The Design of Spoken Language Interfaces

Alexander I. Rudnicky
March 1990
CMU-CS-90-118

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This report describes how a speech application using a speaker-independent continuous speech system is designed and implemented. The topics covered include task analysis, language design and interface design. An example of such an application, a voice spreadsheet, is described. Evaluation techniques are discussed.

This research was supported by the Defense Advanced Research Projects Agency (DOD) and monitored by the Space and Naval Warfare Systems Command under Contract N00039-85-C-0163, ARPA Order No. 5167.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of DARPA or the U.S. government.

Keywords: Human–Computer Interaction, Artificial Intelligence Applications, Spoken Language Understanding

Contents

1 Introduction

The ability to use speech enhances the quality of human communication, as reflected in shorter problem-solving times and in general user satisfaction [1]. Unfortunately, these benefits have not yet been realized for human-computer communication, due to the inherent limitations of existing speech recognition technology. Recent advances, e.g. [6], have made it possible to build "spoken language" systems whose capabilities allow for natural interaction with computers. Spoken language systems (SLSs) combine a number of desirable properties. Recognition of *continuous speech* allows users to use a natural speech style. *Speaker independence* allows casual users to easily use the system and eliminates training as well as its associated problems (such as drift). *Large vocabularies* make it possible to create habitable languages for complex applications. Finally, a *natural language* processing capability allows the user to express him or herself using familiar locutions.

The availability of such features as speaker-independence and continuous speech removes a number of restrictions that formerly limited the use of speech interfaces (but for the same reasons simplified the development of such interfaces). Speaker-independence makes a system accessible to casual users who do not need to commit to an enrollment procedure, as do users of speaker-dependent systems. Similarly, the ability to process continuous speech removes a major restriction on the user and allows relatively natural communication of complex information.

Removing these restrictions complicates the interface design process. To preserve the naturalness of communication, the designer must make an effort to define an expressive system language that allows the user the freedom to use fairly natural modes of expression. The system design must also minimize a system's attentional requirements, allowing the user to concentrate on the task at hand rather than on the operation of the speech system. This latter goal requires the system to handle a variety of phenomena related to spontaneous speech, such as pauses in the middle of an utterance and the intrusion of both environmental and talker-generated non-speech sounds.

Our interest is in understanding the design of spoken language systems for complex problem-solving environments in which the user can take advantage of speech to control a variety of applications. To develop an understanding of such environments, we implemented a voice-operated spreadsheet program and used it to study user interaction. The purpose of this report is to describe the techniques we used for developing and evaluating this system. We present a rather detailed description of each step in the process, as we believe that any implementation of a spoken language system meant for working environments requires an approach similar to the one we describe.

Since we were interested in examining a voice interface to basic spreadsheet functions, we chose to work with *sc*, a UNIX-based spreadsheet originally implemented by James Gosling, for which source code was available. *Sc* is similar in functionality

to VISICALC, an early spreadsheet program. To provide speaker-independent and continuous speech recognition, we made use of the SPHINX recognition system [6]. The resulting system is VSC, or "voice sc".

2 Task Analysis

Before designing and building a speech interface, it should be determined how, or even if, speech can be used for a particular application. Although speech is clearly a desirable input medium, it does not follow that it should be used in all cases. Moreover, potential interactions, both positive and negative, between different input modalities must be considered during the design process.

A drawing program offers a good example of an application where the decisions on how to use speech are clear-cut. Some components of such a program are clearly unsuited for voice control. Due to the nature of the process, producing graphics is most efficiently done through direct manipulation, as afforded by a mouse or similar device. On the other hand, generating discrete inputs such as mode selection seems to be better suited for voice. While a primary argument for introducing voice remains the naturalness of speech input, a number of specific benefits can also be realized through the introduction of a voice channel. The drawing application illustrates two of these. Apart from the use of command keys (which have their own problems), conventional mode selection strategies require that the user either release the mouse in order to key in mode changes, or move the pointer to a menu panel. The availability of a parallel voice channel would eliminate the resulting disruption [9]. A related advantage is the ability to generate inputs without having to look away from the work area, often a costly operation in terms of time and effort [5].

