6th Symposium on Human Interface
Oct. 24–26, 1990 Tokyo

VOICE AND WINDOW SYSTEMS: SOME USER-INTERFACE CONSIDERATIONS

Chris SCHMANDT
Speech Research Group — Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Abstract. There is growing interest in voice as a data type for computer workstations. But, with window systems already quite ubiquitous in such workstations, we must consider the role of voice with respect to the window system. This paper covers three areas of intersection between speech processing and windows. The first is the use of speech recognition for navigation between windows. The second is user interaction with and visual representations of recorded voice. The third is transfer of multi-media data between applications using mechanisms provided by the window system.

Keywords. Speech recognition, window systems, multi-media.

Introduction

As workstations become more powerful, greater emphasis will be placed on multi-media systems. This is leading to growing interest in use of voice on the desktop. Voice makes attractive the user interfaces. Voice is key in communication, and hence, is at the root of all collaborative (CSCW) systems. Finally, as the digital telephone network (ISDN) grows, we expect stronger links between computation and telephony.

Speech is portable; we need no technology to use it among ourselves. We can use speech while walking down the corridor, and, with a telephone, we can use it from any remote location. Speech is natural; we learn it easily as small children. Speech is also very rich, conveying through intonation vastly more information than a simple transcription of speech would.

Speech technologies have various applications. Speech recognition may be used for data input (i.e., the proverbial "listening typewriter"), or as a command channel for user interaction. Speech synthesis may allow remote voice access to computer systems, including reading of human-authored text, such as electronic mail. Digital recording of speech provides voice as a data type, and applications such as voice mail and voice annotation of text.

At this time, speech coding is the most advanced of these three technologies; and, it is likely that within several years every workstation may support digitized speech. This is, however, no guarantee of its acceptance, as speech is a difficult medium with which to work. Speech is slow; several hundred words per minute would not be tolerated by modem users, in comparison to the speed to which they are accustomed. Speech is serial; an audio menu can be perused only by listening to each item in sequence, while a visual menu allows more random access via the user's wandering eyes. Finally, speech is "bulky"; we cannot perform key word searches with speech, making indexing difficult.

These difficulties with digitized speech conflict with the goal of voice as a ubiquitous data type, one which may be edited, browsed, transferred between applications, and presented with a consistent user interface. The last two requirements mentioned are exactly the concerns that window systems attempt to address. Window systems are ubiquitous; they give users the ability to perform a number of tasks in parallel, provide a programming substrate upon which direct manipulation interfaces can be built, and through style guides and inter-application communication mechanisms, provide a coherent whole within which applications co-exist. For voice to be successful, it must be integrated into window systems at the application and user interface levels.[1]

This paper discusses three examples of interactions between voice and window systems, including the description of a particular implementation of each under the X Window System.

Speech Recognition for Navigation

Much of the current work on speech recognition focuses on voice input replacing the keyboard. Indeed, the "listening typewriter" seems to be the "Holy Grail" of much commercial speech recognition work. The listening typewriter requires both a large vocabulary and a speaker adaptive recognition system, which must employ auxiliary sources of knowledge such as syntax, task constraints, or word order probabilities. Current speech recognition systems simply are not capable of this task; recognition accuracy is not adequate for most text input. What then is the role of speech recognition in computer workstations?

At the Media Laboratory, we have chosen to focus on speech as a means to augment the mouse, by now a standard workstation component. We hope to take advantage of the boost in user performance, which the introduction of a second input channel affords, by distributing user input across the multiple channels of voice and fingers (keyboard). Experimental evidence suggests that when subjects perform multiple tasks on multiple input channels, their performance improves over the case in which the same tasks are performed using a single input modality [8, 7, 1]. Because both the mouse and keyboard are manual, they do not experience this effect.

When we began this work, we identified three main uses of the mouse in window systems. One is to identify *focus*, i.e., which window will receive keystrokes. Another is *navigation*, the specification of window position and layout, including the stacking order, which affects window visibility. The third use is *direct manipulation* interfaces, wherein an application uses mouse input (via buttons, toggle switches, scroll bars, and the like) as user input.

