstations. We desired to interface various voice capabilities (digital recording, speech recognition, speech synthesis) and the telephone to the ordinary software development environment in a manner that would allow rapid prototyping. Our design decisions were influenced by the need to build something realizable in the short run and extensible in the long run. Because of the computational resources available in our laboratory, we wished to be able to use audio functions from a number of different kinds of workstations, such as Suns, HP Bobcats, and Symbolics lisp machines. In addition, we wanted to maintain flexibility to incorporate new speech-processing hardware. We did *not* make any attempt to build the least expensive solution in terms of production costs, although we will discuss comparative costs of alternative solutions.

Our approach to audio/telephone service is an IBM PC-based server, utilizing various commercial speech and telephone interface cards. The server and workstation communicate over a serial line. The server is dedicated to a single workstation and is assumed to be in close proximity to the microphone and speakers. Longer distance audio transmission is done using ordinary switched analog telephone lines. A high-level command structure was designed to isolate the workstation from audio-processing and storage requirements. The audio file system resides on the server. It has been imperative to provide efficient low latency real time audio operations; the server architecture must not interfere with responsiveness.

In the following sections we will outline the service requirements of these various applications of audio and present a server architecture and functional interface to the client designed to satisfy these requirements. We will indicate our motivations for this architecture and for the use of standard telephone lines for audio transport, especially during conferencing. We will also discuss a number of alternate server architectures.

2 Service Requirements

The hardware, protocol and architecture of the audio component to be added to a workstation depends upon the applications for which the medium is to be used. This section will consider several classes of applications and their corresponding functional requirements. In the next section we will describe an implementation designed to serve all these applications; a partial implementation would be adequate for specific limited situations.

2.1 Conventional telephone applications

A number of applications seek to integrate telephones and workstations. These include placing outgoing calls from an on-line database (selected by menu or perhaps in response to a piece of electronic mail) and voice message functions such as answering

Proceedings, Second IEEE Conference on Computer Workstations, IEEE, Santa Clara, CA, 1988

machines of various levels of intelligence. A phone line interface is required to detect incoming and place outgoing calls.

Simple audio functions to provide recording and playback of sound files allow message taking to support answering machine or voice mail applications. Recording is facilitated if pause detection is provided to enable voice-activated variable-length messages (voice-activated switch). Lengths of sound files may be used for the sake of graphic display of pending messages. DTMF decoding may be useful to allow the workstation's owner access of messages from remote sites by phone, as with "beeperless remote" answering machines.

2.2 Remote access of text databases

Closely related to external access to audio databases, such as voice messages, is remote retrieval of text information from a workstation. Although conceptually similar, text access requires different audio capabilities and provides an interface to different types of data. Such exchanges may include electronic mail perusal [8,16] (particularly useful if an audio message can be recorded as a reply), database query, or order entry.

These functions again require the telephone line interface, ring detect, etc. DTMF decoding is required for input, and text-to-speech synthesis for output. Because speech output is slow, interruption by the user must always be possible. Local acronyms, jargon, and proper names may require a text-to-sound exceptions dictionary, which might be stored most conveniently in the synthesizer.

2.3 Audio documents

Audio may be used for documents, either alone or in a multimedia context including text, graphics, and image data types. Both authoring and perusal of such documents must be considered.

Audio documents obviously require recording and playback capabilities. In addition, especially for editing or annotation, it is useful to be able to play back or record smaller pieces of the entire sound. Both the editing and perusal software probably need to know the length of the audio segments for display purposes. Sound file editing functions must be provided, such as cut, paste, and replace [1,11,15]. These functions must incorporate the details of whatever speech compression algorithm may have been used to record the sound.

Transmission of audio documents may be done over analog phone lines or digital computer networks. In the latter case, transport protocols must be provided, but, as with electronic mail, this need not be real time. For analog transmission, some handshaking would be needed between sites, either through the computer network to set up the call, or perhaps using a modem or DTMF signals to preface the analog transmission with header information.

2.4 Computer conferencing

Computer conferencing is concerned with allowing multiple users of workstations to exchange information and work together, with some level of connectivity maintained between workstations during the conference [6,7,10,21,22]. These conferences may be multimedia, including video or facsimile links as well as audio, although the audio is probably the primary channel [12]. These conferences are real time shared workspace events, not discussions carried on with a number of parties through mail distribution lists.

In this environment the audio task consists of establishing a real time link between users at their workstations. This link may be analog or digital, and may use computer or telephone networks. Although some researchers argue for the need to synchronize audio with other media being transmitted (video, mouse motion, screen display), we believe this is either unnecessary (in the event that all media are being transmitted in real time) or counter-productive (there is no need to always slow down *all* channels to the *slowest* channel, as humans will compensate with appropriate behaviors when needed). For this reason, we have opted to use the telephone network, accessed and routed through the phone line interface.

3 Server Architecture

Our implementation of an audio/telephone system to handle the above service requirements consists of a PC-based server using appropriate speech and telephone interface boards and communicating with its client over a 9600 baud asynchronous serial interface (see figure 1). The server is dedicated to a single workstation, so its audio is routed to that workstation's location. The server has its own disk on which to store speech files, as well as other information such as user templates for speech recognition.

