Bernard A. Rozmovits

The Design of User Interfaces for Digital Speech Recognition Software

Digital Speech Recognition Software (DSRS) adds a new mode of interaction between people and computers—speech. DSRS is a command and control application integrated with the UNIX desktop environment. It accepts user commands spoken into a microphone and converts them into keystrokes. The project goal for DSRS was to provide an easy-to-learn and easy-to-use computer–user interface that would be a powerful productivity tool. Making DSRS simple and natural to use was a challenging engineering problem in user interface design. Also challenging was the development of the part of the interface that communicates with the desktop and applications. DSRS designers had to solve timing-induced problems associated with entering keystrokes into applications at a rate much higher than that at which people type. The DSRS project clarifies the need to continue the development of improved speech integration with applications as speech recognition and text-to-speech technologies become a standard part of the modern desktop computer.

In the 1960s and early 1970s, people controlled computers using toggle switches, punched cards, and punched paper tape. In the 1970s, the common control mechanism was the keyboard on teletypes and on video terminals. In the 1980s, with the advent of graphical user interfaces, people found that a new mode of interaction with the computer was useful. The concept of a pointer—the mouse—evolved. Its popularity grew such that the mouse is now a standard component of every modern computer. In the 1990s, the time is right to add yet another mode of interaction with the computer. As compute power grows each year, the boundary of the man–machine interface can move from interaction that is native to the computer toward communication that is natural to humans, that is, speech recognition.

DSRS Product Overview

Very simply, DSRS is an application that provides speech macros. The user speaks a command, phrase, or sentence (that is, an utterance), and DSRS performs some actions. The action might be to launch an application, for example, in response to the command "bring up calendar"; or to type something, for example, in response to "edit to-do list," to invoke emacs \files\projectA\todo.txt. DSRS not only houses the speech macro capability but also provides a user interface, a speech recognition engine, and interfaces to the X Window System.

Following is a high-level description of how the software functions. Commands are spoken into a microphone, and the audio is captured and digitized. The first step in the processing is the speech analysis system, which provides a spectral representation of the characteristics of the time-varying speech signal. Next is the feature-detection stage. Here, the spectral measurements are converted to a set of features that describe the broad acoustic properties of the different phonetic units.[1] These representations of the speech signal are then segmented and identified as phonetic sequences. The speech recognition engine accepts these phonetic sequences and returns word matches and confidence values for each match. These data are used to determine if each match is acceptable. If a

match is acceptable, DSRS retrieves keystrokes associated with each utterance, and the keystrokes are then sent into the system's keyboard buffer or to the appropriate application. For instances of continuous speech recognition, a sentence is recognized and keystrokes are concatenated to represent the utterance. For example, for the utterance "five two times seven three four equals," the keys "52 * 734 =" would be delivered to the calculator application.

Although this concept seems simple, its implementation raised significant system integration issues and directly affected the user interface design, which was critical to the product's success. This paper specifically addresses the user interface and integration issues and concludes with a discussion of future directions for speech recognition products.

Project Objective

The objective of the DSRS project was to provide a useful but limited tool to users of Digital's Alpha workstations running the UNIX operating system. DSRS would be designed as a low-cost, speech recognition application and would be provided at no cost to workstation users for a finite period of time.

When the project began in 1994, a number of command and control speech recognition products for PCs already existed. These programs were aimed at end users and performed useful tasks "out of the box," that is, immediately upon start-up. They all came with built-in vocabulary for common applications and gave users the ability to add their own vocabulary.

On UNIX systems, however, speech recognition products existed only in the form of programmable recognizers, such as BBN Hark software. Our objective was to build a speech recognition product for the UNIX workstation that had the characteristics of the PC recognizers, that is, one that would be functional immediately upon start-up and would allow the nonprogrammer end user to customize the product's vocabulary.