Speech recognition seems particularly appropriate for the first two functions. In the case of managing keyboard focus, the mouse suffers the disadvantage of requiring the user to remove her hands from the keyboard (to find the mouse) exactly at the time when she desires to type into a different window. For navigation, the mouse suffers from being a two dimensional input device in the two-and-one-half dimensional world of overlapping windows. If a window is entirely buried by other windows, then it cannot be accessed until some of the windows covering it are moved.

In order to use voice for window navigation, we built *Xspeak*, an application which allows users to name windows by voice. When a window's name is spoken, it moves to the foreground on the screen, and the cursor is moved into the window to allow it to receive input. In other words, the user can change applications without removing her hands from the keyboard.

Itself an X application, Xspeak includes a graphical control panel (figure 1 and 2) which provides additional feedback (useful when recognition deteriorates) and utilities to train the recognizer and set audio gain levels. The control panel also provides a mechanism for the user to name a new window, which does not already exist in the speech recognizer's vocabulary.

An informal study of the use of Xspeak was conducted over several months with six subjects [6]. The subjects were four student programmers, employed for the summer, plus two of the Xspeak designers. For some of these users, Xspeak was quite attractive and heavily used, but for oth-

[1]I do not mean to suggest that voice and windows should be handled by the same server; I prefer an architecture with separate servers for each (and for other media as well). This paper is about higher level links between them.

ers, the mouse was the preferred means of interaction with windows. Those who enjoyed Xspeak thought speech input was faster, though, in truth, it was slightly slower than the mouse for the most simple window manipulations (timing was derived from videotaped sessions). Such a result would be expected from the divided attention hypothesis, which states that the cognitive load on the user is reduced by spreading multiple tasks across multiple input channels.

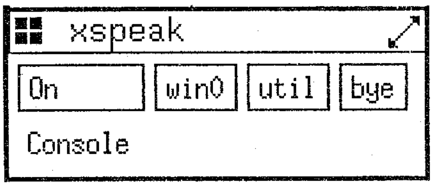

Figure 1: Xspeak control panel. The bottom text object displays recognition results.

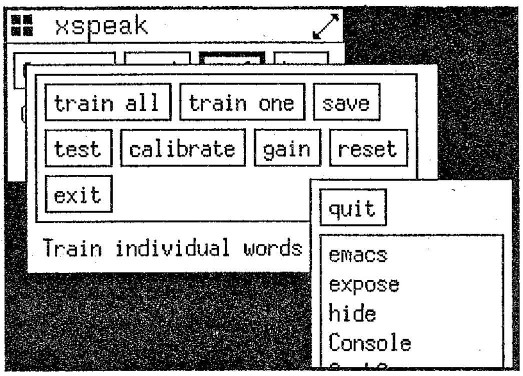

Figure 2: Xspeak pop-up utility menus.

Although it would be premature to draw strong conclusions from such an unconstrained pilot study, it did reveal a potential for improvement of the interface to window systems by the addition of voice input. We would not expect all users to embrace speech recognition, and some have already developed other mechanisms (icon managers and "rooms" among them) for partially coping with this problem. High accuracy speech recognition without wearing a microphone on one's head is also a difficulty.

Some of our users complained that they already use the mouse for many applications with direct manipulation interfaces, and although they may have desired to use voice to switch to a different window, there was no point in doing so if the mouse was required once arriving there. Users also wished to activate commands within text based Unix shells by voice, much like keyboard macros, but possibly involving focus shift as well.

These responses have encouraged follow up work to develop a language whereby a broader range of user input may be managed by voice. Such a language makes extensive use of the X event model. For example, the voice command "read mail" may be decomposed into the following series of events:

1. If a window named "mail" does not exist, start the mail reader application in a window so named and wait until that window has been exposed.

2. If the window named "mail" is not visible, move it to the top of the window stacking order and wait until it is exposed.

3. Move the pointer to the location of the button named "InBox".

4. Send mouse button down and up events, on button one, to the application.

Note that this language requires both conditional statements and procedural waits (for asynchronous X event notifications).

