Augmenting a Window System with Speech Input

Chris Schmandt, Mark S. Ackerman, and Debby Hindus
Massachusetts Institute of Technology

Despite high expectations, there have been few convincing demonstrations of speech input in desktop computing environments. We have focused on window systems, where speech might provide an auxiliary channel to support window navigation.

Xspeak, our speech interface to the X Window System, associates words with each window. Speaking a window's name moves it to the front of the screen and moves the cursor into it. Speech does not provide a keyboard substitute, but it does assume some of the functions currently assigned to the mouse. Thus, a user can manage a number of windows without removing his or her hands from the keyboard.

We provided this interface to a group of student programmers who used it for several months. This pilot study was designed to identify some initial considerations for using speech recognition in workstations. The manner in which our programmers used voice pointed out its strengths and weaknesses. Recognition accuracy was critical, although some of our most enthusiastic users had some of the poorest recognition scores. The most consistent users started to use more windows and to allow more overlapping of windows. Some users had already developed their own techniques, which our voice interface couldn't help, for coping with multiple windows. Speech proved to be neither faster nor slower than the mouse, although the choice of which medium to employ was in part related to what else the user was doing with his or her hands.

With Xspeak, window navigation tasks usually performed with a mouse can be controlled by voice. A new version, Xspeak II, incorporates a language for translating spoken commands.

In a windowing environment, many applications support a direct-manipulation interface, where the user can click on buttons, pull scroll bars, and so on. Our student users complained about lack of voice access to these mouse functions. This led us to develop a user interface specification language so that voice commands could interact with applications by generating a series of mouse-motion, button-press, and key-press events. To improve recognition, vocabulary subsets specific to an application can be enabled either by voice or by mouse motion into a window.

The first part of this article gives some necessary background in speech recognition and window systems, with an analysis of how they might be combined. The second part describes Xspeak, our first navigation application, including its operation and our field study of its use. The final section introduces Xspeak II, an improved version that includes a user interface specification language, a rich tool for adding voice input to applications.

Background

Speech is a difficult input medium. Although speech recognition has received considerable positive publicity that has

0018-9162/90/0800-0050$01.00 © 1990 IEEE

raised the expectations of interface designers, the available devices leave much to be desired, particularly in recognition accuracy. Many variables affect error rates, including vocabulary size and composition, users' attitudes and speaking styles, ambient noise, and microphone type and placement.[1,2] Many of the high recognition rates reported are achieved by skilled users reciting lists of words in acoustically stable environments. This is very different from actual use in office environments where users may not be used to speech recognition.

Because of the difficulties in achieving high recognition accuracy, most successful applications use small vocabularies in amenable environments. Examples include:

- Inspections, such as noting defects of major appliances on a factory floor or testing circuit boards. The user's hands remain on the target of inspection, and voice is used to record results.
- Sorting, of either baggage or packages, where the user's hands are busy and voice is used to specify routing.
- Visual monitoring, especially with microscopes, for inspection in integrated-circuit and biomedical applications. The user's eyes are accommodated to the task, and the user's mouth may be in a stable position for a microphone.

These situations benefit from voice primarily because the user's hands and eyes are otherwise occupied.

The role of speech recognition in desktop computing is not so well established. There is little conclusive evidence that speech is superior to the keyboard for data entry, much less for free-form typing and editing. (For an excellent survey of the literature, see Martin.[3]) Much of the current work in large-vocabulary speech recognition is biased toward development of the so-called "listening typewriter," an automatic transcription device for business correspondence. Yet word processing may comprise only a fraction of a user's computer activities. And, although memorized text can be spoken as much as 500 percent faster than it can be written, dictation is not necessarily faster. Because composition requires the bulk of a writer's time, dictation may increase speed by only 20 to 65 percent.[4]

When might speech input be useful in a workstation? The evidence suggests that voice input is more valuable in conjunction with other input devices (such as keyboard and mouse). Judging by the successful industrial applications of speech recognition, in which the user performs an activity in parallel, we surmised that allowing users to remain focused on the screen and keyboard, instead of fumbling for the mouse, would be beneficial in a workstation environment.