We studied several speech recognition products, including Talk→To Next from Dragon Systems, Inc., VoiceAssist from Creative Labs, Voice Pilot from Microsoft, and Listen from Verbex. We decided to provide users with the following features as the most desirable in a command and control speech recognition product:

- Intuitive, easy-to-use interface
- Speaker-independent models that would eliminate the need for extensive training
- Speaker-adaptive capability to improve accuracy of words
- Continuous speech recognition capability
- Prompts for active vocabulary
- Minimum use of screen area
- User control over the user interface configuration
- Simple mechanism to modify and create new vocabulary
- Integration with the X Window System
- Support for out-of-the-box desktop applications provided with the UNIX operating system
- Support for vi and emacs editors, and for C programming

The DSRS Architecture

DSRS comprises several major components which are outlined below and illustrated in Figure 1. Of these components, three are licensed from Dragon Systems, Inc.: the front-end processor, the recognizer engine, and the speaker-independent speech models.

Dragon Systems, Inc. was chosen as the provider of the speech recognition engine based on the accuracy of their technology, their products and expertise in other local languages, and their long-term commitment to speech recognition.

Data acquisition consists of the microphone, audio card, and the multimedia services application programming interface (API) that provides support for the sound card.

The *front-end processor* analyzes a stream of digitized data and differentiates between silence, noise, and speech; it then extracts a set of computed features from the speech signals.

The *recognizer,* or speech recognition engine, accepts the computed representation of the speech in the form of feature packets which drive the Hidden Markov Models to recognize utterances. Hidden Markov Models are basically state machines that transition from a beginning state to a number of internal states and then to a final state based on input data and probabilities.[2] Each transition carries two sets of probabilities: a transition probability, which provides the probability of this transition being taken, and an output probability density function (PDF), which is the conditional probability of emitting each output symbol from a finite alphabet given that a transition is taken.[3] The PDFs are adapted when the model is "trained," that is, customized, by the individual user.

The *finite state grammar* is a state machine that contains a representation of the vocabulary supported by DSRS. Each state contains words, phrases, or sentences; their associated actions; and the information needed to transition to the next state. The current state is used to control the Active words.

The *speech models* are a set of utterance models used by the recognizer. DSRS provides vocabulary and speaker-independent models for the applications supported by DSRS. Users who wish to include their own

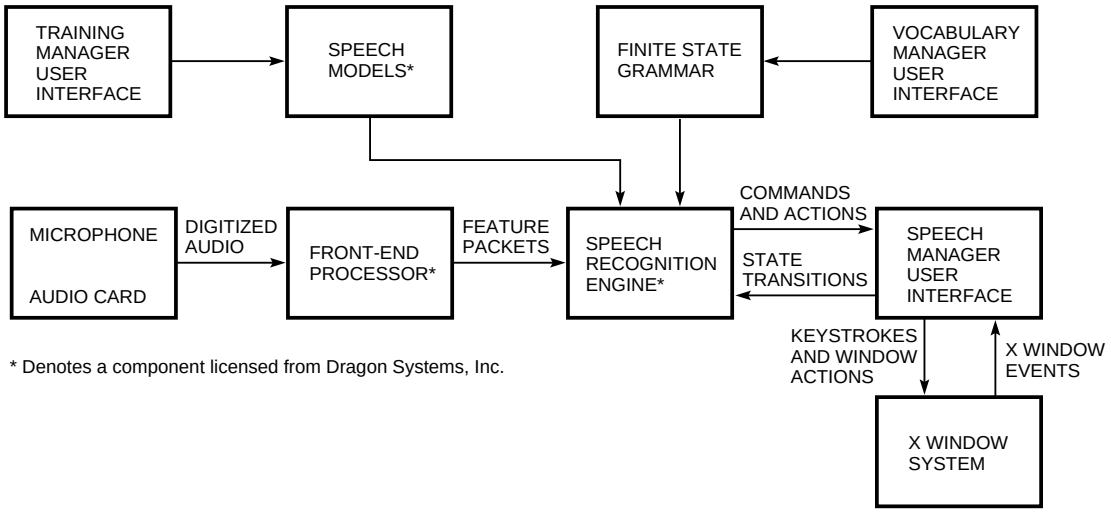

Figure 1
DSRS Architectural Block Diagram

words can create models using the Vocabulary Manager user interface.

The *Speech Manager* is the main user-interface component. The Speech Manager window provides visual feedback to users. It also keeps track of the current window in focus and acts as the agent to control focus in response to users' speech commands.

The *Vocabulary Manager* user-interface window displays the current hierarchy of the finite state grammar file. The Vocabulary Manager allows the user to customize using the functions for addition, deletion, and modification of words or macros. Also in this window, the command-utterance to keystroke translations are displayed, created, or modified.