Xspeak is not, itself, a window manager, and it did not require the modification of any window managers.[2] This is possible due to the X Window System architecture, which provides the window system as a separate server process, with an inter-process communication method to exchange protocol requests to the server and receive events from the server by applications [3]. This is particularly true of the enhanced version of Xspeak, which relies on the ability to send "synthetic" mouse button and key press events to other applications in order to activate them. Xspeak also listens for window configuration notification events, which tell it when windows of interest have been made visible. This is necessary when Xspeak wishes to expose a window, which must occur before some types of input can reliably be directed to the window.

SoundViewer Widget

Recorded voice is a difficult medium for information presentation, at least in comparison to screen-based text display and mouse-based menu interaction. As mentioned earlier, speech is difficult to manipulate because it is slow and serial, and it requires our attention to decode it. Although we expect that enabling workstations to deal with voice as a data type will result in increased opportunities for remote telephone based computer access, it will still be the case that most workstation use, regardless of the data media, will be done while sitting before a screen.

Access to audio data can be improved by providing a graphical user interface for sound playback. Such an interface should provide a means of starting and stopping playback, and allow random access into a sound file using the mouse. Visual indicators should cue the user to the approximate length of the sound [2]; listeners have little tolerance for lengthy voice messages and would benefit from knowing the length of a sound at a glance. It would be beneficial to provide a single, consistent user interface to stored audio across all applications. This would make it easier for the user to detect the presence of audio data in an application, and the single interface would make access less confusing. It is useful to provide this graphical interactor as part of the toolkit which applications use to access the window system, both for consistency as well as ease of programming.

We designed the SoundViewer widget[3] to fulfill exactly these goals. The SoundViewer provides display of sound much like a scroll bar, which moves when the sound plays. It indicates sound length both with its width as well as with tick marks representing units of time (figure 3). When the user clicks a mouse button on the SoundViewer, the sound plays and the cursor bar moves along (horizontally) in synchronization. The user may grab the bar with the mouse and move it around for random access during playback. In providing such an interface, we overcome some of the limitations of the slow and serial nature of speech, by allowing the user more control over its playback.

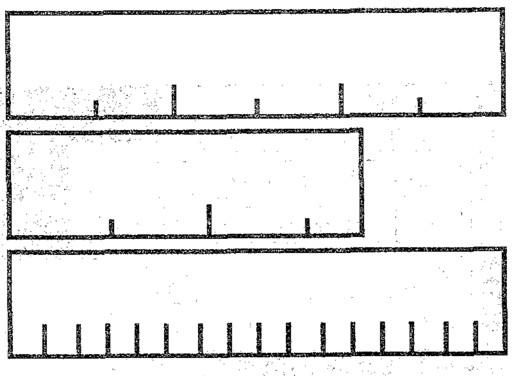

Figure 3: The SoundViewer widget, showing two fixed mode sounds and one scaled mode display at the bottom.

[2]It should be noted that Xpeak will work only with the so-called "real-estate" driven window managers, which assign keyboard focus to the window in which the cursor appears. Xspeak would require a simple modification to work with "click-to-type" window managers, which require a mouse click in a window to transfer focus to that window.

[3]In the X Window System, user interactor objects are called *widgets*.

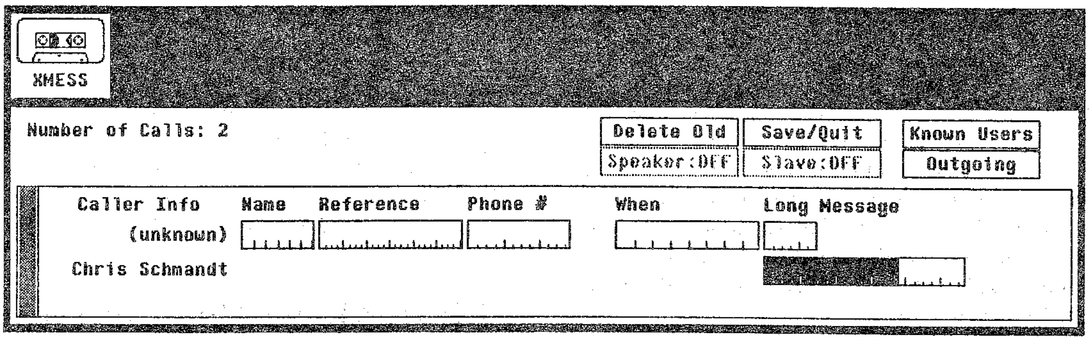

Figure 4: SoundViewer widgets used in a telephone answering machine.