The server is a standard 5 MHz IBM XT. We have used speech cards from Dialogic (phone interface and voice digitization with ADPCM) and Texas Instruments (optional phone interface, digitization using a variety of compression algorithms, speech recognition, and text-to-speech). Server software is written in C with assembly language as needed and runs under MSDOS with an added interrupt-driven serial I/O handler. Client software has been written for Sun workstations and, partially, for Symbolics computers. One of our former research computers, a Perkin Elmer 3230 running a homemade operating system (Magic 6), was also configured as a high-end server for some applications.

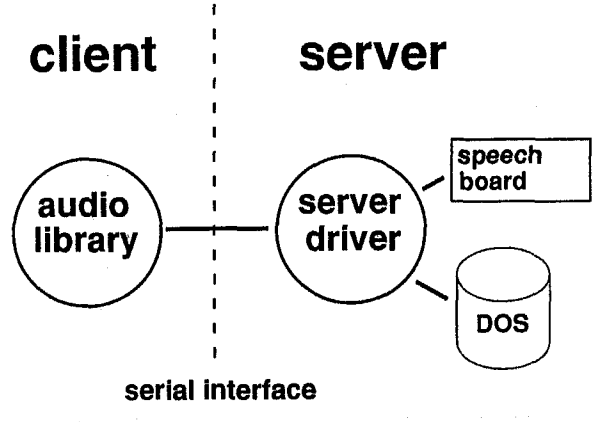

Figure 1: The basic server-client architecture

The protocol between client and server is designed to minimize communication requirements and hide device-dependent details of server configurations. Most commands are simple byte-oriented command/acknowledge sequences, with arguments such as strings (file names) and integers (file lengths). A simple packet protocol provides for burst-oriented transfers such as uploading a sound file; the host specifies the block size depending on how well it can handle bursts of serial input.

The server supports the construct of an open *file* which may stay open for a series of operations. Portions of that file may reside in memory temporarily as a *buffer*. The server may support multiple *channels*, each associated with a particular hardware component (board, phone line, A/D). However, channels may also be *virtual*, sharing hardware and independent at the software level only. Virtual channels each maintain their own buffer in memory, as well a descriptor specifying an open file. Events defined on virtual channels may be queued to minimize latency (see below) or to support modularity in client software.

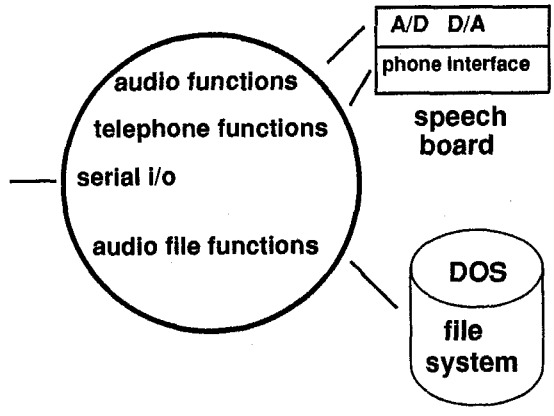

Figure 2: Server functions and components

4 Server Functions

The server supports a number of functions for audio and telephone service at various levels (see figure 2). This section describes those functions including several improvements in progress or at the planning stage at the time of this writing. Note that each function will be implemented as completely as possible for a particular hardware device. For example, independent functions are provided to control playback and recording amplification; on some boards these may be identical, while on others they may have no effect at all.

4.1 Audio functions

Simple audio functions: Simple audio control begins with PLAY(*sound_name*), RECORD(*sound_name*), PAUSE, CONTINUE, and HALT. These regulate the flow of data between files and sound processing hardware such as digital/analog converters. Amplification control is provided through VOLUME (*value*) for playback and GAIN (*value*) for recording. CALIBRATE allows the server to set gain appropriately on devices with Automatic Gain Control (AGC). SPEED (*value*) controls the rate of playback. METHOD (*value*) selects a speech rate or coding algorithm for recording, allowing the host control over the disk space/voice quality tradeoff where applicable (not yet implemented).

Audio status functions: Several functions provide access to status information about sounds and the current audio-processing state. GET_LENGTH (*sound_name*) returns the length of the sound; all times are in milliseconds. DONE returns a flag indicating whether the current play/record operation has finished. WHERE returns a value indicating how many milliseconds of the current sound have been played or recorded; in conjunction with GET_LENGTH this can be used for graphic displays showing progress while the sound is played [15,17]. DURATION (*value*) sets the maximum time to record a sound if a HALT or PAUSE command is not received.

Specialized audio functions: Several specialized functions have been written for particular applications. One set of functions allows control over automatic pause detection and silence removal during recording, as in voice-activated tape recorders. PAUSE_DETECTION (*flag*) enables pause detection. PAUSE-VALUES (*start_value*, *stop_value*) defines the length of two time periods. The first period is the initial timeout: if audio is not detected in this time period, recording is aborted. The second value defines the length of continuous silence that terminates recording. CLIP removes the leading and trailing silences from sounds recorded with pause-detection enabled; this saves disk space and allows a crisper user interface by avoiding playback of silence later.

Some audio-processing functions provide specific functions used for acoustic analysis in speech research or for displays for audio editors (see section 5.3). The functions may be applied to portions of an audio data file, with their results sent to the host. MAGNITUDE returns average magnitude, ENERGY returns power, and PITCH returns fundamental frequency.

