MAS.632 Conversational Computer Systems
Fall 2008

8

Using Speech Recognition

The previous chapter offered an overview of speech recognition techniques and the difficulty of identifying the words in a speech signal. This chapter discusses how speech recognition can be used in applications. It examines the merits as well as the constraints of using voice input, which limit the classes of applications for which it is suitable. Because the dominant difficulty with using speech recognition devices is usually their poor performance, this chapter emphasizes interaction techniques to manage recognition errors. Finally, several case studies in applications of voice input are detailed.

USES OF VOICE INPUT

Voice input can be suitable to an application for one of several reasons. In some circumstances, no other input channel may be available due to the lack of an interface device such as a keyboard or mouse. Even when alternative input devices are available, speech recognition may be more efficient; we can speak faster than we write or type. Voice can also be employed effectively when the user's hands and eyes are occupied with another task.

Sole Input Channel

Regardless of its limitations, voice input is useful when no other input mode is available or when alternative modes present only limited functionality. For this reason, speech recognition currently attracts wide interest from the manually-

disabled population for whom conventional keyboards and mice are extremely demanding.

For many telephone-based voice applications such as those described in Chapter 6, the 12 button keypad is sufficient. Speech recognition can be employed instead of or in addition to touch tones to provide access to a greater number of commands, to make it easier for the caller to remember them, and to enable users to phone in from rotary-dial telephones. Although speech recognition provides a more flexible interface, it is also significantly limited in this context. Because of the limited bandwidth of the telephone audio channel, both high and low frequency sound is lost, removing some cues useful to acoustic classification. Although the quality of telephone networks is improving, some calls, such as from cellular phones, contain significant noise. Even when the telephone circuit itself introduces no noise, users of telephone-based services are often travelers calling from public telephones in noisy locations. Also, as discussed in Chapter 6, because the telephone circuit mixes both the transmitted and received speech it is difficult for the user to interrupt and be correctly recognized.

Speech recognition has been used in a number of telephone-based services usually employing small vocabulary speaker-independent recognition because the identity of the caller is not known. An alternative approach uses speaker-independent recognition or touchtones for the caller to "login" and then loads that caller's personal vocabulary into the recognizer for the remainder of the call. Some examples of telephone-based recognition are listed here.

- Telephone-based financial services. A caller enters an account number by voice and is told the current trading level of stocks in his or her portfolio or speaks code numbers for a specific stock to hear its price. Because speaker-independent recognition is often limited to the digits zero through nine plus "yes" and "no," it has not been possible until recently to use a more convenient interface that allows the user to speak stock names.
- Automatic directory assistance for pay-per-use telephone services. Voice menus list the types of services one can call, and the user selects one by speaking choices while traversing the menu. The application then gives a telephone number for access to the chosen service.
- Automated collect call completion. The calling party is prompted to record his or her name. A computer phones the called party, announces the requested collect call, plays the recording of the caller identifying herself, and then asks whether the charges will be accepted. The words "yes" and "no" are recognized; failed recognition results in transfer to a human operator for confirmation.

Speech recognition has enormous potential as the primary means of interacting with new generations of very small hand-held computers. Improvements to laptop computers are mostly in performance; a full keyboard suitable for touch typing places limits on further size reductions. A number of "palm top" computers provide applications such as a calendar and telephone directory but are limited in the amount of data input allowed by their tiny keys. An all-voice

computer, perhaps including a small touch-sensitive display, could be built in a package similar to a miniature tape recorder. At the time this book is being written several manufacturers have introduced families of integrated circuits for digital consumer answering machines; these inexpensive parts are also suitable for a pocket computer.

Auxiliary Input Channel

Speech recognition is useful as an input channel even when a keyboard is available, i.e., when speech is not the only possible means of interacting with a computer.[1]

Limited vocabulary speech recognition has been used for a variety of applications in which it essentially substitutes for the keyboard; for a survey of such comparisons see [Martin 1989]. In many traditional screen and keyboard applications such as authoring computer programs, speech recognition has not yet offered a great advantage over keyboard input. However, speech has proven effective in applications in which the user's hands and eyes are otherwise busy, such as baggage sorting, examining printed circuit boards for defects, and entering the results of visual analysis of specimens under a microscope. Speech input is attractive in such situations because of the inconvenience of putting down the object of one's attention for the sake of using the hands on a keyboard.

A particularly strong motivation for voice input is our ability to perform multiple tasks simultaneously if we can employ multiple input modalities. As discussed in some detail in Chapter 6, dividing tasks between verbal and manual channels can result in increased effectiveness and speed by a user. Voice input has particular importance for computer workstations because users are already using a keyboard and often a mouse for a variety of tasks. Computer users are usually occupied manually by typing or using the mouse and visually by the computer display. Although a good typist does not watch his or her hands, most computer tasks involve interacting with data, text, or an operating system for which visual feedback is essential. Because the mouse is used for *relative* positioning, it cannot be operated without attention to the screen. Computer users also often perform multiple tasks in parallel, especially if multiple phases of the same application, such as using a menu and typing data, are considered to be separate tasks.

Computer users, then, can take advantage of voice input for both reasons: being otherwise occupied manually and visually and performing multiple tasks in parallel. In Chapter 6 the same arguments were made to justify potential advantages of voice output. Martin [Martin 1989] demonstrated that for a computer-aided design task involving extensive use of display, keyboard, and mouse, experienced users were able to get more of their tasks done while spending less time looking at the screen when voice input was added. These results motivated

[1] The "listening typewriter" is a special case of keyboard replacement which will be discussed in the next section.

Xspeak, a Media Lab interface using speech recognition for navigation in a window system, which is described as a case study later in this chapter.

Another potential use of voice as an auxiliary channel of user input is interacting with a computer at a small distance. A user might wish to start issuing commands such as logging in or initiating a time-consuming task such as sorting electronic mail while walking into the office and hanging up his or her coat. Unfortunately, few recognizers can perform adequately under the highly variable acoustic conditions occurring while the talker moves about in an office, but it may be feasible to electronically "point" an array of microphones at the mobile sound source.

Voice input can also be found in commercial products to speed-dial telephone numbers. The user trains templates and enters the associated digits using the telephone keypad. The user later places a call by picking up the handset and speaking the name to call. Since the caller is about to speak into the telephone during the call, voice-dialing is an intuitive extension of the voice modality. For speech recognition, the telephone handset microphone is well positioned with respect to the mouth; picking it up provides the cue to begin recognizing.

Keyboard Replacement

The previous section argued that there are situations in which speech recognition may be a useful supplement to the keyboard because of other activities a user needs to perform manually. This section considers the potential of speech recognition as a direct keyboard replacement for the purpose of text entry. Although dictation is in many aspects the antithesis of conversational interactive systems, such a "listening typewriter" is the subject of much speech recognition research and the focus of several attempts to develop recognition products.