In the *Training Manager* user interface, the user may train newly created words or phrases in the user vocabulary files and retrain, or adapt, the product-supplied, independent vocabulary.

The DSRS Implementation

As the design team gained experience with the DSRS prototypes, we refined user procedures and interfaces. This section describes the key functions the team developed to ensure the user-friendliness of the product, including the first-time setup, the Speech Manager, the Training Manager, the Vocabulary Manager, and the finite state grammar.

First-time Setup

DSRS requires a setup process when used for the first time. The user must create user-specific files and settings. The user begins by selecting the microphone and by testing and adjusting the microphone input volume to usable settings. The user is then prompted to speak a few words, which are presented on the screen. DSRS uses the speech data to choose the speaker-independent model that most closely matches the speaker's voice. There are models for lower- and higher-pitched voices. The software copies the selected model to the user's home directory; the model is then modified when the user makes changes to the provided models and vocabulary. After setup is complete, the next step is the Training Manager which presents the user with a list of 20 words to train; when this step is completed, DSRS is ready for use. The Training Manager is described in more detail later in this section.

The procedure above was developed to take a new user through the entire setup process without the need to refer to any documentation. Once the user files are created, DSRS bypasses these steps and comes up ready to work. A notable change that we made to the setup was instigated by our own use of the software. We found that inconsistent microphone volume settings were a frequent problem. When systems were rebooted, volume settings were reset to default values. Consequently, we created an initialization file that records the volume settings as well as all user-definable characteristics of the graphical user interface.

Speech Manager

Once DSRS is ready and in its idle state, it presents the user with the Speech Manager window, an example of which is shown in Figure 2. The Speech Manager provides the following critical controls:

- Microphone on/off switch.
- A VU (volume units) meter that gives real-time feedback to the audio signal being heard. A VU meter is a visual feedback device commonly used on devices such as tape decks. Users are generally very comfortable using them.

Figure 2
DSRS Speech Manager Window

- Two user-controllable panes that display the Always Active and Active vocabulary sets. The Always Active vocabulary words are recognized regardless of the current application in focus. The Active vocabulary words are specific to the application in focus and change dynamically as the current application changes. The vocabularies are designed in this way so that a user can speak commands both within an application context and in order to switch contexts.
- Three small frames that provide status information to the user.
  - The Mode frame indicates the current state of the Speech Manager: command and control or sleeping.
  - The Context frame displays the class name of the application currently in focus. This context also determines the current state of the Active word list.
  - The history frame displays the word, phrase, or sentence last heard by the recognizer. The history frame is set up as a button. When pressed, it drops down to reveal the last 20 recognized utterances.
- A menu that provides access to the management of user files, the Vocabulary Manager, the Training Manager, and various user-configurable options.

Training Manager

The Training Manager adapts the speaker-independent speech models to the user's speech patterns and creates new models for added words. Our study of PC-based speech recognizers led us to the conclusion that the design of a training interface is critical to obtain good results. For example, the training component of one PC-based recognizer we examined did not provide clear feedback to the user when an utterance had been processed, thus causing the user confusion about when to speak. This confusion lead to training errors and frustration. Another recognizer did not allow the user to pause while training, a major inconvenience for the user who, for example, needed to clear his throat or speak to someone.

We developed the following list of design characteristics for a good training user interface.

- Strong, clear indications that utterances are processed. We added a series of boxes that are checked off as each utterance is processed and a VU meter that shows the system is picking up audio signals.
- Reduced amount of eye movement needed for the training to proceed smoothly and quickly. We placed visual feedback objects in positions that allow users to focus their eyes on a limited area of the screen and not have to look back and forth across the screen at each utterance.
- A glimpse of upcoming words. A list of words is displayed on the user interface and moves as words are processed.
- A progress indicator. Text is displayed and updated as each word is processed, indicating progress, for example, Word 4 of 21.
- Option to pause, resume, and restart training.
- Large, bold font display of the word to be spoken and a small prompt, "Please continue," displayed when the system is waiting for input.
- Automatic addition of repeated utterances that are "bad" or do not match the expected word.
- Control over the number of repetitions.