To the extent that the tasks of navigation and interaction with the applications are separable, a performance improvement might be expected by splitting the input. Cognitive experiments have shown that a person's ability to perform multiple tasks is affected by whether those tasks use the same or differing modes, for example, spatial and verbal modes.[5,6] Such observations led Martin[3] to design an experiment using speech recognition for an alternate input channel in a CAD system employing both keyboard and mouse. Her subjects were indeed more productive with the addition of voice. She attributed this in part to the speed of speech recognition versus typing long command names and in part to the ability of users to split attention across channels, that is, to remain visually focused on the screen while using spoken commands.

Martin's second finding suggested the expected utility of speech as an interface to a visually complex window system. Moving between tasks, that is, between windows, is normally accomplished by using the mouse to move a cursor. This requires both manual and visual attention. Applying the divided-attention hypothesis and using different input channels for different classes of tasks might enhance navigation between windows.

Window systems. Windows are now commonplace on bitmapped computer workstations. Window systems allow the screen to be divided into a number of regions, with each region allocated to input or output from a particular computer process or program. Because windows are so ubiquitous and are indeed the substrate on which so much workstation use is based, we felt that no research into speech and user interfaces should ignore them. We chose to work with the X Window System because it is a de facto standard across workstations.

The X Window System defines a standard way for application programs, or X *clients*, to communicate with a separate process, the X *server*, that controls screen display and handles user input. Servers are typically provided by a hardware manufacturer. X clients include applications such as Xterm, a terminal emulator; Xclock, a clock; and Emacs, a programming editor.

Window managers, a specialized type of client, control the placement of application windows on the screen, usually through user control. They can vary a great deal in X, and because window managers are just applications in X, they can be used interchangeably. (For a taxonomy of window managers, see Myers.[7])

Two characteristics of window managers are important for our purposes. First, they may be *tiling* or *overlapping*. With an overlapping window manager, windows can partially obscure one another; with a tiling window manager, they cannot. Second, window managers differ according to how they select which window receives keystrokes. The mechanism for shifting input focus may be *click to focus* (sometimes called "sticky"), requiring a mouse button click within a window before keystrokes are accepted. Or the window manager may be *real-estate based* and automatically shift the focus to the window where the mouse pointer appears. The window managers selected by our users were all overlapping and real-estate based; none were modified.

Window systems and speech recognition. Where, then, can speech be most profitably employed in a window system? And what functionality should it augment?

Before placing a speech interface within a windowing system, we had to consider how window systems are used. But there has been surprisingly little study of this or why users prefer a particular interface. Gaylin[8] discusses frequency of use of some window operations. Card, Pavel, and Farrell[9] provide a loose taxonomy of how windows are used in tasks. More important for our purpose was Bly and Rosenberg's[10] comparison of tiled and overlapped windows in a task that involved searching for information between windows. When the amount of text to be searched was not entirely visible, they found that overlapping windows were more effective than tiled windows, with an interesting bimodality. For the most-experienced users, overlapping windows were faster, but for some less-experienced users, they were significantly slower. Bly and Rosenberg attributed this to the added navigational tasks of manipulating the various windows. However, despite this added cognitive load, their users preferred overlapping windows.

Window systems force use of a spatial metaphor. Users organize their windows

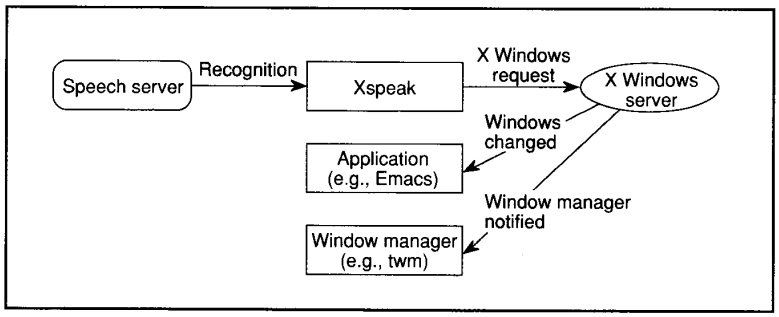

Figure 1. Interaction between processes in Xspeak.

geometrically, perhaps stacking them in layers. Visually, it is relatively simple to recognize a window when there are few windows and each is in a distinct geometric position. But, as the number of windows increases, it becomes progressively more difficult to find a window through visual inspection. Moreover, the mouse, a two-dimensional spatial input device, is not matched to the two-and-a-half dimensions of overlapping windows. (Tiled windows, if they are stacked in layers, may also have planes.) As the number of windows grows, using the mouse to interact with a "buried" window becomes more difficult. A window with no part exposed may be inaccessible to the mouse until other windows are moved out of the way.