UPLOAD_SOUND(*sound_file*) and DOWNLOAD_SOUND(*sound-file*) functions must be provided for the sake of transmitting an audio file digitally over the computer network. They use the packet transfer mode to move the actual sound data between the client and server. Of course, a downloaded file will not be playable on a server that does not support the same speech coding algorithm as that used when the sound was originally recorded. Note that we do not use UPLOAD_SOUND for audio editing; this is instead supported at the server level (see 'Audio editing functions' below) to isolate the client from knowledge about the details of speech coding.

Sound file system functions: Sounds are stored in the file system of the server, and although details of that implementation are not needed by the client, some general file system functions are required. NAME (*sound_name*) opens a sound file for some series of operations, such as editing. ALLOCATE (*sound_name*, *value*) reserves space for a particular sound file, and was convenient for a particular audio file system that forced sound to occupy contiguous disk blocks for transfer efficiency. Standard file system commands include CHANGE_DIRECTORY (*directory*), CREATE(*sound_name*), REMOVE(*sound_name*), and COPY(*source sound_name*, *destination sound_name*). A DIRECTORY function to get a list of files in the current directory is currently being implemented, as are MAKE_DIRECTORY and REMOVE_DIRECTORY.

Audio editing functions: Sound files are stored on the server, and we desire to avoid uploading them to the host. Some audio editing functions are provided for on the server, to be controlled by user interfaces written on the client. LOAD_BUFFER reads all or part of a file into a specified section of a memory buffer. SAVE_BUFFER writes all or part of a memory buffer to a file. COPY_BUFFER writes a section of one buffer to another. SET_BUFFER_LENGTH allocates memory for buffers. Higher-level operations such as *cut* and *paste* are built from these protocol primitives.

In order to interactively edit sounds, it is necessary to play segments of larger sound files. START (*value*) and STOP (*value*) set the time bounds for PLAY operations. Another aid to editing is a visual display. The MAGNITUDE and PITCH and ENERGY commands discussed previously can be used to obtain data for graphic displays (see section 5.3).

4.2 Telephony functions

Outgoing call functions: A simple set of functions controls the placement of outgoing phone calls. HOOK takes the phone on or off hook to initiate or terminate a call. DIAL (*number_string*) causes a number to be dialed, typically with DTMF signalling. CALL_PROGRESS returns the current call completion status (ringback, busy, dialtone, "answer", etc.). The sophistication of this information depends on the level of signal processing supported by the board and may also depend on characteristics of the local PBX.

Incoming call functions: Incoming calls need to be detected and answered, and for interactive functions, DTMF digits need to be decoded. Both non-blocking (poll) and blocking (sleep) versions are provided for flexibility. POLL_RING and WAIT_RING detect ring, while POLL_TONE and WAIT_TONE return the value of a single DTMF digit if one occurs. The HOOK function described above answers the phone.

Audio routing functions: For some functions, such as conferencing or hands-free phone operation, one wishes to connect the phone to the local audio environment using microphones and speakers. In other cases, such as an answering machine, it is appropriate to disable the microphone and probably the speaker as well, except for call screening. The MONITOR function sets flags to enable microphone and/or speaker connection to the phone line. Some boards do not allow these to be switched independently.

4.3 Speech recognition and synthesis functions

We are expanding the audio server functions to support speech recognition and synthesis. More advanced speech and phone boards are built around digital signal processor components, such as the Texas Instruments TMS320 family of microcomputers. In addition to speech coding and advanced call progress functions, these boards may support recognition and synthesis algorithms. In such cases the server should support these as well.

Speech recognition: The local disk on the dedicated server is especially useful for speech recognition, which involves training and storing vocabulary reference templates. If high-speed transfer between word template storage and recognizer memory is available, dynamic vocabularies may be implemented. Templates may be changed in real time as a function of user input to provide recognition for a larger number of words [20]. The key requirement is high-speed transfer between disk and recognizer memory.

DOWNLOAD_VOCABULARY(*vocabulary_name*) and UPLOAD-VOCABULARY (*vocabulary_name*) provide for transfer of vocabularies between disk and the recognizer memory. TRAIN (*word-number*) and UPDATE (*word_number*) provide for the initial training and later modification of previously trained templates; in some implementations these may be identical, while in others the latter may invoke a template-averaging function. RECOGNIZE causes the server to return a stream of word numbers as they are recognized until a halt command is received. RECOGNIZE_ALL returns a packet for each word, with device-specific recognition information such as second word guess and quality of the match. VOCAB_LENGTH reports how many words are in the currently loaded vocabulary.

Speech synthesis: Work is in progress to implement a set of functions providing access to speech synthesis (text-to-speech). In addition to entering synthesis mode and receiving a stream of text to synthesize, synthesizers generally also support functions to control speed, pitch, "voice" (speech style), stress, and phonemic input mode. We have not yet decided whether to support these as separate functions, or simply have the client embed them in the serial stream as escape sequences to remain compatible with commercial stand-alone synthesizers. In any event it is most useful at the *client* end to support functions such as SPEAKING_RATE and BASELINE_PITCH as separate entries. The client also has to know when the server has finished speaking text, to allow synchronization with other events.