As the example in Figure 3 shows, the Training Manager presents a word from a list of words to be trained. The word to be spoken is presented in a large,

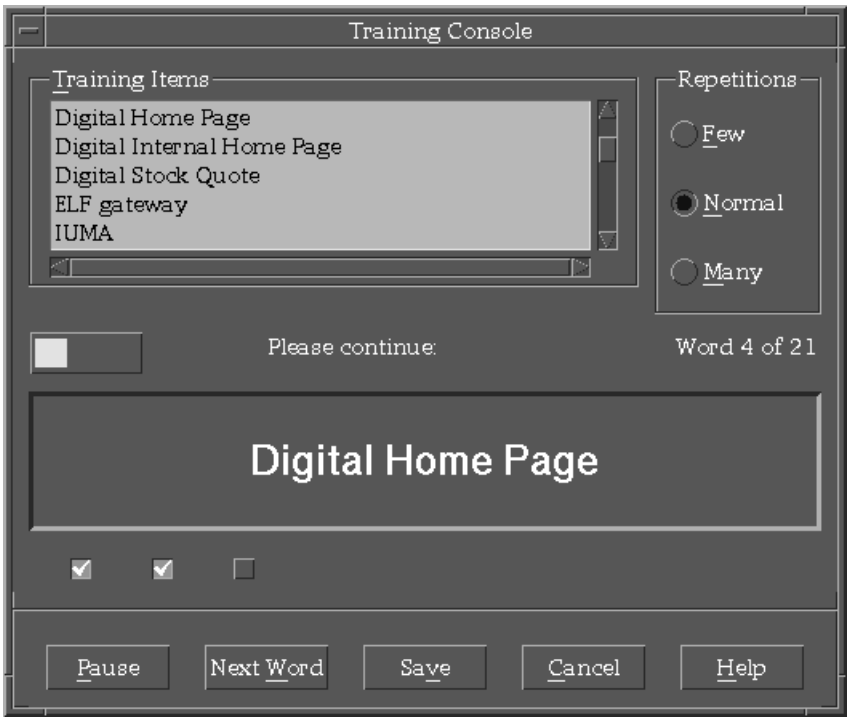

Figure 3
Training Manager Window

bold font to differentiate it from the other elements in the window. To train the words, the user repeats an utterance from one to six times. The user must speak at the proper times to make training a smooth and efficient process. DSRS manages the process by prompting the speaker with visual cues. Right below the word is a set of boxes that represent the repetitions. The boxes are checked off as utterances are processed, providing positive visual feedback to the speaker. When one word is complete, the next word to be trained is displayed and the process is repeated. When all the words in the list are trained, the user saves the files, and DSRS returns to the Speech Manager and its active mode with the microphone turned off.

Vocabulary Manager

The Vocabulary Manager, an example of which is shown in Figure 4, enables users to modify speech macros by changing the keystrokes stored for each command and by adding new commands to existing applications. Users can also add speech support for entirely new applications. The vocabularies are represented graphically as hierarchies of application vocabularies, groups of words, and individual words. The Vocabulary Manager provides an interface that allows manipulation of this database of words without resorting to text editors. The Always Active vocabularies are accessible here and are manipulated in the same manner as the application-specific vocabularies. With the Vocabulary Manager, the user may import and export vocabularies or parts of vocabularies in order to share commands and thus enable speech recognition in applications not supported by default in DSRS.

Finite State Grammar

The finite state grammar (FSG) is a state machine with all the vocabulary required to transition between states and conditions. The FSG has two distinct sets of vocabulary, which have already been mentioned: the Always Active, or global vocabulary, and the Active, or context-specific, vocabulary.

In creating the FSG, we found that we needed special functions for interaction with the windowing system and representations for all keyboard keys. While creating these special functions, we designed the interaction for maximum convenience. For example, when a user speaks the phrase "go to calculator" or "switch to calculator" or simply "calculator," the meaning is readily interpreted by the software. For the user's convenience, these phrases trigger the following conditional actions.

- If a window of class "calculator" is present on the system, then set focus to it. This is done regardless of its state; the window may be in an icon state, hidden, or on another work space such as may be found in the Common Desktop Environment (CDE).
- If the window does not exist, then create one by launching the application.