A listening typewriter may offer several advantages over a keyboard. We can speak significantly faster than we can write or type; we usually speak at several hundred words per minute; an excellent typist can produce 80 or 100 words per minute, and we handwrite at less than half that rate. Of course many of us are not efficient typists, and some professionals are quite adverse to using a keyboard. Even efficient typists may suffer from physical disabilities that interfere with productivity.

At first glance, speech recognition seems to be a superb keyboard replacement. We could imagine such a device useful in any number of large vocabulary computer applications, but we must consider a number of factors in assessing the utility of the listening typewriter. Can speech recognition devices take full advantage of the faster rates of speaking over writing? What portion of the time needed to create a document is consumed by the text input stage? How difficult is it to learn to dictate, and are dictated documents of comparable quality to written ones? Finally, just how extensible is the listening typewriter?

Large vocabulary speech recognition by machine is extremely difficult. As the number of words stored in memory increases, there is an increased likelihood that words will be confused, and pattern matching to select which word was spoken will take correspondingly longer as well. Because a large number of words

may be spoken at any time, word boundary detection is also confounded: As a result many very large vocabulary systems are being developed using isolated word recognition.

Although there are several products and a number of research projects offering some degree of large vocabulary recognition, this work is not yet mature enough to allow an assessment of speech effectiveness for text entry using these devices. The number of similar sounding words confounds recognition in large vocabularies. As a result, current recognizers operate either in less than real time or are constrained to isolated word recognition. But insights into the utility of speech for text input can be gained both from simulations in which a hidden human performs the voice-to-text transcription in real time and by studying how people dictate. Much of the research in this area was done in a series of experiments by John Gould at IBM, who studied dictation as well as simulated listening typewriters.

In one study focusing on the difference between speaking and writing letters, Gould found that speaking required 35% to 75% of the time required for writing and that across both media about two-thirds of the time was spent in planning [Gould 82]. In another study, Gould determined that although people thought their written letters were of better quality, independent judgments did not show this to be true [Gould and Boies 1978]. This work and a similar study [Gould 1978] suggested that dictation is not a difficult skill to acquire.

Gould then provided a simulated listening typewriter to subjects who used it to dictate letters [Gould *et al.* 1983]. He wished to evaluate the importance of vocabulary size as well as the constraints of discrete versus connected speech input. A human transcriber behind the scenes played the role of the speech recognizer and enforced interword pauses for discrete input mode. Depending on the selected vocabulary size, the transcription software determined whether each word should be "recognized." The subject was required to spell an unrecognized word letter by letter. The subjects included both expert as well as naive dictators.

Gould measured the composition time and evaluated the quality of the letters produced by his subjects who used the simulated recognizer. He found that by both criteria speaking letters was comparable to hand writing them. The fact that expert subjects (experienced with dictation) were not significantly faster than novices suggests that the experimental setup was not as efficient as simple dictation; using any of the simulated technologies, someone talking to a computer was slower than speaking into a tape recorder.

Although Gould's subjects preferred connected over discrete speech, the difference was not as large as one might expect given the requirement to pause between each word. This may be due in part to the nature of the experiment. In one task, subjects were required to write a final draft version of a letter. If the recognizer made an error, the user's only recourse was to back up and delete each intervening word, replace the misrecognized word and then repeat all the deleted words. This may have encouraged subjects to pause between words until each recognition result was displayed on the screen, even while using the connected speech mode. This may also partially explain why expert dictators did not perform as well as might have been expected, given their experience.

What do Gould's studies indicate as to the suitability of speech as a keyboard replacement? It seems easy to learn to input text by voice, and the resulting documents are of equal quality to those written or typed. Although isolated word recognition is significantly slower than connected recognition, it does appear to have utility and may be acceptable in an office environment. But there is more to writing a passage than entering words. Although speaking may be five times faster than handwriting and two to three times faster than an experienced typist, *composing* a letter takes a significant portion of the time spent creating a document, thus diminishing some of the speed improvement afforded by voice input. After initial authoring, documents may require editing, which Gould noted is easier with handwriting. Some editing functions such as moving a cursor to select a region of text may be awkward to perform by voice, and it is a challenge for the recognizer to detect when the user wants to switch between text input and editing.[2] Finally, it must be noted that although Gould's studies compared voice to handwriting, typing is becoming an increasingly common skill and is much faster than handwriting (although still much slower than fluent speech).

Despite these objections, the listening typewriter would still undoubtedly be useful for some users and in some work environments. Research in this direction hastens the arrival of large vocabulary general purpose recognizers as well. However, specific assumptions about the structure of written language used to constrain the recognition task may render the recognizer less useful for other potential applications. For example, the language, or sequences of words, that a user would speak to a computer operating system would possess its own unique syntax very different from the business correspondence used to seed the trigram probability tables. In addition, operations such as selecting a file name or entering the name of a new file to be created much less constrain the choice of the next word than natural languages and therefore are harder to recognize. The terseness of text-based computer interfaces is efficient for the skilled user, but by removing redundancy they comprise a more difficult language for recognition.

This section has considered the production of text documents by voice in contrast to keyboard or handwritten input. Voice as a document type was considered in depth in Chapter 4. Although dictation and using a speech recognizer instead of a keyboard are similar tasks, both involve quite different authoring styles than recording short voice memos that are never converted to text. Comparisons of creation time with quality across voice and text documents may not be very meaningful as the document types are most likely to be used for rather different styles of messages.

[2]To achieve acceptable rates of recognition with such large vocabularies, recognizers must take advantage of the context in which the word occurs. The requirements imposed by assumptions of context, both preceding and following the word just spoken, interfere with providing immediate editing commands. For example, the Tangora recognizer described in the previous chapter does not report a word as recognized until at least one additional word has been spoken to afford adequate context for effective recognition.

SPEECH RECOGNITION ERRORS

The primary limitation of speech recognition is the accuracy of current recognizers. As a result, coping with errors dominates user interaction techniques. The difficulty of recognizing speech further limits the range of applications for which it is suitable. Accuracy is so impaired by noise that speech recognition can be used only with great difficulty in even moderately noisy environments or over impaired communication channels such as cellular telephone connections. Speech recognition applications must operate with smaller vocabularies, making it difficult to support broad functionality, in order to attain acceptable recognition rates. Even the most simple application must deal with recognition errors; for complex applications error detection and correction may require more programming than the portion of the application that actually performs the user's suggested action.

Classes of Recognition Errors

Speech recognition errors may be grouped into three classes.

1. *Rejection:* The talker speaks a word from the recognizer's vocabulary but it is not recognized.
2. *Substitution:* A word spoken from the vocabulary is recognized as some other word.
3. *Insertion:* Extraneous words are recognized in response to stimuli such as the talker's breathing or lip smacking or perhaps environmental noise such as a door slam or keyboard clicks.