4.4 Latency-Reducing Functions

Speed is a serious drawback of audio as a data medium, and responsiveness is needed for any good user interface. In isolating the workstation from audio operations, an additional delay is introduced into a play or record operation. As it has been essential in our work to minimize this delay, two levels of functionality have been added to the basic server audio operations to minimize latency.

At the lower level, latency may be reduced through PREPARE_PLAY and PREPARE_RECORD functions, which act on the currently specified sound files. PREPARE_PLAY causes the file to be opened, the associated memory buffer to be filled from disk, and hardware data buffers to be preloaded if possible. PREPARE_RECORD creates the file (if necessary), opens it, and allocates memory buffers. The associated CONTINUE_PLAY and CONTINUE_RECORD functions simply enable the board to start moving data between buffers and hardware. A number of independent channels can be prepared and then played in sequence or as needed.

This suggests the next layer of latency reduction, that of *queues*. To play two sounds consecutively without queues, one plays the first one, prepares to play the second one, then sits in a loop polling the server until the first play is finished, at which point the second play is continued. But even with the second file already open and preloaded into memory, we have found that the latency for the client to detect completion of the asynchronous play event on the server is excessive under Unix^{TM}. The so-

lution is to set up a series of virtual channels and use QUEUE (*channel_number*) to establish an ordered series of events, which may be combinations of plays and records (with pause detection or maximum length completion). The DONE command is then specific to a given channel, and can be used at any time to tell which events have been completed. New events may be added to the queue while in the process of performing currently loaded events, and the queue may be flushed (in its entirety) with the HALT command.

The combination of advance preparation and queues of server events allows a series of audio events to occur with a minimal initial delay and at the maximum throughput provided by a particular server configuration. Queues are simple linear sequences with no conditionals, branching, or waits for input. Events may be appended to the queue while queue processing takes place, but may be removed only by flushing the entire queue. The workstation should provide the appropriate environment for more sophisticated audio operations, especially as queues free it from repetitive polling at a play-by-play level.

5 Design Considerations

We were driven to build the server described in this paper by our need to add audio and telephony functions to workstations for the sake of building interactive speech systems. We faced a number of design questions: whether to add a speech board to the workstation or whether to use a dedicated but separate server, how to route real time audio, and where to locate the audio file system. Our PC based server was chosen not because of a theoretical analysis which indicated it was the best choice, but because of practical constraints of cost, required flexibility, and the possibility of gradual development and refinement over time to suit the needs of different projects.

The reason this paper is being published is that our design proved workable and we desired to share our experiences with others facing similar decisions, so in this section we attempt to justify some of our choices, including the motivation for a server and reasons for our particular server architecture. We will also discuss some telephony issues, as well as some important design constraints in terms of division of functionality and data structures between the client and server. In the following section, we will briefly mention some alternative server architectures which should be considered in evaluating our design in one's own context.

5.1 Motivation for a Server and Our Architecture

To integrate audio with the workstation, we had two main options: a signal processing board as part of the workstation or a server which is dedicated to a single workstation, and contains a speech board and possibly its own local disk. In this section we present our reasons for choosing a server approach, and discuss some of the reasons for choosing our particular PC based server architecture.

Audio as a data medium requires either lots of real time data compression or lots of storage. The microprocessors employed in workstations are poorly suited for intensive digital signal processing, which can be done much more efficiently by specialized microcomputers designed for executing these tasks. Diskless networked workstations are becoming more popular as a cost-effective solution to large operating systems, but moving large audio files between such nodes and their disk servers with low latency can be difficult, especially when other data is moving over the network.

Audio record and playback functions are inherently real time, with necessity to keep a constant flow of data between hardware digital/analog converters, buffers in memory, and files stored on disk. Pause detection for voice-driven recording requires real time access to audio data to detect energy levels and time pause durations. Unfortunately, a good real time operating system may well be a poor software development environment. Unix with all its tools is typically ill-suited for sustained low latency I/O. A separate server allows the best of both worlds, and leaves enough free cycles on the workstation to support a sophisticated yet responsive user interface.

Having decided to work with servers, a PC-based, dedicated system with a local disk allowed rapid and flexible development. As the servers are based on off-the-shelf hardware we have saved considerable development costs and were allowed to begin work on our real research agenda into the use of audio at a much earlier stage than if we had built our own boards to run directly on the workstations. In fact, client and server software was (and continues to be!) developed as new tasks require enhanced functionality.

The PC-based server allows us optimal hardware and software flexibility. On the hardware side, there are a large number of commercial products for the PC bus supporting telephone interfaces, signal processing, speech recognition, speech synthesis, and any number of data compression algorithms. On the software side, the ability to write code at a low level to communicate directly with these boards allows us to get the most performance out of the hardware while hiding its specifics in the server software; this may well provide advantages over interfacing directly to stand-alone speech devices [20].

We decided to use a local disk on the PC, as opposed to a shared file server for the PCs or the PCs and workstations, largely for simplicity and some fear of saturating a file server with audio file access. The latter was especially true as we anticipated storing a variety of data in different formats, such as speech recognition template files and voice files using various data compression schemes. We also wanted to be able to accurately account for delays at various levels under best and worst case conditions, and a networked file systems adds a layer of complexity to these calculations.