Speech offers an alternative. Voice, not being tied to a spatial metaphor, can interact with windows directly, regardless of their degree of visual exposure. Speech, then, could let users employ many task-specific windows. Furthermore, navigation is a good candidate for optimization via the use of multiple input channels.

The above suggests that in a complex window environment, especially with users who would like to create many windows, an interface designed to improve navigation would provide faster access to various windows. Therefore, navigation was our prime candidate for a speech interface. Further, to the extent that navigation could be differentiated as a separate task from the activities occurring within each window, multimodal input might lessen the user's cognitive load. This could allow successful use of a larger number of windows dedicated to specific tasks.

Xspeak

Xspeak is an application, not a window manager, that allows voice access to windows in the X Window System. It runs on Sun workstations (it should run with any X Windows server), using a Texas Instruments speech card in a PC-based audio server.[11] We did not modify the X server, the user's window manager, or any other application.

Xspeak associates windows with voice *templates*, words trained and stored in the recognizer and constituting its vocabulary. Speaking a window's template pops the window to the foreground and moves the mouse pointer to the middle of the window. The window manager, which does not distinguish this motion from mouse motion, shifts the input focus to the appropriate window. At this point, keystrokes are directed to the application running within the window. Figure 1 shows this interprocess communication.

Xspeak also allows users to fully lower or raise the current window. Users can move between windows and rearrange them without removing their hands from the keyboard. However, the mouse remains the sole means for moving and resizing windows. These operations, which are much less frequent than navigational operations,[8] are cumbersome to perform with voice commands.

Providing a speech interface to the window system was relatively straightforward. Because all applications are in separate processes from the X Windows server, moving a top-level window does not corrupt any application's data structures. Xspeak would have been much more difficult to implement in a window system that does not separate server and client.

In Xspeak, a configuration file associates window titles with the template numbers in the recognizer. Xspeak associates the window title (the window name property set by the application on its top-level window) with a particular window ID, which is used to modify the window stacking order. Xspeak also lets users name new windows not found in the configuration file. To do this, the user clicks on the window being named so that Xspeak can determine the window ID. The user then speaks the new name, that is, he or she trains a recognizer template. The configuration file also lets users start applications. If a window's name is spoken and no matching window ID can be found, the rest of the corresponding entry in the configuration file is executed to create the new window. For example, in a configuration file, this line

```
emacs -f emacs
```

would make Xspeak create an Emacs window if one did not already exist.

Xspeak includes a graphical control panel (see Figure 2) that serves several functions. Its status display indicates to the user that the recognizer is working. When the user says a word, this panel displays the word or a message indicating that no word was recognized. This feedback is essential when recognition accuracy is low due to poor word training or increased background noise.

The control panel includes a button to invoke window naming. The user can disable or enable recognition, using another button, to avoid spurious recognition while conversing or answering the phone. From the Xspeak control panel, users can select the utility function ("util" in Figure 2) to access less frequently used commands. This panel contains commands to test, calibrate, and retrain the recognizer. Users attempting to improve recognition accuracy frequently chose to retrain individual words.

Microphones. We were unwilling to subject our users to head-mounted microphones because these microphones are uncomfortable and tend to slip. Also, if users forget they are wearing the microphone, they may be unpleasantly reminded when they attempt to drink coffee or answer the telephone. Instead we placed a super-cardiod microphone (Sennheiser ME-80) next to the workstation monitor. Our choice of microphones contrasts significantly with much other published work on speech recognition and resulted in poor recognition performance. It is more common to use a head-mounted, noise-canceling microphone to control microphone acoustics and compensate for background noise.

A more directional microphone might have decreased background noise from

sources such as fans and telephones, but it would also have been more sensitive to the speaker's position. To make our microphone work well, we had to change a number of internal parameters in the recognizer to deal with the higher noise level. (Although these internal parameters are documented, a systems developer unfamiliar with the operation of speech recognizers probably couldn't decipher them, much less optimize their values.)