Different strategies are required for coping with each type of error. All errors are more likely to occur when using connected recognition, large vocabularies, and speaker-independent devices. Because of rejection and insertion errors, a connected recognizer may even report incorrectly the *number* of words spoken in addition to making a substitution error on any particular word.

Users want the best recognizer they can buy; however, it is very difficult to measure error rates of a particular device. Error rates will be a function of the background noise level, the vocabulary set, the speaker's cooperation, attitude, and experience. Most manufacturers claim recognition accuracy in the high 90% range, but these rates are unrealistic and rarely realized in practice. Accuracy measurements across recognition devices are a meaningless basis for comparison unless the same test conditions are reproduced for each measurement. This requires using a standardized test, with both test and training data taken from the same audio source, e.g., a test tape or CD.

Real users rarely achieve the rates claimed for a device by the vendor. Reducing accuracy to a single metric can be misguided because rejection, substitution, and insertion errors may have different impacts on the underlying application. The difference between 95% accurate recognition and 98% may be very significant in terms of the difficulty in improving the recognition algorithm (to halve the error rate), but this reduction is much less significant for user interface design; in both cases, the recognizer is correct most of the time but makes occa-

sional errors. The speech input interface still has to cope with errors or it will have little utility.

Factors Influencing the Error Rate

Recognition errors are caused by many factors; some are more obvious than others. Techniques can be employed to minimize errors, so it is important to recognize early in a design what factors are likely to contribute to an increased error rate. Understanding the causes of errors also helps us appreciate why some applications are more amenable to speech input than others. The necessity of detecting and correcting errors precludes the use of recognition from some applications where it may seem ideal at first glance.

Word Similarity

Words which sound similar, such as "bat" and "bad," are difficult for humans to distinguish, so it should not be surprising that they are also frequently confused by a speech recognizer. Without a thorough understanding of the acoustic representation used by the recognizer, it is difficult to know which words sound similar, but a good first approximation is that words containing similar phoneme sequences are likely to be a source of recognition errors.

As previously mentioned, as the number of words to be recognized increases it becomes increasingly difficult to distinguish between them. Longer words contain more phonemes and hence are more likely to contain cues allowing them to be distinguished from other words. For example, the words "impossible" and "improbable" contain nearly identical sequences of phonemes, but the fricative "s" is a robust cue for differentiation. Errors can be minimized by careful selection of the vocabulary set to be used in an application, avoiding short and similar sounding words. When using a discrete word recognizer, it is helpful to string together short words such as "to the right of" into phrases which are trained and spoken as if they were single words.

But modifying one's vocabulary to optimize recognition performance may not be possible or may detract from the advantages of speech input. "Five" and "nine" could be confusing words for recognition because of their common vowel, but if the application requires the user to enter a telephone or credit card number, the use of these words cannot be avoided. Although "affirmative" and "negative" are more easily distinguished than "yes" and "no," they take longer to pronounce and more importantly are not part of most of our daily vocabularies. In situations in which speech recognition is essential, users may be amenable to using specialized vocabularies; however, this is at cross purposes to the claims of naturalness and ease of use for voice input.

Acoustic Environment

Noise interferes with recognition—although some recognizers are more sensitive to noise than others. Environmental noise varies widely; there are vast differ-

ences in the amplitude and spectral characteristics of the acoustical ambiance in an office, on a factory floor, or inside an automobile. In a noisy environment, special noise-canceling microphones may assist recognition (see Figure 8.1). These microphones attempt to remove background noise from the speech signal and are usually worn on the head or held near the mouth. This position is a hindrance in many circumstances.

Impairments in the acoustic channel may be constant or vary with time; constant characteristics are easier to model and hence are easier to compensate for. Speech recognition over the telephone provides an example of each. As discussed in Chapter 10, a telephone circuit transmits only the portion of the speech spectrum below approximately 3500 Hertz,[3] so a recognizer designed for telephone use must not rely on acoustic cues based on frequencies above this. This limited bandwidth is a predictable characteristic of the telephone channel. In addition, if a call is placed from a public telephone, there may be highly variable background noise such as traffic sounds, flight announcements, or the caller at an adjacent telephone. Because it is difficult to model such noise, it is also difficult to compensate for it.

[3]A manifestation of limited bandwidth is the increased difficulty we have in identifying a person by voice over the telephone.

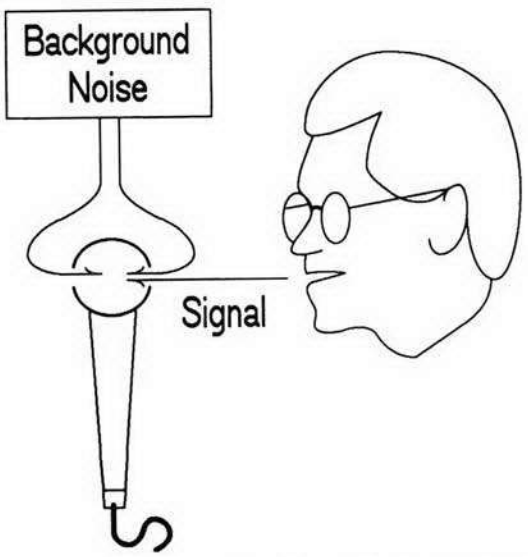

Figure 8.1. A noise-canceling microphone. Background noise enters the microphone in both the front and rear openings, but the user's speech is detected primarily at the nearer opening. This allows the microphone to subtract the background noise from the combined input of speech and background noise signal.

Talkers' Characteristics

Recognition simply works better for some people than others; this is what is sometimes referred to as the "sheep-and-goat" problem. Some users, the "sheep," can achieve high recognition accuracy. They are more cooperative and adaptive with their speech habits or perhaps simply more consistent in how they speak. Others, the "goats," have considerable difficulty with recognition. They may experience "microphone fright" or be resentful of the implications of a computer assisting them on the job. Many people compensate for errors during recognition by speaking louder, which may cause distortion by overloading the audio circuits, or by trying to speak more distinctly, which may make their speech less similar to the templates that they trained while relaxed. A user with a cold and congested nasal passages may need to train a new set of templates to achieve acceptable recognition due to the changed spectrum of the vocal tract transfer function.

INTERACTION TECHNIQUES

It is impossible to build an effective speech recognition application without taking care to minimize errors if possible and incorporating a means of coping with recognition errors. The previous section catalogued factors influencing the error rate; some of these may be at least somewhat under control of the application, such as utilizing a recognition vocabulary that avoids words that sound similar. This section also discusses other activity that can be incorporated into the user interface to diminish error probabilities. But errors will happen. This section emphasizes interaction techniques that can be employed by the user interface to allow applications to utilize recognition despite the errors.