For the most part, the client does not need direct access to the audio file system. It is important to be able to obtain some information about sound files, such as lengths and data compression modes for files and a directory listing of sound names. Audio editing functions can be provided on the server, which can compensate for whatever audio coding was employed while applying delete and insert operations, and thus avoiding uploading large amounts of data to the client. As we use analog telephone lines for real time audio transmission, uploading is necessary only for non real time applications like mail and document transmission, so server-client bandwidth is not critical.

5.2 Use of Telephone Lines for Audio Transport

For some of the applications using audio on workstations, par-

ticularly conferencing and sending audio documents, speech data must be transmitted between multiple sites. This may be done over packet networks [2,9,23,25] or ordinary telephone lines. Our approach uses phone lines because they are inexpensive, readily available, and guaranteed to be real time.

Analog audio transmission over the switched telephone network is easily implemented and inexpensive. The telephone system is an efficient network for voice transport. To support the range of applications required for integrating the workstation into daily work life, a telephone interface is essential anyway, if only for adding dialing ability to an on-line directory. We believe that other telephone functions, such as voice message systems and remote access of databases, are also desirable.

Audio document transmission may be done either as analog over phone lines or digitally by uploading files from the server and sending them as ordinary electronic mail over local or long-haul computer networks, to be downloaded to the remote server from the remote workstation. The latter may be preferable to avoid degrading the speech by repeated conversion between digital and analog, especially as this transmission (again, like mail) need not be real time. However, digital transmission is possible only if both sites support the same speech compression or coding algorithms, or easy conversion is possible (such as between ADPCM and simple PCM). In the long run we can expect an all-digital telephone system such as the evolving ISDN (Integrated Services Digital Network) standard, which will solve some quality problems. There is some effort at the international standards committee level to utilize more recent speech coding techniques to allow higher audio bandwidth (6 kHz) phone connections, especially for conferencing.

For conferencing applications speech must be transported in real time; transmission delays can seriously undermine the interactive nature of the conference, which is exactly why one would choose to conference instead of sending a message. Packet networks are ill-suited for audio transmission, unless they support specialized stream protocols [5,9]. Audio requires packet transfer in a bounded amount of time (if a packet arrives late or out of sequence it is useless), but can tolerate a certain amount of error and still remain intelligible. Where standard Ethernet has been used for audio transport in a workstation environment, a separate net has usually been employed [3,23,25].

5.3 User Interface Considerations

The separation between server and client is in many ways attractive for modular software development and the flexibility of hardware substitution to support a specific application. However, this separation impinges on the user interface in some undesirable ways, mostly by inserting another layer of delay between the application and the actual audio events.

Speech is a difficult medium at the user interface; it is slow (typically on the order of 100 baud in text equivalent) and serial in nature (as opposed to *text* menus, which stay on the screen until a selection is made). To compensate, interactive voice systems require low latency audio events and interruptablity by the user. As an interface, sentences may need to be pieced together from portions of pre-recorded audio, as "You have..." "...four..." "...new messages". If an answering machine asks a question, it needs to be able to start recording the response immediately, or part of it might be lost. The overhead of opening files and preloading hardware data buffers may prevent these sort of operations.

However, some aspects of the speech channel work in our favor. A voice mail system, for example, is mostly idle, waiting for an expected event such as a phone ring or a DTMF signal, so it has plenty of time to prepare for the next event or series of events. Even while playing pieces of sounds, the very slowness of speech allows an ample amount of time to *prepare* for the next event while the current event (e.g., playing a greeting) is underway. This allows us to benefit by the PREPARE_PLAY, PREPARE_RECORD, and QUEUE functions. As long as the actual sound files are not too short, the server can keep up with an arbitrarily long sequence of events in real time.

It is also important to note that speech output, because of its speed, should almost always be interruptible. This implies that while playing, recording, or synthesizing speech, the server needs to monitor the phone line (if any) for DTMF tones and the communications port for fresh commands from the client.

The real benefit of the server approach for the user interface is that it frees the workstation from the demands of keeping up with audio data moves between memory, disk, and speech hardware, leaving cycles free for sophisticated interfaces and such amenities as window systems. We offer several examples from our work.

The *Conversational Desktop* [18,19] explored speech as an interface and data medium in a network of workstations. It was built around graphical and natural language interfaces and utilized speech recognition, synthesis, audio record/playback, and telephone management hardware. The latter two functions were implemented in the server. Besides offering interfaces and agents on Suns for scheduling and remote database access, the Desktop could record event-triggered audio memoranda, place and take calls, and record phone messages. Messages were taken using a conversational method previously employed in *Phone Slave* [17]; the necessity of a series of closely joined record and play operations caused us to implement audio queues.

Pitchtool is an audio analysis and editing tool used for research into human speech prosody (rhythm and meter). Pitchtool uses the server for record and playback functions, employing Linear Predictive Coding as it provides convenient access to pitch and energy values. The user interface is implemented in the Suntools window environment (see figure 3) and allows a researcher to display pitch and energy values and modify them with a mouse. Pitchtool also supports editing functions (developed from server primitives) that are convenient for extracting interesting portions of conversation from long dialogs.