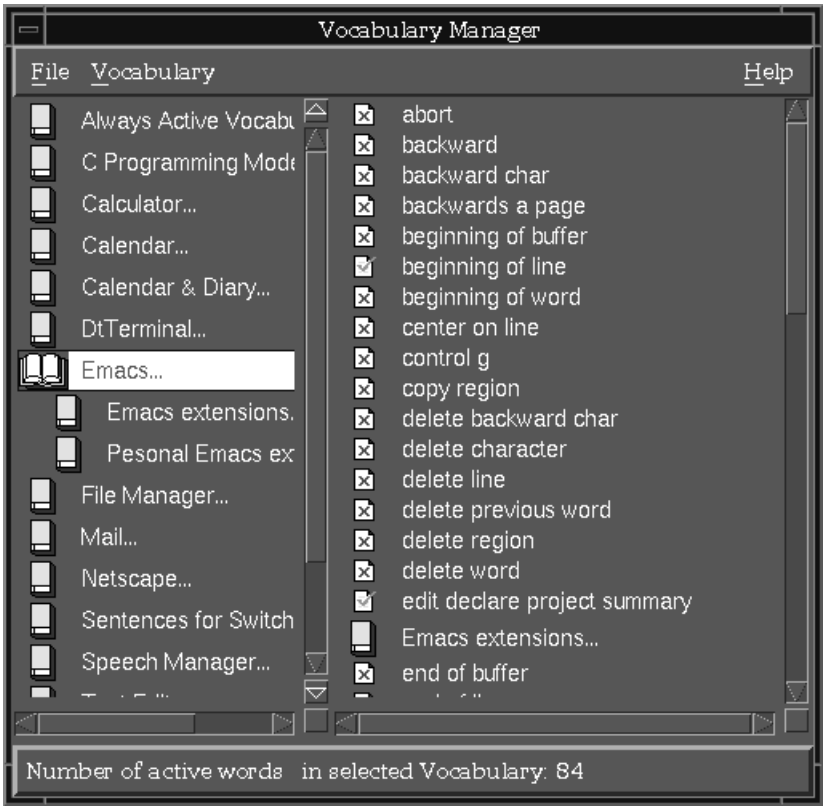

Figure 4
Vocabulary Manager Window

The simple logic of this special function enhances user productivity. Often workstation and PC screens are littered with windows or applications icons and icon boxes through which the user must search. Speech control eliminates the steps between the user thinking "I want the calculator" and the application being presented in focus, ready to be used. The DSRS team created a function called FocusOrLaunch, which implements the behavior described above. The function is encoded into the FSG continuous-switching-mode sentences in the Always Active vocabulary associated with the spoken commands "switch to <application name>," "go to <application name>," and just plain "<application name>."

Applications like calculator and calendar are not likely to be needed in multiple instances. However, applications such as terminal emulator windows are. DSRS defines the specific phrase "bring up <application name>" to explicitly launch a new instance of the application; that is, the phrase "bring up <application name>" is tied to a function named Launch.

The phrases "next <application name>" and "previous <application name>" were chosen for navigating between instances of the same application. DSRS remembers the previous state of the application. For instance, if the calendar application is minimized when the user says "switch to calendar," the calendar window is restored. When the user says "switch to emacs," the calendar is returned to its former state. In this case, it is minimized.

DSRS also adds speech control to the common window controls such as minimize, maximize, and close. These functions operate on whatever window is currently in focus.

Another convenient command is "Speech Manager go to sleep." When the user speaks this command, DSRS transitions into a special standby state. In this state, termed "sleeping," the recognizer is still listening but will return to command and control mode only when the command "Speech Manager wake up" is spoken. The "go to sleep" command puts DSRS into a standby state, allowing normal conversation to take place without words being recognized as commands and causing unwanted events to occur.

Version 1.1 of DSRS adds even more functions, such as the "microphone off" command, which goes a step beyond "go to sleep." With "microphone off," the input audio section is completely released and DSRS will no longer listen until the microphone is manually turned back on. This function allows the

user to launch an audio-based application that will record, such as a teleconferencing session. Version 1.1 also includes a function that allows the user to play a "wave," or digitized audio clip. Audio cues may thus be played as part of speech macros. The "say" command invokes DECtalk Text-to-Speech functionality so that audio events can be spoken.[4]

Since speech recognition is a statistical process and prone to errors, the design team deemed "confirm" an important function to protect user data and prevent unwanted actions. The "confirm" function protects certain sensitive actions, such as exiting an editor, with a confirmation dialog box. Simply adding the "confirm" syntax within a speech macro causes the dialog box "are you sure?" to appear. The vocabulary is switched to respond to only yes and no so that a higher reliability can be achieved. If the user says no or presses the no button, the computer returns to its previous state. If the user says yes, the action following the "confirm" function is executed.