Because recognition errors may be due to different causes (insertion, rejection, and substitution errors were just described) multiple error recovery techniques may be needed in a single application. There are two aspects of coping with errors. **Error detection** identifies that an error has occurred. Detection can be triggered by the user's apparently requesting an improper or meaningless action by the system or by the recognizer's reporting that a rejection error has occurred. **Error correction** techniques can then be employed to rectify the error and allow the application to determine what the user actually requested. Recognition errors may not be readily detectable by an application. If the user says "Call Barry" but the recognizer reports "Call Harry," this semantically correct substitution error might easily pass unnoticed, so the application must provide for user-initiated error correction as well.

This section focuses on techniques, largely conversational, for understanding the user's spoken input. If an error is detected, what should a user interface do about it? If the user detects the error, what corrective action does the application provide? This section discusses techniques to minimize recognition errors based largely on the observations made above on factors contributing to errors. It then examines confirmation strategies whereby the user is informed of what the application is about to do and has an opportunity to veto any command triggered by

erroneous recognition. The final part of this section presents methods whereby either the application or the user may invoke iterative techniques to detect and correct recognition errors from input command sequences already in progress.

Minimizing Errors

Recognition errors result in impaired application performance and increased user frustration. Whenever a recognition error occurs, a repair strategy must be invoked by either the user or the application. Repair inevitably necessitates the user speaking more words, which may in turn be misrecognized in addition to the error that the dialogue is meant to correct. As the error rate increases, it takes correspondingly longer to complete whatever task is supported by the application, and user satisfaction with speech recognition plummets. Error rates can be diminished by compensating for the factors resulting in errors; several factors were discussed in the previous section:

- Careful selection of the recognition vocabulary.
- Control of environmental acoustics and the use of a head-mounted noise-canceling microphone.
- Training users to speak clearly and consistently.

Several additional techniques can be used to further minimize recognition errors.

Vocabulary Subsetting

Many recognizers allow the application to specify that only a subset of the full vocabulary should be available for recognition at a particular moment. This dynamic modification of vocabulary size can result in significant performance improvements if the structure of the task permits. Many speech recognition applications use speech input much like filling out a multiple choice form; at any juncture in the user interaction there are a small number of choices of what to say next. For example, if the user is asked to speak a telephone number, recognition can be restricted to the digits plus a few key words, such as "area code" and "quit." If the application asks a yes-or-no question, the active vocabulary can be reduced to just the two possible responses.

Language Model Subsetting

Most current large vocabulary speech recognizers employ language models to improve accuracy by specifying all possible input sentences prior to recognition. This is similar in concept to vocabulary subsetting where the subsetting is done on a word-by-word basis while attempting to recognize a sequence of words. The application can facilitate this process by activating and deactivating portions of the language model (sometimes called "contexts") during the course of interaction. For example, at the stage of a transaction where the user is expected to

speak a dollar amount, the recognizer can be told to use only the small portion of its language model dealing with dollars, cents, digits, etc.

Explicit Indication of Attention

In many acoustic environments background noise is highly variable, which results in spurious insertion errors when short, loud sounds are picked up by the microphone. Insertion errors also occur if the user speaks with another person such as a telephone caller or a visitor. Such errors can be minimized by disabling recognition when it is not in active use.

One simple means of disabling recognition is the addition of a mechanical switch on the microphone cable. A hand-held device employing speech recognition may require a push-to-talk button in a noisy environment. If the user is at a computer, a mouse button may be used the same way. But any mechanical switch detracts from the benefit of being able to use recognition while one's hands are otherwise occupied.

Another alternative is to use voice commands such as "stop listening" and "pay attention." After the user says the first of these commands, recognition results are discarded until the second is spoken. Note that it is important to use phrases for this software switch that are unlikely to be spuriously recognized themselves. Another disadvantage of the stop listening approach is that the user is likely to forget that recognition is disabled until speaking several times with no apparent results.

Some versions of CMU's Sphinx recognizer used a key word to indicate attention and trigger recognition: "Computer, when is my meeting?" "Computer, cancel the meeting." Recognition control can also be implemented using the keyboard or mouse if the user works at a computer console. Finally, it is possible to use the direction in which the user speaks as the focus mechanism. The Media Lab's Conversational Desktop project [Schmandt *et al.* 1985] used such an approach; the user turned toward the screen when he or she wished to be heard, and an array of microphones detected this change of attention.

Confirmation Strategies

The need to cope with the high likelihood of errors must play a lead role in the choice of interaction techniques. The interface designer must consider the consequences of the application performing an incorrect action on the basis of misunderstood input and how to improve the chance that the user will notice the error before doing further damage. By setting a high rejection threshold, an interface designer can decrease the number of misrecognized words but at the cost of an increase in rejection errors. The user can be more confident that the application will not err, but it will likely take longer to perform the correct action.

Whether it is faster to reject many possibly correct utterances or accept errors and allow the user to correct them depends on the nature of the task, the complexity of the input, and the time required for recognition. Depending on the prob-

abilities of making errors with particular vocabulary sets, different correction or confirmation techniques can be modeled as Markov processes and their behavior predicted [Rudnicky and Hauptmann 1991]. But for some applications an error cannot be repaired. This is the case when the computer is going to perform some action with dangerous or irrevocable side effects (launching a missile or deleting a file). Such performance errors can be avoided by offering the user a chance to cancel a misrecognized request before the application has acted on it. At the cost of requiring greater participation from the user, several different confirmation strategies can be invoked to avoid making a mistake.

Explicit Confirmation

When misrecognition would cause an irrevocable performance error, explicit confirmation may be required from the user, elicited by echoing the request as recognized: "Please confirm that you wish to delete the file named 'book'." If the user's response is then limited to "yes" or "no," vocabulary subsetting can be employed to decrease the chance of misrecognition. In addition, the rejection threshold can be increased for the reply for added confidence that the user really wishes to proceed with a dangerous action.

Explicit confirmation is slow and awkward, requiring responses and user feedback with each interaction. As described here, it offers the user little recourse for misrecognition except the opportunity to try again without causing damage. Hence, it should be used sparingly. If the consequences of system action and recognizer accuracy together suggest use of explicit confirmation on every user request, the task is likely to be ill suited to speech recognition.

A more flexible variation is to use explicit confirmation only when the user's choice is implausible or the recognition score for a word is low. In such circumstances an effective technique is to ask "Did you say . . . ?" and temporarily switch to explicit confirmation mode.

Implicit Confirmation

An alternative to explicit confirmation is *implicit confirmation,* in which the application tells the user what it is about to do, pauses, and then proceeds to perform the requested action. The application listens during the pause, thus giving the user the opportunity to cancel the request before any action occurs. If the request takes significant time to perform, such as dialing a number from a cellular phone, the application may be able to begin performing the task during the pause; if the user cancels, the call attempt is abandoned before the call has had time to go through.

Implicit confirmation allows the system to perform more effectively as recognition accuracy improves. The user is required to intervene only in the case of an error; otherwise the request and resulting action proceed smoothly. Implicit confirmation could be slightly slower than explicit confirmation as the application must pause long enough to provide ample time for the user to respond. The user can be allowed to terminate the pause by saying "OK"; this is similar to the

optional termination techniques discussed in Chapter 6. If the user always responds, this is identical to explicit termination. Although the application is idle during this pause, the user can be performing other tasks and thinking about what to do next. Because the application has confirmed what it is about to do, the user need not devote additional attention to the task.