Recently a modified version of the basic server has been developed as an audio-processor for work studying human "back channels" during telephone access of driving directions using text-to-speech synthesis [14]. While the computer is reciting directions, the listener can indicate comprehension, need for more information, or requests for more time. We are attempting to classify these discourse events acoustically by duration (and ultimately pitch contour) to decode the back channel without using word recognition. This interaction requires close synchronization of speech events between the client and the server. The server was modified to allow the record function to return to the client real time notification of the beginning and duration of speech input

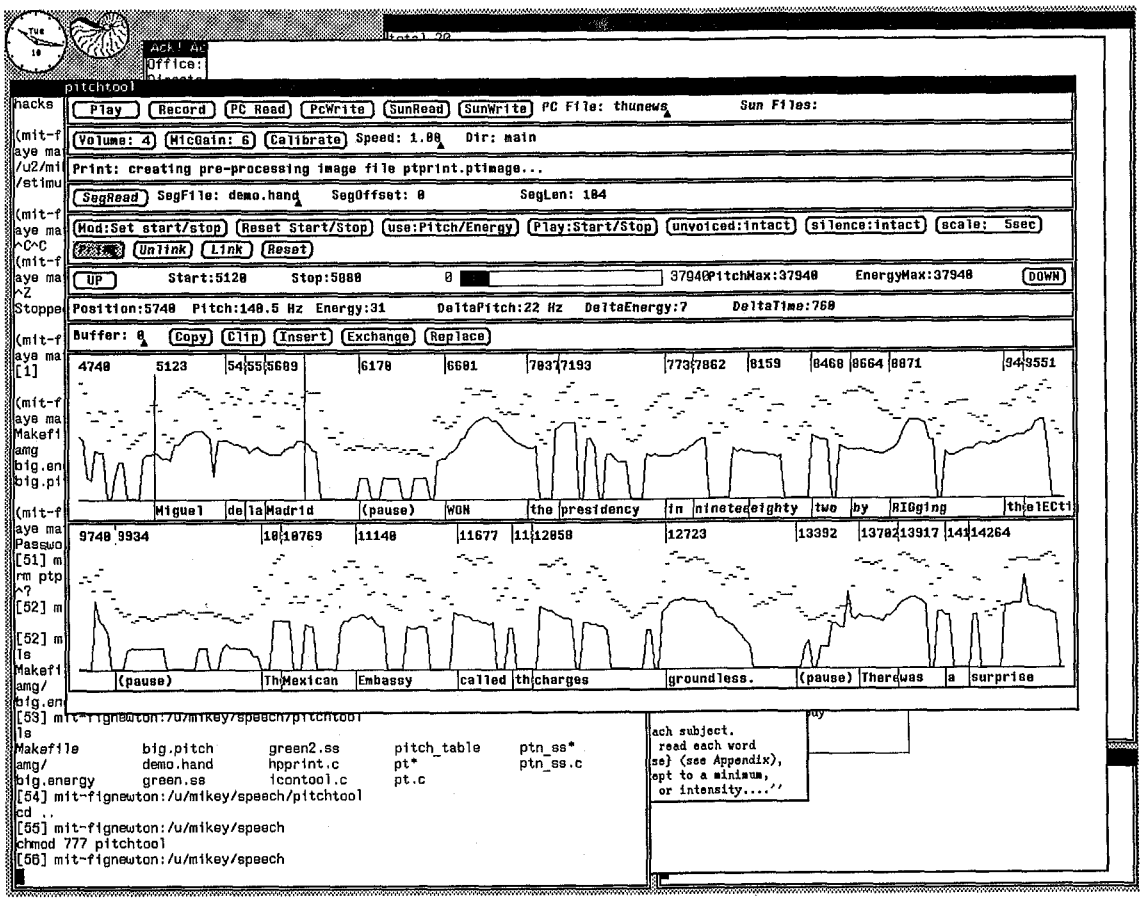

Figure 3: The Pitchtool Interface

when detected by the existing pause detection and average magnitude functions.

6 Alternative Server Architectures

In this section, we consider some alternative server architectures and briefly discuss their merits. A longer paper would be required to discuss them in adequate detail. It is our belief that with all servers, a *functional* interface similar to that described above would be useful.

6.1 Board level audio on the workstation

The entire server described above could be built as a single board to plug directly into a given workstation, using Multibus or VME bus or whatever one's favorite workstation supports. This board would include a telephone line interface, digital/analog converters (codec), probably a digital signal processing microcomputer to perform tasks such as speech compression, and an independent microprocessor to implement server functions such as editing, speech synthesis, and communications. The server disk would be replaced by the workstation's disk or file server. The board would include a DMA interface to the disk controller to minimize impact on the workstation processor running the user application.

This arrangement is logically similar to our server architecture, with the serial interface replaced by the workstation bus and the PC speech/phone board replaced by the one in the workstation, with control software in the on-board microprocessor. Many latency issues may be reduced, as would cost in a production version. Other tradeoffs have already been discussed.

The main disadvantage of this approach is the load it may place on the workstation's microprocessor or file server during record or playback, which may interfere with other processes or workstations and detract from the user interface. Careful design of such a board is required and without experience building audio interfaces it would be difficult to specify correctly.