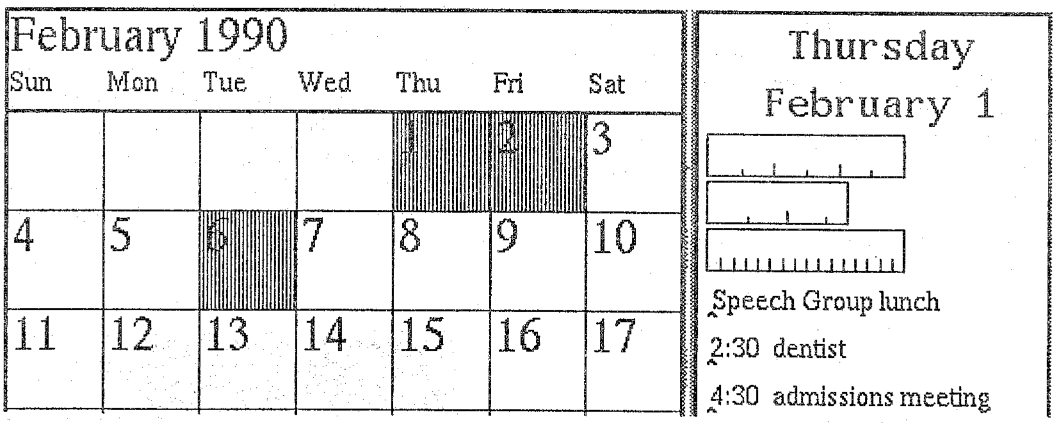

Figure 5: A multi-media calendar uses SoundViewers for recorded voice.

The SoundViewer has several display modes to accommodate sounds of various lengths. In the "fixed" mode, the application tells the SoundViewer its width (in pixels) and the length of sound it should display. These determine the number of milliseconds of sound per pixel, and the widget picks the appropriate spacing and height of the tick marks (for example, it may display seconds and tenths of seconds for short sounds, but tens of seconds and seconds for longer ones). When a sound name is assigned to the widget, it determines the actual display width of this sound, given its duration and the widget's mapping of duration to width. Thus, the displayed width of the sound is a length cue to the user.

Fixed mode is appropriate in a situation where there is not a great deal of variation in sound lengths. If very long sounds are interspersed with short sounds, either the former will not fit on the screen, or the latter will be too small to be of use. Unfortunately, some applications, such as voice mail, may experience a great deal of sound length variability. This problem can be accommodated using the SoundViewer's "scaled" mode, in which the widget scales the number of milliseconds per pixel to fill the allocated width. Since time is stretched, as a sound plays, the cursor moves with a variable rate, depending on the scale factor. The width of the widget is not a good cue to sound length, but the spacing of the tick marks and variable sound bar velocity during playback reveal the time compression.

The SoundViewer widget is currently being used as the standard interface to recorded audio in a number of applications at the Media Laboratory. One of these is *xmess*, a graphical user interface (figure 4) to recorded telephone messages taken by a conversational answering machine [4, 5]. Each message is displayed as one row of SoundViewers, one for each recorded message segment. Message segments can be played by clicking on each individual SoundViewer. The entire message is played by clicking on the message sender "button" on the left; as each segment plays, the appropriate SoundViewer is automatically activated.

Another application is *xcal*, the graphical interface to a multi-media calendar. Because a calendar database needs to be consulted frequently and always kept up to date, we built a telephone-based interface which uses touch tones for input and speech synthesis for output to describe appointments. When the user wishes to add an appointment, touch tones can be used to specify the date and time, but the actual calendar entry is then recorded. The screen-based interface (figure 5) shows calendar items as either text or SoundViewers, depending on their medium.

Multi-Media Cut and Paste

One of the most frequently used functions of a window system is the ability to move data from one application to another by selecting some portion of the text displayed in one window and depositing it to another, where it is treated as keystroke input. This procedure is referred to by a variety of names, including "clipboard", "selections" and "cut and paste".