Error Correction

The previous section described confirmation techniques to ensure that erroneous recognition did not occur or was canceled by the user. This was based on a binary judgment by the user as to whether recognition was correct; rejection caused the entire input to be discarded. While this may be necessary in error-critical situations, a more effective strategy is to attempt to detect recognition errors and invoke repair dialogue to resolve misrecognized words.

An error may be noticed by the application or it may escape detection only to be noticed by the user. The application may detect that an error occurred because the resulting sentence is syntactically ill formed or semantically meaningless; techniques for such analysis are discussed in Chapter 9. Or it may be that an apparently well-formed request is meaningless in the current state of the application and its associated data. For example, consider a spreadsheet in which the rows are numbered and the columns are lettered. "Add row five and row B" is a syntactically well-formed sentence in and of itself but does not describe an operation that can be performed on this spreadsheet. "Add row five and row twelve" could be a valid operation, but if row twelve has not yet had values assigned to it, the operation still cannot be performed. An application supports a more limited set of operations than can be described with the associated vocabulary, and the current state of the application further constrains acceptable user input.

Depending on the class of recognition error (rejection, insertion, or substitution), the application may be faced with an incomplete utterance, a meaningless utterance, or a valid request that is unfortunately not what the user asked. If the application detects the error, how can it determine what the user wants? If the user detects the error, how can the application allow the user to make repairs?

Additional Recognition Information

In addition to identifying which word was detected, some recognizers supply additional information that can be used during error recovery. One such piece of useful information is the word that is the **next best match** to the spoken input. The similarity scores for each word indicate how reliable that choice is and how plausible the second choice is if the first is deemed to be in error. If the first choice is not meaningful in a particular context, the second choice may prove to be an acceptable alternative.

Some recognizers are capable of reporting the **confusability** of the words in its vocabulary. Confusability is a measure of how similar each word is to every other word in the vocabulary and gives an indication of the probability that any other

word will be substituted. If the recognized word is inappropriate in the current context but another word that is highly similar is appropriate, the application could use the alternate instead of the recognized word.

In the spreadsheet example, the phrase "row B" was suspect. Either second choice or confusability information could be used to indicate whether the user actually spoke "row three" or "column B," for example.

Alternate Input Modalities

In addition to speech, other input channels may be used to point or select the data represented by an application. Hauptman found that users were able to make effective use of pointing in combination with limited vocabulary speech recognizers to describe graphical manipulations [Hauptmann 1989]. Pointing devices may be two dimensional (mouse, stylus, touch screen, joystick) or three dimensional (gesture input using a variety of magnetic or ultrasonic techniques). Eye trackers report the angle of the user's gaze and can be used to manipulate an object by looking at it [Jacob 1991].

The selection gesture may be redundant to some words in a spoken command; this is especially true of deictic pronouns ("this," "that," "these," "those"). If the gesture can be correctly synchronized with the reporting of speech recognition results, the gestures can be used in place of any pronouns not recognized. This technique was first used in Put That There, a project described later in this chapter. Carnegie-Mellon University's Office Manager (OM) system used mouse-based pointing as a means of explicit error recovery [Rudnicky *et al.* 1991]; after recognition, a text display of the user's request was presented, and the user could click with the mouse on a word to be changed. Voice has been combined with gesture in a three-dimensional model building task [Weimer and Ganapathy 1992, Weimer and Ganapathy 1989]. Starker and Bolt combined speech *synthesis* and eye tracking into a system that discussed what a user was looking at [Starker and Bolt 1990]; and more recently the same research group has incorporated speech recognition as well [Koons *et al.* 1993].

Dialog With the User

The errors that an application detects result in a meaningless or incomplete request. The application must then employ techniques to elicit additional information from the user. The simplest but not very effective technique is to ask the user to repeat the request. If the user repeats the request word for word, it is quite possible that the same recognition error will occur again. If the user instead rephrases the request, different recognition errors may occur, and it is challenging to merge the two requests to decide what the user really wanted.

It is better for the application to ask the user one or more specific questions to resolve any input ambiguity. The phrasing of the questions is critical as the form in which a question is asked strongly influences the manner in which it is answered. The goal of efficiency suggests that the questions be brief and very direct. The goal of minimizing user frustration would also suggest asking ques-

tions that can easily be answered by a single word because words spoken in isolation are more easily recognized than connected speech. Consider a schedule application as an example. The user requests "Schedule a meeting with Rebecca tomorrow at three o'clock"; however, the word "three" is rejected. Because the application requires that a time be specified to complete a schedule request, the error can be detected. In return, the application could ask "At what time?"

Echoing

Asking a terse question is fast but conveys little information about the words that the application assumes to be correct. In the above example, if the application responded "What city?" the user would realize the application had probably completely misunderstood. But if instead of "Rebecca" the speech recognizer had detected "Erica," the query "At what time?" could not convey the occurrence of this error to the user.

Echoing is a technique to offer extensive feedback to the user. For example, the application might have asked "At what time do you wish to meet with Erica tomorrow?" so an alert user could notice the error. Even when the application receives an apparently correct command sequence, echoing it to the user provides an opportunity for the user to take further action. The major drawback of echoing is that it takes time and makes the application much more verbose.

Backtracking

When recognition errors result in a well-formed and plausible command, an application cannot detect the error and will perform the wrong action in the absence of a confirmation strategy. The user may recover from such an error by backtracking, in which the user simply cancels a request or asks the application to "undo" the action. This is appropriate only when the command can, in fact, be canceled with little or no penalty. To implement backtracking, the application must remember its state before each round of user input; the complexity of this memory is task dependent. Backtracking is a corrective behavior invoked entirely by the user but assisted by the application's ability to track its state.

CASE STUDIES

What do these factors lead us to conclude as to the suitability of speech recognition to real world work situations? Speech is fast and can be effective for text entry provided the recognizer performs adequately. Performance is enhanced by constraining the recognition choices at each word boundary, i.e., by minimizing the perplexity of the recognition task. This implies that recognition for free-form dictation is more difficult than for application-specific tasks.

Recognition is often suitable for tasks in which one's hands and eyes are busy, providing a valuable auxiliary channel for tasks such as computer-aided design. But note that recognition in this context is more suited to selection of one of a

small number of items from a list (e.g., a menu) or selecting an object from some number on display than for continuous tasks such as positioning a cursor.

Recognition is particularly difficult in noisy situations, such as in an automobile or on a factory floor; under these circumstances users may be forced to wear head-mounted microphones. Recognition may be possible in a relatively quiet office environment, but privacy concerns and social conventions make it unlikely we will talk to our computers if we share our office.