6.2 Dedicated servers using Ethernet

Another approach to audio service is identical to the one described here with the substitution of Ethernet (or equivalent LAN) for the serial connection between workstation and server; the server stills stores the audio information. The attractiveness of this is largely a matter of taste. Except for uploading sound files, the protocol between workstation and server consists of a few bytes at a time, and packetization costs probably make up for the somewhat slower serial interface. Ethernet may be cheaper to install, although network interface cards are more expensive than serial cards for PCs. Network interfaces would support error-free versions of our protocols more easily.

A variation on this theme would be for the servers on the network to also use the network file system supporting the workstations, thus realizing economies on disk storage. It is unclear whether a network file system would adequately support the bursts and large amounts of audio data required for sound reproduction. On the other hand, this approach allows the workstation to access the sound files easily, as a common disk server is employed.

6.3 Centralized servers with packet network audio

Another approach to audio service is a centralized audio server, with local, addressable network interfaces to convert between audio and packets [2,23,25]. The central server may be optimized to deal with audio, and addressed by workstations over the same or another network with commands such as "play file xxxx to address yyyy". The telephone interface may be local, next to the workstation and addressed over the network, or also centralized with all audio digitized and packetized.

The Xerox Etherphone project employs a dedicated Ethernet to transport audio data to the audio-processors in offices. The centralized server is optimized for an audio file system. Text-to-speech synthesis is also available from a server on the network using commercial synthesizers and a packet audio connection to the workstation. Performance is subject to standard CSMA/CD network considerations. It works well up to a saturation level where failure is rapid, but it may be a problem for busy networks where transmission time is not bounded because of collision backoff strategies, although this limit is not reached in normal use.

The AT&T Rapport computer conferencing system [3] also uses a separate, dedicated Ethernet to carry voice between special audio-processing boards on the workstations. It is not clear whether there is any audio storage in this system.

The Island project at Cambridge uses a slotted ring architecture [9], which guarantees a bounded transmission time. Multiple media are allowed on the same network, and specialized processors ("translators") deal with media-specific issues. Telephone services are centralized, and the PBX is accessed over the network. There is currently work in progress [24] to utilize higher bandwidth portions of the ISDN telephone system to link rings at remote sites, with gateways between ISDN lines and local rings.

6.4 Centralized, based on switched phone lines

A similar but more centralized architecture may emphasize switched telephone lines, in that both the local PBX *and* the phone lines to local workstations are part of the server [8,13]. The switch and any associated audio hardware are controlled over the network by workstations. The switch can route audio from devices to any office, as well as perform a full range of telephony functions. Although the cited examples have been designed primarily as sophisticated programmable telephone switches with network interfaces to workstations, their utility is being enhanced by the addition of speech synthesis and audio record/playback resulting in a functionality similar to our servers.

There is a trend among commercial PBX manufacturers to add features like voice mail to the central switch. For now, these switches generally lack interfaces to computer networks, and their instrument interfaces are considered proprietary, but it is possible to build a PC-based interface between the PBX one side and the computer network on the other, to obtain a limited range of networked functionality. At the University of Southern California Information Sciences Institute, a Rolm switch has been so interfaced to a networked PC [4], as a quick means of adding telephone functions to the workstation.

7 Future Work

Several areas remain to be worked out in the work described for our dedicated audio servers. The relationship between channels and queues should be improved upon to support multiple boards, each of which may be considered a channel. With support for multiple boards, we need a channel request function; ideally the client requests a channel by specifying desired classes of functions (e.g., telephone, audio playback, LPC analysis) and is returned an appropriate channel number.

Our speech synthesis interface functions are not yet complete. Various subsets of the remaining functions have been implemented on several speech boards as applications required them; no implementation is complete.

Our protocol assumes error-free byte-level transmission between client and server. In fact this has been the case for our site. In the case of byte lossage (typically, serial buffer overflow on the client due to delayed interrupt service), error recovery is possible but not guaranteed. Packet transmission error detection is limited to byte count, not corruption. Although not in fact a problem in our installations, these should ideally be fixed.

The client at the moment has no means to select from a number of possible recording qualities supported by most boards, or to query the server as to the recording method used for a particular sound. Control over low level telephony signalling parameters may be useful. These might include duration of DTMF signals, DTMF silence intervals, and hook-flash duration. PBX-specific call progress detection may be needed for some environments.

8 Conclusion

We have described an audio and telephony server designed to add these media to a workstation environment. This approach was dictated by our research agenda and resources. We needed to get basic functionality quickly in an environment rich in good programmers. We wanted to be able to use the server from a variety of workstations, and over time needed to support a variety of functionality in the server requiring several different speech boards.

The most difficult aspect of building these servers has been dealing with the specific limitations of each speech-processing board and missing functionality in vendor-supplied device drivers. This has been generally offset by the low cost of the boards and speed of software development when porting server software to new hardware. We have been able to manage the tradeoff between correct functionality and the time needed to design systems from scratch by the separation between client and server, aiming for the maximum functionality for the client sound library and implementing it as best as possible on the server.

We have built a range of fairly powerful tools and applications around the server architecture. The set of functions de-

scribed herein has grown as needed for new applications and will probably continue to expand to meet new needs. At this point, however, it is easy to prototype many applications on the workstation using the existing functions.

The most obvious limitation to working with PCs is speed. They are slow for software development and slow for real time applications, especially using DOS for file system access. On the other hand, faster PCs are available and getting quite inexpensive and, once code for a server is finished, little more work is needed on it.