Not using noise-canceling microphones tends to cause *insertion errors*, that is, picking up of background noise as speech. Recognizers are generally poor at discriminating whether a particular word is within their universe of templates. The consequence of insertion errors is window reconfiguration; suddenly, user input goes to the wrong window (especially annoying if keyboard noise caused the error). Thus, we set the rejection threshold on the recognizer rather high, at the price of making the rejection of correctly spoken words much more likely.

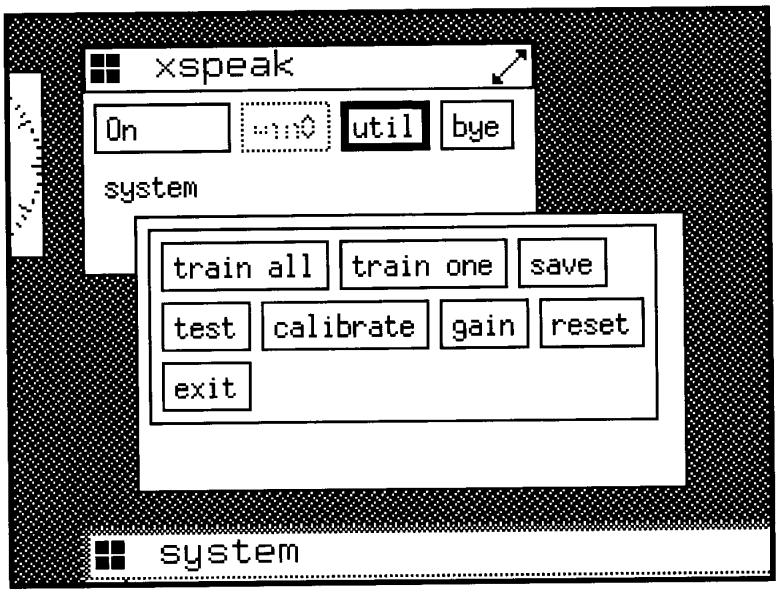

Figure 2. Xspeak control and utility panels.

User experiences. To better understand how Xspeak would be used and what effects it would have on users, we conducted a small pilot study. We wanted to know under what circumstances users would choose voice input for navigation, where they would encounter difficulties, and how Xspeak would affect their window use. By observing real users, we could learn what changes and enhancements were needed to improve Xspeak. We were also curious about how users would react to using the less-than-perfect speech-recognition system on a long-term basis. This last issue has not received much attention; most published studies of user reactions to speech-recognition systems have used the research technique of using a hidden human to simulate a perfect recognizer.

Over a summer, four student programmers in the speech group, as well as two of the authors, used Xspeak in their day-to-day programming tasks. With one exception they were already familiar with the X Window System. After an entry interview, the users were trained to use Xspeak. We tracked usage over a two-month period via extensive automatic logging, periodic videotaping, and frequent short interviews. We derived recognition accuracy rates, and we collected timing data for comparable mouse and speech actions over a small set of navigation tasks. (A detailed report on our methodology and results is available.[12])

From our analysis of these empirical and observational data, we reached the following conclusions about our users' experiences with Xspeak:

• Recognition is not straightforward. Although we used Xspeak in relatively quiet offices, the microphone configuration resulted in recognition errors.

Several steps were required to deal with these errors. First, we changed a number of the recognizer's internal parameters. Second, we set a high recognition threshold. Third, we placed the microphone on a stand to one side of the monitor, carefully positioned to point toward the user; a better solution might employ a microphone built into the keyboard or monitor bezel. Fourth, we provided our users with utility functions to retrain, recalibrate, and reset the speech recognizer; one user retrained 109 individual words in 79 sessions.

Despite these actions, low recognition accuracy rates remained a problem. They ranged from slightly less than 50 percent to more than 80 percent (measured for the six pilot users during a randomly selected session and confirmed in a follow-on study involving three of the six users). Poor recognition accuracy was the greatest impediment to acceptance of Xspeak. The users who persisted had some of the highest overall recognition rates but also developed successful strategies to overcome errors.

• Some programmers preferred using a faster workstation without Xspeak to using a slower workstation with the speech interface. This might have been exacerbated by relatively slow performance by the X Windows server for programmers developing X Windows applications. In any case, a somewhat improved user interface is no substitute for a faster processor.