This chapter has discussed both the characteristics of applications that may make them successful candidates for speech recognition as well as interactive techniques required to allow users to cope with recognition errors. This section considers two case studies to illustrate these themes. The first case study examines Xspeak, which used rudimentary isolated word recognition to manage windows on a workstation display. Xspeak illustrated how commercially-available recognition may already perform adequately for deployment into normal office environments[4] and discusses experiences of real users incorporating recognition into their jobs.

The second case study, Put That There, demonstrates an early application of multiple error management techniques. Put That There employed connected speech recognition, and the nature of the application required very different error management than Xspeak. Put That There was one of the first speech user interfaces to merge both recognition and synthesis into a conversational system.

Xspeak: Window Management by Voice

This chapter has contrasted the large vocabulary listening typewriter with smaller vocabulary application-specific recognition. But few workstation-based applications use recognition, and the discussion of error handling techniques should make it clear that incorporating recognition into an application requires considerably more programming than simply reading from an additional device for input in addition to the mouse and keyboard. Furthermore, most desktop computers are now running window systems, allowing users to employ multiple applications during the same work session. If several applications use speech recognition, will they compete for the microphone? Recognition results must be distributed to the correct application process using some indication of the user's intention, such as knowledge about the vocabulary employed by each application, or which window is currently active.

Window systems are ubiquitous on computer workstations. The window system divides the screen into multiple regions ("windows"), each of which can be devoted to a single application. An operating system may allow multiple programs to execute simultaneously; the window system allows each of them to display output without disturbing the others. A typical workstation user is likely to run many

[4]Since this project was completed, a number of products to control windows by voice have appeared on the market.

applications as a matter of course, each utilizing one or more windows (see Figure 8.2). These might include a clock, calendar, mail reader, text editor, and perhaps a debugger with a graphical user interface.

The window system may allow windows to overlap or may tile them such that all are visible simultaneously. Overlapped windows are currently more popular; the added load on the user to keep track of partially "buried" windows allows more applications to make effective use of the screen simultaneously [Bly and Rosenberg 1986]. A **window manager** controls which window receives keyboard input or **focus.** A **real-estate-driven** window manager assigns input to the window on which the mouse cursor appears; **click to focus** requires the user to click a mouse button on a window to shift the focus there. The mouse is also used to hide and expose windows and to specify their size and location. A third use of the mouse is user input for applications supporting direct manipulation interfaces with buttons, scroll bars, sliders, and toggle switches.

Xspeak was a Media Lab project that explored an alternative use for voice in the multiple-application workstation environment [Schmandt, Ackerman, and Hindus 1990]. Instead of focusing on the applications themselves and rewriting them to support speech recognition, Xspeak instead used voice input to supplement the mouse for switching *among* applications under X windows. Xspeak provided an extension to window systems by matching limited recognition capabilities to a well-defined set of window operations. Xspeak is an example of an application for which small vocabulary speaker-dependent isolated word recognition is adequate. Although Xspeak was implemented using external recognition

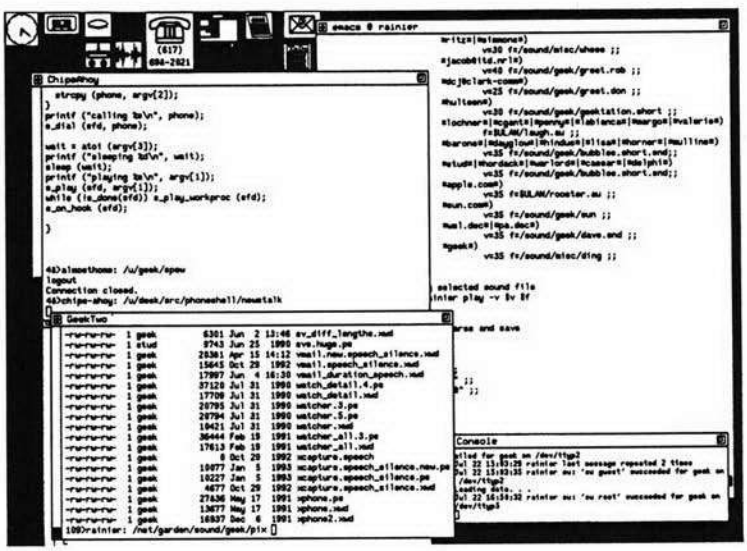

Figure 8.2. Many windows may be displayed simultaneously on a single screen.

hardware, suitable software-based recognition requiring no additional hardware is already becoming commercially available.[5]

Xspeak Functionality

Xspeak provided for the use of voice for the first two mouse functions previously mentioned: focus management and navigation among windows that may be obscured by other windows. Xspeak worked with real-estate-driven window managers supporting overlapped windows. Each window had a name; speaking its name exposed the window. Xspeak then moved the cursor into the window so the window manager assigned it the keyboard focus. Thus the user could move among applications sending keystrokes to each in turn without removing his or her hands from the keyboard. Additional commands allowed the user to raise or lower the window in which the cursor currently appeared.

Xspeak employed a small window to provide both textual feedback to the user and recognition control capabilities (see Figure 8.3). As each word was recognized, it was displayed in this window. In ordinary use this window was ignored as the window reconfiguration provided adequate feedback, but the display panel was useful when recognition results were particularly poor and the user needed to confirm operation of the recognizer. Through the control panel the user could name a new window; this involved clicking on the window and speaking a name that was then trained into a recognition template. The control panel also provided a software "switch" to temporarily disable recognition when the user wished to speak with a visitor or on the telephone. Another button brought up a configuration window, which allowed the user to retrain the recognizer's vocabulary, configure the audio gain of the speech recognizer, and transfer the recognition templates to disk.

Xspeak was an application not a window manager (see Figure 8.4). It worked by capturing recognition results and sending X protocol requests to the server. As with any client application, these requests could be redirected through the window manager, and the window manager received notification of the changes in window configuration or cursor location. Because Xspeak did not replace the window manager, it could be used across a wide range of workstation window system environments.

Why Speech?

Xspeak was motivated by the belief that window system navigation is a task suitable to voice and well matched to the capabilities of current speech input devices. Using speech recognition to replace the workstation keyboard would be very difficult; it would require large vocabulary speech recognition devices capable of dealing with the syntaxes of both English text as well as program text and oper-

[5]Xspeak has recently been revived using all-software recognition. This greatly facilitates deployment of the application.