In the case of the spreadsheet, we felt that all its functionality could be accessed by speech, including both command and function invocation as well as numeric data entry. The one exception was the input of arbitrary text information. This latter requirement highlights one of the shortcomings of current recognition technology, the difficulty of incorporating novel items into the recognition lexicon. The problem actually consists of a number of separate problems, some quite difficult:

- Determining that the user has indeed produced a new word, which requires proper delimitation of that word (or words) in a continuously spoken utterance, as well as some method for distinguishing new words from poorly recognized instances of known words.
- Creating a representation suitable for subsequent recognition: In a system like ours this requires the derivation of a phonetic transcription for the new word. If a whole-word template approach is chosen, this might not be necessary.

- Establishing an orthographic version of the word suitable for display. A user could simply type in the string corresponding to each new word. It would, however, be preferable if the system could infer the proper written form, based on the rules of the language, or at least was able to present the user with a reasonable approximation that the user could modify to suit.
- Establishing the syntactic and semantic role of the new word, so that subsequent uses of the word can be handled properly. Creating a cell label in a spreadsheet is a good example. The system needs not only to recognize the new word, but must understand that its occurrence constitutes a reference to a cell location or perhaps its contents.

The system we have built does not allow for active entry of new words into the lexicon. In fact, all arbitrary string inputs are handled by keyboard input.

In our experiments with the system, we simulated the ability to define arbitrary words by including a vocabulary keyed to the tasks we asked our subjects to perform (personal finances). For situations where the area of application is well-defined, a suitable vocabulary can be incorporated into the system beforehand and would give the user the flexibility to use intuitively preferable modes of expression, i.e., keyed to the conceptual categories being worked with (such as meaningful cell labels) rather than the abstractions provided by the base system (row and column coordinates).

3 Language Design

Once the functionality suitable for speech has been identified, an exact specification for a language that invokes this functionality must be developed. A *spoken* language is unique and differs significantly from other classes of language. That this is so should be apparent if one considers the difference between expressing something verbally and writing it out. Written language affords the opportunity to choose expressions carefully and to revise the presentation until it's satisfactory. In the verbal case, particularly if production is spontaneous, the same presentation will be very different. It will make use of a different argument structure, grammatical constructs and word choice. The design of a recognition language must therefore capture the properties of spoken language.

The manner in which a user might be disposed to address a computer application may differ significantly from the manner in which another human might be addressed [4, 7], a process known in linguistics as a change in *register*. Thus, it may not be sufficient to know how people express certain requests to each other, it may actually be necessary to understand how they would do so when knowingly addressing a computer application.

A spoken language interface also needs to do more than provide a direct trans-

lation of existing commands designed for another modality. Thus, one goal in the design of keyboard command languages is to minimize the number of keystrokes required to produce a result. Such a goal may not be relevant for a speech command language, where the effort of producing an individual entry is less. In fact, a reasonable goal for spoken commands might be to provide more verbose (and natural) command expressions.

The above observations are, however, complicated by a number of factors. For example, the verbosity of inputs may be linked to an individual's level of expertise. While the novice user may find it useful to be able to express requests in a voluble mode ("Please put twenty five dollars into cell A nine"), an expert might prefer to use a terser telegraphic style ("A nine. twenty five"). Depending on the goals of the design, the language would need to accommodate one or both of these styles. Another source of difficulty is the tendency of users, when speaking, to express themselves in terms of goals rather than component actions. It is more natural to say "Add the entries in column C and place the result in A one" than to say "go to A one. enter the sum of C four through C twelve plus the sum of C fourteen through C eighteen". A natural language component that can handle such abstract forms, as well as a reasoning component, needs to be incorporated into an application if such requests are to be allowed. The use of such expressions does not necessarily reflect a simple tendency to speak "naturally". It might be better thought of as indicating the user's desire to avoid the effort needed to specify a detailed action sequence. Another practical consideration for the user is not having to remember the exact component actions needed to reach some goal.

The properties of spoken language can also motivate the reorganization of an existing interface. For example, in menu-driven interfaces, limited real-estate (and other considerations) force the design of complex menu trees, and require a succession of interactions to produce a result. Spoken commands provide the user with a large virtual menu of choices, so that a succession of menu accesses can be replaced by a single voice command. If the system has the capability of interpreting requests in terms of goal states, then such a "voice menu" can be quite large, since the user does not need to remember the details of invocation for each command.