• For simple change-of-focus tasks (moving the mouse from one exposed window to another exposed window), speech was not faster than the mouse. In fact, it was marginally slower. The speed advantage shifted toward speech if the destination window was partially or completely hidden. Exposing such a window requires no additional time for a voice interface but does require several additional mouse actions.

• Navigation in a window system *can* be handled with speech input. Users were able to move among and restack windows with ease. They learned the interface quickly and needed little tutoring to use the basic functions, although the control panel required more training.

• Some users were not helped much by the voice interface. One of them, a very experienced window-system user, had already developed techniques for coping with many windows (by using many icons). Another spent much of his time thinking at the keyboard and had few interactions with his windows; with this work style, transitioning among windows may be less critical.

Verbs

create	Start an application, thus creating its windows.
recall	Reposition a window to the top of the window stack.
hide	Reposition a window to the bottom of the window stack.
return	Reposition a window to its previous position in the window stack.
configure	Move or resize a window.
place	Move the mouse to a specified position or named window without restacking.
if-elseif-endif	Conditionally execute a block of instructions.
wait-on	Stop execution until some condition is achieved or a timeout occurs.
send	Send a specified X Windows event to the named application window.
string	Send a series of keyboard events to the named application window.
activate	Activate a recognizer subtemplate.
name	Rename a window from a specified set of names.

Conditions

process	Determine whether the named process is executing.
iconified	Determine whether the named window is iconified.
map	Determine whether the named window is on the screen.
xevent	Determine whether the specified X Windows event has been sent to a named window (used for handshaking with the server).
timer	Determine whether a specified time has elapsed.

Figure 3. G-XL language.

• Toward the end of the observation period, we noticed that the users most inclined to use voice increased the number of overlapping windows or the degree of overlap.

• We found the use of voice in navigation an incomplete substitute for the mouse. Our users did not rely on the speech interface to the exclusion of the mouse. They still had to use the pointer to interact with the direct-manipulation interfaces within applications. Having a hand already on the mouse accounted for 59 percent of the times users navigated with the mouse rather than with Xspeak. Some users found it awkward to use both interfaces simultaneously. Others wanted to use Xspeak to handle direct-manipulation buttons or to start programs.

Xspeak II

User interfaces require iterative design cycles. Hence, a key goal of our Xspeak prototype was to learn what facilities would be useful in a speech interface. After considering Xspeak's usage, users' requests, and our improved understanding of the possibilities of a speech interface, we redesigned Xspeak to fix bugs, correct mistakes, and, most importantly, add features.

We made two major changes. First, since context-dependent recognition improves recognition rates, we added the ability to create subtemplates.

Second, Xspeak II includes a specialized language, G-XL, to facilitate general-purpose handling of the window system. Where Xspeak was limited in its use of the pointer device, Xspeak II allows greater flexibility in the speech interface. Users can employ direct manipulation using voice, interacting with an application in addition to simply selecting it.

Application-sensitive recognition. Increasing Xspeak's scope would require a potentially much larger vocabulary. But a larger vocabulary is apt to introduce more recognition errors because more words could be confusable. This standard speech-recognition trade-off was critical in Xspeak; its recognition accuracy was already barely acceptable.

To minimize the impact of this trade-off, voice-input applications commonly break the vocabulary into subsets. If the dialogue can be structured so that the branching factor remains small, then the number of words active at any point can be minimized. For Xspeak II, we chose to create vocabulary subsets according to applications. Grouping and enabling the templates lets Xspeak II switch among applications as they are invoked. Xspeak II also maintains cooperability with the mouse; whenever the mouse is used to enter a window, the corresponding vocabulary subset is enabled.

Xspeak II language, G-XL. The original Xspeak was limited in its range of operations. For instance, users could not use voice to control direct-manipulation objects such as scroll bars. Furthermore, there was no way to group functionality (such as having two windows pop to the top of the window stack), to conditionally invoke programs based on the user's current environment, or to wait for a window to become exposed before proceeding.

G-XL, Xspeak II's language, addresses many of these limitations. It also meets three major requests of the pilot-study users: macro capability for all X Windows events, greater control over screen events and process sequencing, and direct manipulation of objects. We have designed and are implementing G-XL to provide a flexible interface between speech and a window system. With G-XL, users can tailor their speech interface in a variety of ways.