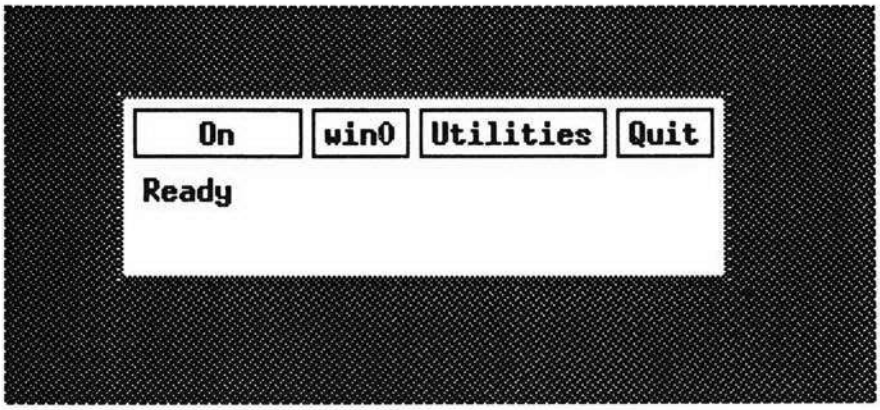

Figure 8.3. Xspeak control panel.

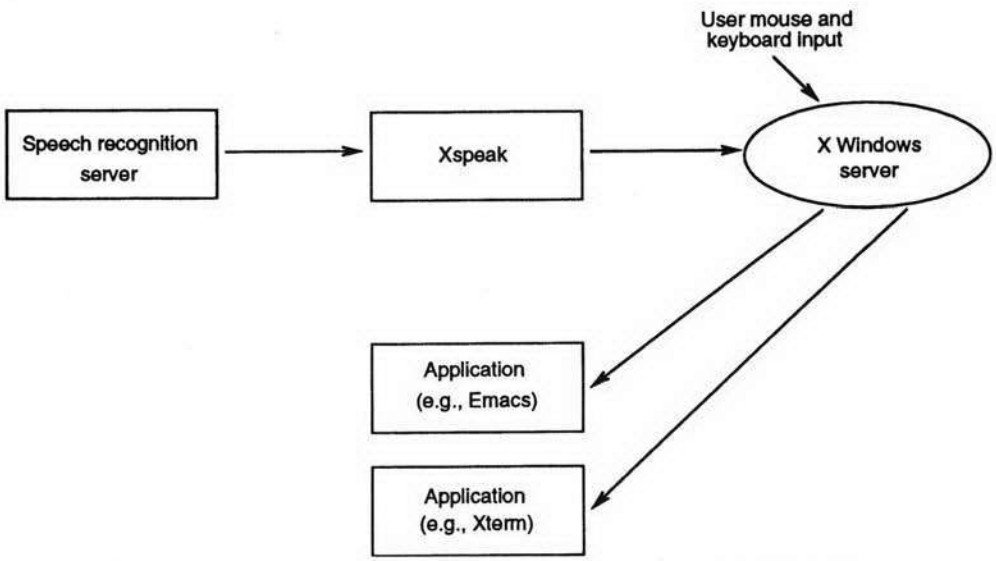

Figure 8.4. Xspeak cooperated with other processes under X windows.

ating system commands. In contrast, a typical user employs a relatively small number of windows, and because the user need speak only the name of the window, isolated word speech recognition suffices for the task. Using speech recognition to replace the mouse takes advantage of the fact that the user's hands and eyes are busy on the keyboard and screen. Using voice input saves the trouble of finding the mouse manually, performing an action, and then repositioning the hand on the keyboard.

Voice input also overcomes the dimension mismatch between windows and the mouse. The mouse is a two-dimensional input device, but windows are two-and-a-half dimensional; windows are planar but the plane of any window can be in front of or behind the planes of other windows. Once a window gets buried, the mouse must be used to move other windows out of the way until some part of the buried window is exposed, so mouse actions increase with the number of windows. But speech input bypasses the dimension problem and exposing windows is no more (or less) difficult than recalling an element from a list of names.

Finally, it was hypothesized that the divided attention theories previously mentioned would support improved performance by allocating multiple input channels to the task of working under a window system. To the extent that navigation and focus management are distinct tasks from the operations being performed inside the selected windows, dividing these tasks across modalities should enhance user performance.

Xspeak in Use

A small number of student programmers used Xspeak during their normal workday for several months and were observed by interviewing, videotaping, and logging of window system events [Schmandt *et al.* 1990]. These observations support the hypotheses about the utility of voice in this task only weakly, yet clarify some of the difficulties of speech recognition.

The dominant factor in user interactions and acceptance of Xspeak was errors. Even in this simple task, recognition rates were abysmal (often less than 80% correct), in large part due to the microphones selected. Because it seemed doubtful that programmers (or the real users toward whom an interface such as Xspeak would eventually be targeted) would wear head-mounted microphones, high quality directional microphones were positioned on stands next to the computer monitors. These microphones admitted much nonspeech noise, and the recognizer responded with a propensity for insertion errors due to sounds such as the keyboard, doors closing, and conversations in the hall. An insertion error would result in the user's keystrokes going into the wrong window; unless noticed immediately this could cause serious damage. To eliminate insertion errors, rejection thresholds were set rather high, resulting in the tendency towards rejection errors reflected in the recorded error rates.

Despite these errors, several users expressed a preference for the recognition-equipped workstations. Those realizing the most advantages from Xspeak were the very active users who kept themselves busy with tasks running in many windows. One such user had already developed techniques using an "icon manager"[6] to control windows and found no added value to speech input. One user employed very few windows and spent much of his time at the workstation during this study thinking and never made much use of recognition. All users abandoned the

[6]An icon manager displays a control panel listing the names of all windows. The user can make windows appear and disappear by clicking on this icon manager menu.

microphone-equipped consoles when significantly faster workstations became available elsewhere in the laboratory. Nonetheless, a majority of the users were able to use Xspeak effectively and incorporated it into their daily work routines. Some were extremely enthusiastic and competed for access to Xspeak.

Users did make frequent use of the control panel to test the recognizer and retrain words when recognition was especially poor. Users needed help selecting a vocabulary set appropriate for recognition; one user tried to train nonspeech sounds such as a sigh and a hiss as window names with no success. Summarizing the observed usage and opinions garnered from interviews, Xspeak was certainly intriguing and was perceived positively despite the high error rates, but it simply did not work reliably enough to change work habits. Most users thought that Xspeak was much faster than using a mouse (when recognition succeeded), despite measurements showing that for the most common very simple window actions (switch focus between two exposed windows) the mouse was actually somewhat faster.

The last observation might lend some credence to the divided attention theory, as performance would have to be measured by software production quantity or quality and the perceived speed of recognition might indicate that it interfered less with the programmers' cognitive processes. But the users' strongest complaint (after poor recognition accuracy) was that they wanted to use speech to interact with applications, not just select between them. The clean division of mouse functionality presented above did not dominate the users' perceptions, which suggests that using recognition for window management is just one aspect of multimodal input at the workstation. Although an interaction language to integrate speech recognition into application commands by simulating X input events and distributing them to application windows was designed, it was not implemented.