Another concept encoded in the FSG for user convenience is menu flattening. Menu displays are hierarchical because the number of menu entries that can be shown on the screen at one time is limited. A good example is the File menu. When the user clicks the mouse button on File, a drop-down menu appears containing actions such as Open file, Save file, Save file as …, Print, and Exit. However, hierarchical menus do not really represent the way people normally think about actions; for example, when the user thinks "exit," he or she must then take the steps file and exit. With speech recognition, the computer can take the interim steps. The FSG in DSRS was built to handle two cases: (1) The user says "file" and "exit," and (2) the user says only "exit" and DSRS performs the file and exit sequence transparently. This second mode connects the actions more closely with the user's thought processes and does not force a sequence of actions in order for tasks to be performed. The menu-flattening feature of DSRS was encoded into the FSG file. While the example given may seem trivial, the concept is an important one and can be used to flatten many levels of menus. For instance, users take several steps to change the font or type size on a region of highlighted text in a word processing program. The following could conceivably be invoked as a speech macro: "Change to Helvetica Bold Italic 24 points."

Integrating Speech Recognition in Applications

As described in the section Overview, DSRS feeds keystrokes to applications. Therefore, the preferred application development method for allowing access to functions—one that will allow integration of speech recognition—is accelerator keys. Typically, accelerator keys are in the form of CTRL + <key> bindings that allow direct access to a function, regardless of menu hierarchies. It should be noted that this lack of hierarchy limits the number of directly accessible functions.

A second method for integrating speech within an application is through menu mnemonics. Mnemonics are the keyboard equivalents signified in application menus by an underlined letter. The first mnemonic is invoked by a combination of the ALT key and the underlined letter, which can be followed by another underlined letter. For example, pressing ALT + f invokes the file menu item; pressing x immediately thereafter invokes the "exit" entry for the application.

Integrating speech recognition becomes difficult when application functions are not accessible through the keyboard. Applications designed to allow access to functions only by means of the mouse cannot be speech enabled as DSRS is currently implemented. Although DSRS can send mouse clicks into the system, consistently locating the mouse pointer on applications is difficult. The next sections further illustrate the issues that stemmed from these integration issues as we implemented and tested DSRS.

Client–Server Protocols

Applications such as emacs and Netscape Navigator have protocols that allow other processes to send commands to them. For example, a file name or a universal resource locator (URL) may be sent via the command line. DSRS exploits this facility in Netscape Navigator to allow Web surfing by voice. For example, in the Netscape context, the speech macro "Digital home page" would translate to the following command issued to a window: netscape-remote openURL("http://www.digital.com"). Although this command string seems a bit awkward, the result is that the actions being taken are all transparent to the user and they work very well.

Problems Encountered in Implementation

Unlike the applications discussed in this paper, some applications are not developed with good programming practices. Neither are the keyboard interfaces well-tested. We encountered the following types of problems when using the keyboard as the main input mechanism.

- Applications had multiple menu mnemonics mapped to the same key sequence. This approach could not work even if the keyboard were used directly.
- Application functions controlled by graphic buttons were accessible only by mouse.
- Keyboard mapping was incomplete, that is, mnemonics were only partially implemented.

In the implementation of DSRS, we encountered one unexpected problem. When a nested menu mnemonic was invoked, the second character was lost. The sequence of events was as follows:

- A spoken word was recognized, and keystrokes were sent to the keyboard buffer.
- The first character, ALT + <key>, acted normally and caused a pop-up menu to display.
- The menu remained on display, and the last key was lost.

We determined that the second keystroke was being delivered to the application before the pop-up menu was displayed. Therefore, at the time the key was pressed, it did not yet have meaning to the application. It is apparent that such applications are written for a human reaction–based paradigm. DSRS, on the other hand, is typing on behalf of the user at computer speeds and is not waiting for the pop-up menu to display before entering the next key.