Figure 3 lists the G-XL language. The verbs *create*, *recall*, and *hide* provide the basic functionality of the original Xspeak. To provide needed flexibility, G-XL contains the *if (condition)-elseif-endif* construction and the *wait-on (condition)* construction. The conditions include *map*, to test whether a window is present on the screen; *process*, to test whether a process exists; *timer*, to test for an elapsed time; *iconified*, to test whether the application has been iconified; and *xevent*, to check for most X Windows input events on a window.

Figure 4 shows parts of a G-XL configuration file. The section beginning with *emacs* checks whether there is an Emacs editor on the screen. If there is, whether or not it is iconified, the full Emacs window is popped above any other window. If there is no Emacs window, one is created. The inner *if* block shows how a user can control the shape and position of the Emacs win-

dow. In this case, if the debugger, dbx, is already running, Emacs will appear in a smaller window placed further to the right, so as not to obscure the dbx window. The *configure* verb can resize or move an X window.

Additionally, Xspeak users wanted to handle the direct-manipulation objects (widgets) that are part of the Xt toolkit and used extensively by X Windows applications. To do this, G-XL allows placing the pointer within a specific window (to focus input) and sending an artificial input event. The *place* verb puts the pointer at a given (x,y) location relative to the current window. The *send* verb takes most X Windows input event types and artificially sends them to the given window, usually the current window. Applications do not distinguish between these artificial events and user input, as shown in Figure 5. The *string* verb represents a series of *send keypress* commands and provides for keyboard macros.

G-XL can send keystrokes direct to an application, but it is more than a simple keyboard macro package. First, G-XL knows about a number of X Windows event types, including keystrokes, button presses, and those events returned from the server when, for example, a window is created or resized. Second, as described above, a number of its primitives are actually functions that allow sequencing of operations in the multiprocess X Windows environment, where requests must be acted on by the server. For example, if a window is obscured by another window, the button-press event cannot be sent to the application until the window has been exposed.

Three additional features round out G–XL. The *activate* verb activates a recognizer subtemplate. While the general template is always active, subtemplates are swapped in and out as required; this also provides local scoping. The *return* verb restores the window stack and the pointer location to their states before the current subtemplate was activated. Finally, the *name* verb allows greater flexibility in dynamically creating windows. On a *create* request, which sequentially could produce several windows with the same X title, the *name* verb renames those windows from a set of given names.

G-XL configuration files are compiled by the user and may be specified on the Xspeak II command line. For example, a user might have several different configuration files corresponding to various window managers.

```
template general
  mail
    .
    .
    .
  emacs
    if (!map emacs)
      if (!process dbx)
        create emacs -rv -geometry 80x50+10+50
      elseif
        create emacs -rv -geometry 80x50+500+50
      endif
      wait (xevent MapNotify (window emacs))
    elseif
      recall emacs
      if (process dbx)
        configure emacs 80x50+500+50
      endif
    endif
    activate emacs
  messages
    .
    .
    .
end template
```

Figure 4. Sample G-XL template.

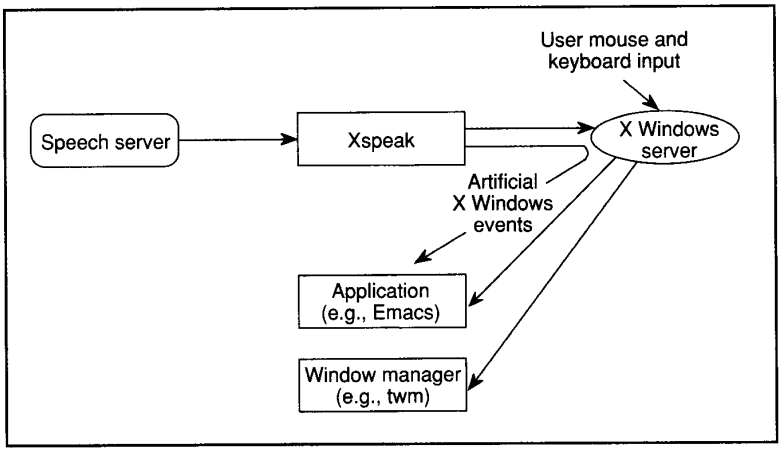

Figure 5. Interaction between processes in Xspeak II.

We intend to make Xspeak II available to a wider audience and to closely monitor their usage patterns and reactions. We also want to gather data to evaluate how adding a speech channel for navigation affects users' number, placement, and use of windows.