Put That There

An early example of an application employing several of the techniques described in this chapter was Put That There, done at an MIT Media Lab predecessor, the Architecture Machine Group, in 1980 [Schmandt and Hulteen 1982, Bolt 1980]. Put That There was an exploration into the use of multimodal input and conversational techniques to resolve recognition errors. While using this application, the user sat in front of a large-screen video projector and used voice and gesture to manipulate ships on a display of a map. Gesture was tracked via a magnetic six-degree-of-freedom sensor worn on the user's hand. The application responded with visual as well as aural feedback using synthesized or digitized speech.

Put That There was a command-and-control application that, depending on the selected database, let the user either create, move, modify, name, or destroy a fleet of ships in the Carribean or build towns, forts, and churches around colonial Boston. The work described here was based on an earlier prototype allowing for manipulation of abstract shapes on a blank screen. Although the prototype presented an impressive demonstration of the technology, the lack of voice response made for an interface only marginally usable: If a single word of an utterance was

not recognized, the mute application offered no feedback, and the user's only recourse was to repeat the utterance until something was recognized.

Speech was the primary input to Put That There, via a connected-speech recognizer with a vocabulary size of about a hundred words. The version described here accepted such utterances as these.

- Create a large red sailboat north of Haiti.
- Delete the cruise ship.
- Move the yellow oil tanker east of the Bahamas.
- Copy that (pointing to a ship) . . . north of that (pointing to another ship or a feature on the map).
- Where is Havana?
- Move the yellow oil tanker south of the large green sailboat.

The application's main input loop waited for recognition results while tracking the hand and recording gestures as they were detected. When speech was recognized, the first stage of the application's parser analyzed the input word by word. Each word was classified as an instance of a small class of semantic types from a grammar very specific to this application. Examples of these classes include the following.

- *Command:* the requested action.
- *Movable-object:* a type of ship, e.g. freighter, oil tanker.
- *Color.*
- *Size.*
- *Destination:* either a map location or another ship.
- *Destination-relative:* prepositional phrases such as "north of" or "below."

The parser generated a frame, i.e., a data structure describing a request in progress, which contained slots for the various semantic classes filled in with the specific instances of the words heard. The input loop then dispatched to command-specific procedures for semantic analysis. These procedures were hard-coded program segments that analyzed the frame to determine whether it adequately specified the requested action; for example, a "move" command requires an object to be acted upon as well as a destination. At this point, pronouns such as "that" or "there" were associated with screen coordinates by mapping gestures to objects identified by the bounding boxes of the various targets on the screen. Even if the pronoun were missing due to a rejection error, a recent gesture could satisfy the target specification requirement if it uniquely identified an object or position.

Database routines were invoked to associate descriptions with objects. If the user spoke "the red freighter" and this uniquely matched a ship, parsing could continue. If no ships matched the description, Put That There would ask "Which object?" In the case of multiple matches it would ask "Which one?" These terse responses were barely adequate for guiding the user to an unambiguous request; they would have benefitted from echoing or other explanatory behavior to help understand the nature of the misrecognition.

If an utterance was complete, then these command-specific analysis routines invoked action routines to update the ship database and modify the display accordingly. Most often, however, one or more words would have been misrecognized, resulting in an apparently incomplete request. Each semantic analysis routine included code to generate a query to the user; these queries were designed to elicit single-word responses because the recognizer exhibited much better performance on isolated word input. The queries such as "Where?", "Which one?", and "Relative to what?" were spoken to the user by a primitive speech synthesizer. Since there was a limited set of queries, the synthesizer was later replaced with higher quality digitized speech stored on magnetic disk.

Parsing and semantic analysis were **re-entrant;** after a query was spoken to the user, additional input was gathered from the main input loop and inserted into the utterance frame exactly as before. Since the frame was not initialized after the query, any new information was merged with the old and the semantic analysis routines could then examine the frame again. This query–response–parse cycle continued until an action could be executed or the user reinitialized the process by saying "restart." The net effect was that as recognition accuracy deteriorated, more time and user interaction were required to complete a request but breakdown was rarely catastrophic.

Several error correction strategies were available to the user if misrecognition resulted in a semantically meaningful sentence and a corresponding erroneous action. Put That There maintained a short history of location and attributes for each ship, allowing the user to invoke the *undo* command. If a ship was created with the wrong attributes such as being green instead of the requested yellow, the *change* command allowed for repairs without the need to start the transaction over, e.g., "Change that to yellow."

Another feature of Put That There was dynamic vocabulary management. The *name* command allowed the user to name a ship and thenceforth refer to that ship by name. Naming was accomplished by putting the recognizer into a selective training mode to create a new template when the user spoke the name and associating this template with the ship in the object database. Dynamic training was facilitated by the fact that the recognizer built vocabulary templates from a single training pass; otherwise, the user might have been required to speak the name several times.

The speaker-dependent aspect of the speech recognizer was used to advantage when a multi-user version of the application was desired. The outputs of two microphones were mixed, the vocabulary size was trimmed to occupy only half of the available template memory, and each user trained his own version of each word. As long as the two operators did not speak simultaneously, recognition proceeded as normal. Since the application knew which user had created an object, this information was added to the database to prevent unauthorized modification of one user's ships by the other user.

Put That There is noteworthy as one of the earliest user interfaces employing either multimodal input or conversational techniques using voice input and output. Although the application itself was only marginally relevant to real command and control systems, the user interaction techniques are applicable to a

variety of voice interfaces. Subsequent projects, most notably Conversational Desktop (described in Chapter 12), focused on the parsing and error management techniques initiated with Put That There.

SUMMARY

This chapter discussed incorporating speech recognition into interactive applications. There are many approaches to recognition, and different classes of speech recognizers are amenable to varying styles of applications. Recognition may be employed when no other input channel is available, as an adjunct to existing input channels (keyboard and mouse), or as a direct substitute for the keyboard for text entry. Each of these application areas has different recognition requirements and derives unique benefits from the use of voice input.

Poor recognition accuracy is the dominant concern in designing interaction techniques to employ speech input. Insertion, rejection, and substitution errors all degrade performance but in different ways. Errors can be minimized by careful choice of vocabulary and microphone placement and by chosing cooperative subjects, but these factors may not apply to real-world deployment of recognition technology. As recognition accuracy improves, errors inevitably occur and can be managed through user participation in a variety of confirmation and repair strategies; however, it must be stressed that some interaction techniques will always be required to ensure accurate communication. This chapter discussed the use of language and task constraints in the automatic detection of recognition errors; we will return to the topics of language understanding and dialogue structure in Chapter 9.

This chapter's discussion of error recovery implies that developing recognition-based applications is much more demanding than simply adding a new input device to existing applications. Xspeak was offered as an example of a meta-application, controlling the window system, and thereby selecting an application to receive keyboard events rather than modifying the applications themselves. Put That There was described as an example of simple parsing techniques and spoken dialogue with the user to resolve ambiguities. Although Xspeak illustrates an attempt to closely match the specific assets of speech recognition to the range of tasks performed at the workstation, the experience of its users also demonstrate the frustrations associated with using recognition in daily work.