To overcome this problem, we developed a synchronizing function. Normally the Vocabulary Manager notation to send an ALT + f followed by an x would be ALT + f x. This new synchronizing function was designated as sALT + f x. The synchronizing function sends the ALT + f and then monitors events for a map-notify message indicating that the pop-up menu has been written to the screen. The character following ALT + f is then sent, in this case, the x. The synchronizing function also has a watchdog timer to prevent a hang in the event a map-notify message. This method is included in the final product.

Guidelines for Writing Speech-friendly Applications

Several guidelines for enabling speech recognition in applications became apparent as we gained experience using DSRS. Coincidentally, a guideline recently published by Microsoft Corporation documents some of the very same points.[5]

- Provide keyboard access to all features.
- Provide access keys for all menu items and controls.
- Fully document the keyboard user interface.
- Whenever possible, use accelerator keys; they are more reliable than using menu mnemonics. Mnemonics can be overloaded or non-functional if the menu is not active.
- Client–server protocols can work well for enabling speech recognition; document fully.
- Do not depend on human reaction times for displayed windows or on slow typing rates.
- Provide user-friendly titles for all windows, even if the title is not visible.
- Avoid triggering actions or messages by mouse pointer location.
- Give dialog boxes consistent keyboard access; for instance, boxes should close when the ESC key is pressed. The dialog box responses yes and no should correspond to the y and n keys.

Application developers who wish to design a speech interface directly into their applications now have this option. Several speech APIs are available. Microsoft offers the Speech Software Development Kit, and the Speech Recognition API Committee, chaired by Novell, offers SRAPI. Computer–human speech interaction is the subject of ongoing research. Much of the government-sponsored research is now being commercialized. Several groups, such as ACM CHI,[6] have been and continue to study speech-only interfaces. They are discovering that "translating a graphical interface into speech is not likely to produce an effective interface. The design of the Speech User Interface must be a separate effort that involves studying the human–human conversations in the application domain." [6]

Future Directions for Speech Recognition

In addition to uncovering points for developers to build speech-enabled applications, we also gained a perspective on how speech recognition may develop in the future. A brief overview of these insights is presented in this section.

Integrating speech and audio output—The addition of a two-way interface of speech and audio that gives users feedback will move the user interface to a new level.

Telephone access—Telephone access can make workstations more valuable communications devices by connecting users to information such as e-mail messages and appointment calendars. The telephone can extend the reach of our desktop computers.[6]

Dictation—Discrete dictation products with capabilities of 60,000 words are commercially available now; in the not-too-distant future, continuous-recognition dictation products will become viable. A command and control recognizer that can be seamlessly switched to dictation mode is a very powerful tool.

Speech recognition integrated with natural language processing—The field of natural language processing deals with the extraction of semantic information contained in a sentence. Machine understanding of natural language is an obvious next step. Users will be able to speak in a less restricted fashion and still have their desired actions carried out.

A new paradigm for applications—A new class of applications needs to be created, one that is modeled more on human thought processes and natural language expression than on the functional partitioning

in today's applications. A user agent or secretary program that could process common requests delivered entirely by speech is not out of reach even with the technology available today, for example:

User: What time is it?
Computer: It is now 1:30 p.m.

User: Do I have any meetings today?
Computer: Staff meeting is ten o'clock to twelve o'clock in the corner conference room.

Computer: Mike Jones is calling on the phone. Would you like to answer or transfer the call to voice mail?
User: Answer it.

User: Do I have any new mail?
Computer: Yes, two messages. One is from Paul Jones, the other from your boss.
User: Read message two.

User: What is the price of Digital stock?
Computer: Digital stock is at $72 1/2, up 1/4.

The example above shows the user agent providing information and interacting with e-mail, telephone, stock quote, and calendar programs. As we move into the future, the computer–user interface should move closer to the interaction model humans use to communicate with each other. Speech recognition and text-to-speech software help in a significant way to move in this direction.[6]

Performance

DSRS word recognition, which is the primary performance measure, is as good as comparable command and control recognizers found on PCs. Training troublesome and acoustically similar words improves the performance. The vocabulary, because of the targets chosen, that is, UNIX commands, does have acoustic collisions, for example, escape and Netscape. Further, we had to use the vocabularies supporting the UNIX shell commands, and commands such as vi can be pronounced in different ways, for example, vee-eye or vie. The shell commands are also full of very short utterances that tend to result in higher error rates.