Selections become complicated when multiple media are involved. First, a means of specifying the contents of the selection must be provided. In text windows, this is usually done by displaying selected characters in reverse video. With our SoundViewer widget, a (temporal) extent of a sound can be selected with the mouse; the selected region is indicated by XOR'ed lines (figure 6).

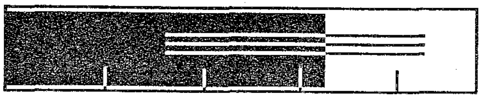

Figure 6: The SoundViewer uses XOR'ed lines to indicate the selection.

Additionally, a means must be provided to allow applications to negotiate data representation during transfer of the selection contents. The X Window System provides some convenient mechanisms for doing this, allowing particular named selections to be used as rendezvous points between applications, as well as including a protocol whereby clients can request and/or deny conversion of the selection into various representation types. For example, selection of a "telephone message" from an answering machine interface. This application could provide the contents of the selection as either a telephone number (which would be requested by a speed dialing tool), a name (requested by a rolodex), a sound name (audio editor), or even as a simple text string (for a "media-feeble" application such as a text editor).

Multi-media cut and paste operations can be implemented easily in X Windows, because the window system selection management protocols allow flexible negotiation as to the representation type of the desired data. The server acts as a broker in selection activity, but does not explicitly own the selection, leaving negotiation under the control of the applications. Whichever application wishes to own the selection, so notifies the server.

An application wishing to receive the contents of the selection makes a request to the server, specifying the representation type (*target* in X parlance). The selection owner receives notification of this request (through a callback if using the Intrinsics, the standard underlying toolkit support layer) and, if it can accommodate the requested representation type, it passes the data back to the server, which then delivers it. If the selection owner does not recognize the requested representation type or cannot cope with it, a rejection goes back to the requester via the server.[4]

This negotiation process makes it easy to extend the standard text selection which most applications currently support. A new type is defined, e.g. XA_SOUND, and sound-capable clients request the selection in this form. This allows SoundViewers to exchange portions of sounds. But the sound-capable application should also be able to supply some reasonable text string (such as a sound file name) when so requested by a media-feeble application. This allows gradual introduction of new representation types, with backward compatibility to weaker applications not yet capable of using the new types. It also allows the application owning a selection with a multitude of interpretations (such as the selected telephone message mentioned above), to easily pass this to another application in the preferred representation types in which that application expresses an interest.

[4]Well behaved X clients should also support representation type XA_TARGETS, which returns a list of supported representation types.

Acknowledgments

A number of members of the Media Laboratory have assisted the author in many aspects of this work. Mark Ackerman wrote the SoundViewer widget, and also contributed to Xspeak with Sanjay Manandhar and Debby Hindus. Sheldon Pacotti wrote Xcal. Lorne Berman wrote Xmess, and implemented much of the multi-media selection handling. Elaine McCarthy provided production assistance and editorial advice on this paper.

This work was funded by the MIT X Consortium and Sun Microsystems.

References

References

[1] D. A. Allport, B. Antonis, and P. Reynolds. On the division of attention: a disproof of the single channel hypothesis. *Quarterly Journal of Experimental Psychology*, 24:225 235, 1972.

[2] Brad A. Myers. The importance of percent-done progress indicators for computer-human interfaces. In *Human Factors in Computing Systems, CHI 85 Proceedings*, pages 11–17. ACM, 1985.

[3] R. W. Scheifler and J. Gettys. The X window system. *ACM Transactions on Graphics*, 5(2):79–109, 1986.

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

[5] C. Schmandt and B. Arons. Phone slave: A graphical telecommunications interface. *Proc. of the Soc. for Information Display*, 26(1):79–82, 1985.

[6] Christopher Schmandt, Debby Hindus, Mark Ackerman, and Sanjay Manandhar. Observations on using speech input for window navigation. In *Interact 1990 Conference Proceedings (to appear)*, 1990.

[7] A. Treisman and A. Davies. Divided attention to ear and eye. In *Attention and Performance*, volume IV, pages 101–117. 1973.

[8] C. D. Wickens, S. J. Mountford, and W. Schreiner. Multiple resources, task-hemispheric integrity, and individual differences in time-sharing. *Human Factors*, 23:211–230, 1981.