Even with the limited utility of our initial prototype, it is clear that at least some users find a speech interface comfortable and beneficial. As we discover how to integrate voice with window systems, we will progress towards a deeper understanding of the roles voice can play in desktop computing environments. ■

Acknowledgments

Sanjay Manandhar did much of the original Xspeak programming to work out the basic interaction mechanisms. Gale Martin provided insightful early discussions. Wendy Mackay provided invaluable assistance with our use of video as an evaluation method. Ralph Swick helped us with the more arcane aspects of the X Window System. The reviewers made many pertinent comments and suggestions. This project was funded by the MIT X Consortium and Sun Microsystems.

References

1. H.C. Nusbaum et al., "Testing the Performance of Isolated-Utterance Speech Recognition Devices," *Proc. 1986 Conf.*, American Voice I/O Society, pp. 393-408.
2. A.W. Biermann et al., "Natural Language with Discrete Speech as a Mode for Human-to-Machine Communication," *Comm. ACM*, Vol. 28, No. 6, 1985, pp. 628-636.
3. G.L. Martin, "The Utility of Speech Input in User-Computer Interfaces," *Int'l J. Man-Machine Studies*, Vol. 30, 1989, pp. 355-375.
4. J.D. Gould, "How Experts Dictate," *J. Experimental Psychology: Human Perception and Performance*, Vol. 4, No. 4, 1978, pp. 648-661.
5. D.A. Allport, B. Antonis, and P. Reynolds, "On the Division of Attention: A Disproof of the Single-Channel Hypothesis," *Quarterly J. Experimental Psychology*, Vol. 24, 1972, pp. 225-235.
6. A. Treisman and A. Davies, "Divided Attention to Ear and Eye," in *Attention and Performance*, Vol. IV, 1973, pp. 101-117.
7. B.A. Myers, "Window Interfaces: A Taxonomy of Window Manager User Interfaces," *IEEE Computer Graphics and Applications*, Vol. 8, No. 5, Sept. 1988, pp. 65-84.
8. K.B. Gaylin, "How Are Windows Used? Some Notes on Creating an Empirically Based Windowing Benchmark Task," *Human Factors in Computer Systems — CHI 86 Conf. Proc.*, ACM, 1986, pp. 96-101.
9. S.J. Card, M. Pavel, and J.E. Farrell, "Window-Based Computer Dialogues," *Proc. Interact 84, First IFIP Conf. Human-Computer Interaction*, IFIP, Geneva, Switzerland, 1984, pp. 239-243.
10. S.A. Bly and J.K. Rosenberg, "A Comparison of Tiled and Overlapping Windows," *Human Factors in Computer Systems — CHI 86 Conf. Proc.*, ACM, 1986, pp. 101-106.
11. C. Schmandt and M. McKenna, "An Audio and Telephone Server for Multimedia Workstations," *Proc. Second IEEE Conf. Computer Workstations*, Computer Society Press, Order No. 810, 1988, pp. 150-159.
12. C. Schmandt et al., "Observations on Using Speech Input for Window Navigation," tech. report, MIT Media Lab, Cambridge, Mass., 1990.

Chris Schmandt is a principal research scientist and the director of the Speech Research Group at MIT's Media Laboratory. He has been at the Media Laboratory since its creation five years ago and spent the previous five years with its predecessor, the Architecture Machine Group. Schmandt's research covers a broad range of conversational computer systems, with current emphasis on voice and telephone interactions with workstations. He holds BS and MS degrees from MIT.

Mark S. Ackerman is a doctoral candidate in information technology at MIT. He works with the Coordination Technology Group at the Center for Coordination Science and with the Speech Research Group at the Media Laboratory. His current research interests include window systems, electronic reference systems, and human-computer interaction. He received a BA in history from the University of Chicago and an MS in computer and information science from Ohio State University.

Debby Hindus is a graduate student and research assistant in the Speech Research Group at MIT's Media Laboratory. Her research interests are in incorporating voice input and output into everyday computer use. Hindus previously consulted on user interface and software development issues. She received a BS degree in computer science from the University of Michigan.

The authors' address is Media Laboratory, Massachusetts Institute of Technology, E15-327, 20 Ames St., Cambridge, MA 02139.