There is still work to be done. Some functions are incomplete. Some board specific features are lost in the generality of the interface. There are several cases where the server software needs to be sped up. But in general this server architecture has proven quite adequate for our fairly demanding needs.

9 Acknowledgments

Much of the motivation for writing this paper and some of the ideas herein came from discussions with other members of the User Interface Task Force of the Distributed Systems Activities Board. Polle Zellweger of Xerox PARC offered numerous valuable comments on the paper content. In addition to the authors, David Chen, Jim Davis, Lorne Berman, Phil Sanborn, and Elsa Chen worked on various portions of the client and server code.

This work was supported in part by DARPA, Space and Naval Warfare Systems Command, under contract number N00039-89-C-0406 and by NTT, the Nippon Telegraph and Telephone Public Corporation. Hardware support was provided by Sun Microsystems.

References

[1] S. Ades and D. C. Swinehart. Voice annotation and editing in a workstation environment. In *Proceedings of the AVIOS '86 Voice Input/Output Systems Applications. Conf.*, pages 13–28, Alexandria, VA, September 1986.

[2] S. Ades, R. Want, and R. Calnan. Protocols for real time voice communication on a packet local network. In *International Conference on Communications*, pages 525–530, Toronto, Canada, June 1986.

[3] S. R. Ahuja, J. R. Ensor, and D. N. Horn. The Rapport multimedia conferencing system. In *Proceedings of the Conference on Office Information Systems*, ACM-SIGOIS, 1988.

[4] S. Casner. Personal communication, 1987. University of Southern California, Information Sciences Institute.

[5] J. W. Forgie. *ST - A Proposed Internet Stream Protocol.* Technical Report Internet Engineering Note No. 119, M. I. T. Lincoln Laboratory, September 1979.

[6] H. C. Forsdick. Explorations into real-time multimedia conferencing. In *Proceedings Second International Symposium on Computer Message Systems, IFIP Technical Committee on Data Communications*, Washington, D.C., September 1985.

[7] J. Garcia-Luna, A. Poggio, and D. Elliot. *Research into Multimedia Message System Architecture.* Technical Report 5363, SRI, February 1984.

[8] 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.

[9] A. Hopper and R. M. Needham. *The Cambridge Fast Ring Network System (CFR).* Technical Report Technical Report No. 90, University of Cambridge Computer Laboratory, June 1986.

[10] K. A. Lantz. An experiment in integrated multimedia conferencing. In *Proceedings CSCW '86: Conference on Computer-Supported Cooperative Work ACM and the Microelectronics and Computer Technology Corporation*, pages 267–275, December 1986.

[11] N. Maxemchuk. An experimental speech storage and editing facility. *Bell System Technical Journal*, 1383–1395, October 1980.

[12] R. B. Ochsman and A. Chapanis. The effects of ten communication modes on the behavior of teams during co-operative problem solving. *Int. J. Man-Machine Studies*, 6:579–619, 1974.

[13] B. E. Redman. *A User Programmable Telephone Switch.* Technical Report TM-ARH-009118, Bellcore, 1987.

[14] C. Schmandt. Employing voice back channels to facilitate audio document retrieval. In *Proceedings of the Conference on Office Information Systems*, ACM-SIGOIS, 1988.

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

[16] C. Schmandt. Speech synthesis gives voiced access to an electronic mail system. *Speech Technology*, 66–68, August/September 1984.

[17] C. Schmandt and B. Arons. A conversational telephone messaging system. *IEEE Trans. on Consumer Electr.*, CE-30(3):xxi–xxiv, 1984.

[18] C. Schmandt and B. Arons. A robust parser and dialog generator for a conversational office system. In *Proceedings of the AVIOS '86 Voice Input/Output Systems Applications. Conf.*, 1986.

[19] C. Schmandt, B. Arons, and C. Simmons. Voice interaction in an integrated office and telecommunications environment. In *Proceedings of the AVIOS '85 Voice Input/Output Systems Applications. Conf.*, 1985.

[20] C. Schmandt and W. Bender. A programmable virtual vocabulary speech processing peripheral. In *Proceedings Voice Data Entry Systems Applications Conference, American Voice Input/Output Society*, 1983.

[21] M. Stefik, D. Bobrow, S. Lanning, and D. Tatar. Wysiwis revised: early experiences with multi-user interfaces. In *Proceedings CSCW '86: Conference on Computer-Supported*

Cooperative Work, ACM and the Microelectronics and Computer Technology Corporation, December 1986.

[22] M. Stefik, G. Foster, D. G. Bobrow, K. Kahn, S. Lanning, and L. Suchman. Beyond the chalkboard: computer support for collaboration and problem solving in meetings. *Communications of the ACM*, 30(1), January 1987.

[23] D. Swinehart, L. Stewart, and S. Ornstein. Adding voice to an office computer network. In *Proceedings of GlobeCom '83, IEEE Communications Society Conference*, pages 392–402, November 1983.

[24] D. L. Tennenhouse, I. M. Leslie, R. M. Needham, J. W. Burren, C. J. Adams, and C. S. Cooper. Exploiting wideband ISDN: the UNISON exchange. In *International Conference on Communications*, San Franciso, CA, April 1987.

[25] 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*, Santa Clara, CA, March 1988.