On the slower, first-generation Alpha workstations, DSRS has noticeable delays, on the order of a few hundred milliseconds. However, on the newer and faster Alpha workstations, DSRS responds within human perceptual limits, less than 100 milliseconds.

Another interesting phenomenon associated with the speed of the workstation is the improvement DSRS makes in user productivity. On a slow machine, the speech interface has little impact if the application is slow in performing its tasks. In other words, the time it takes to perform a certain task is not greatly affected unless the human input of commands is a significant portion of that time. However on a fast machine, the application performs tasks as quickly as the commands are spoken, and the productivity enhancement, therefore, is great.

Summary and Conclusions

The DSRS team accomplished its objective of developing a low-cost speech recognition product. DSRS for Digital UNIX is being shipped with all Alpha workstations at no additional cost. Integration with the X Window System was successful.

With reference to the focus of this paper—developing the user-friendly interface—we found through feedback from our user base that most first-time users perform useful work using DSRS without consulting the documentation. The first-time setup design that provides instructions and feedback to users was successful. The list of Active and Always Active words and phrases is a helpful reference for new users until they learn the "language" they can use to communicate with their applications.

Adding vocabulary for new applications is a bit more challenging because some "reverse engineering" may be required on a particular application. One needs to know the class name of each of the windows and then map the keystrokes for each of the functions to speech macros. Although this procedure is documented in the manual, it can be challenging for users.

Prototypes of DSRS control for sophisticated menu-driven applications, such as mechanical computer-aided design, show excellent promise for enhancing user productivity. For example, with computer-aided design or drafting software, users can focus their eyes on the drawing target on the screen while they are speaking menu functions.

Speech recognition is an evolutionary step in the overall computer–user interface. It is not a replacement for the keyboard and mouse and should be used to complement these devices. Speech recognition works as an interface because it allows a more direct connection between the human thought processes and the applications.

Speech recognition coupled with natural language processing, text-to-speech, and a new generation of applications will make computers more accessible to people by making them easier to use and understand.

Acknowledgments

Thanks go to the dedicated team of engineers who developed this product: Krishna Mangipudi, Darrell Stam, Alex Doohovskoy, Bill Hallahan, and Bill Scarborough, and to Dragon Systems, Inc. for being a cooperative business and engineering partner.

References

1. L. Rabiner and B. Juang, *Fundamentals of Speech Recognition* (Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1993): 45–46.
2. C. Schmandt, *Voice Communication with Computers: Conversational Systems* (New York, N.Y.: Van Nostrand Reinhold, 1994): 144–145.
3. K.F. Lee, *Large-Vocabulary Speaker-Independent Continuous Speech Recognition: The SPHINX System* (Pittsburgh, Pa.: Carnegie-Mellon University Computer Science Department, April 1988).
4. W. Hallahan, "DECtalk Software: Text-to-Speech Technology and Implementation," *Digital Technical Journal,* vol. 7, no. 4 (1995): 5–19.
5. G. Lowney, *The Microsoft Windows Guidelines for Accessible Software Design* (Redmond, Wash.: Microsoft Development Library, 1995): 3–4.
6. N. Yankelovich, G. Levow, and M. Marx, "Designing SpeechActs: Issues in Speech User Interfaces," *Proceedings of ACM Conference on Computer–Human Interaction (CHI) '95: Human Factors in Computing Systems: Mosaic of Creativity,* Denver, Colo. (May 1995): 369–376.

Biography

Bernard A. Rozmovits
During his tenure at Digital, Bernie Rozmovits has worked on both sides of computer engineering—hardware and software. Currently he manages Speech Services in Digital's Light and Sound Software Group, which developed the user interfaces for Digital's Speech Recognition Software and also developed the DECtalk software product. Prior to joining this software effort, he focused on hardware engineering in the Computer Special Systems Group and was the architect for voice-processing platforms in the Image, Voice and Video Group. Bernie received a Diplôme D'Étude Collègiale (DEC) from Dawson College, Montreal, Quebec, Canada, in 1974. He holds a patent entitled "Data Format For Packets Of Information," U.S. Patent No. 5,317,719.