The proper choice of language for an interface is not obvious from a simple survey of an application's functionality. We have found that the most useful approach has been to record the language of users while they perform tasks using the system and to use this as a basis for developing a spoken language suitable for the application. It is critical to observe a sufficient variety of users, since individuals may display idiosyncratic behavior that is not representative of a user population as a whole.

3.1 The "Wizard" Paradigm

The design of a spoken language system presents somewhat of a chicken-and-egg problem. It is not possible to understand how best to design such a language without

knowing how people might wish to interact with a particular application. On the other hand, it is not possible to observe interaction with such a system without having the system already in place.

The solution to this problem is to build a simulation of the target system and to observe users interacting with it. The speech data collected from such a study can then be analyzed and a speech language developed. The function of the speech recognizer and parser is carried out by a human operator, hidden from the user, who is unaware of the operator's intervention. This form of simulation has recently come to be called the "Wizard" paradigm.[1]

In our own work, we have used different forms of the Wizard paradigm, from fairly elaborate simulations that present the user with a complete illusion of a functioning system (see e.g., [4]) to somewhat informal situations designed to elicit speech more characteristic of human-human speech (see [12]).

To develop a speech language for the spreadsheet program, we conducted a series of simulations in which we asked participants to perform several well-specified tasks. To see whether the complexity of the task had any influence on language, we used two tasks, a financial data entry task and a planning task. In the former, people were asked to enter information characteristic of an individual's monthly finances into a pre-programmed spreadsheet. In the planning task, people were given a filled financial spreadsheet and were asked to perform modifications that would satisfy certain furnished constraints (e.g., decreasing expenses by a certain total amount). Interestingly, subjects in both experiments produced very similar protocols, invoking the same facilities in both cases. The only difference noted was a small increase in "speculative" constructions for the planning task (*"What if rent were changed to $500?"*) that express requests in indirect form. Significantly, subjects on the whole used language appropriate to the known functionality of the system they were interacting with and did not, by virtue of using speech, expect expert-system-like properties of the spreadsheet.

3.2 Protocol Transcription

All speech produced during these sessions was recorded on tape and later transcribed. Table 1 shows an excerpt from a transcription. The system of annotation chosen provides fairly detailed information about the contents of subjects' utterances. In addition to information about the lexical and syntactic choices made, the transcript allows for analyses of pause location, emphasis, and interjection. Information about these phenomena is desirable since all have consequences for the

[1] The term comes from a character in L. Frank Baum's book for children, "The Wizard of Oz" book. The wizard being someone who manipulated a large statue from behind a curtain, leaving observers with the impression that they were speaking to the statue itself. The "Wizard of Oz" story was also the source for the no longer favored term "pnambic experiment" (for "Pay No Attention to the Man BehInd the Curtain"), another allusion to the Wizard of Oz.)

Table 1: An excerpt from a protocol transcription

```
under cable, enter fifteen dollars even [*]
oka:y, go to the next screen [*]
oka:y unde:r . movies .. in the entertainment, section, subsection .
enter twenty-eight fifty [*]
oka:y .. u:h in the . FOOD subsection under HOSting [*]
enter .on the line above that, enter a hundred seventy-five dollars [*]
okay, go down to the oh wait
on the CLOthing . entry [*]
ente:r . sum of . eighty-five, eighty-nine, twenty-four [*]
[click] okay
and, under gasoline, enter twenty-two fifty [*]
okay . uh . next screen [*]
```

Notes: The transcription, in addition to recording the speech produced, attempts to indicate additional phenomena of interest, primarily prosody and extraneous events. The [*] mark indicates the point in time at which the "system" is responding. A comma (,) indicates a boundary mark, either a short pause or an inflection. Capitalization (e.g., CLOthing) indicates emphatic stress. Items in square brackets (e.g., [rustle]) describe extraneous audible events.

operation of a speech recognition system.

Transcription, in the course of spoken language system development, actually has two uses. The first is for the initial language analysis being described in this section. The second is for evaluation purposes, where the goal is to match the output of a recognition system to what was actually spoken. The requirements for this second transcription style are more rigid than those for the first and require that certain additional distinctions be made, such as between words that are in the recognition lexicon and those that are not. The following guidelines were used for the evaluation portion of our work:

1. An *event* is defined as audible acoustic energy delimited by silences or by other labeled events. When two events overlap, preference is given to the lexically meaningful element (e.g., word over noise), or to the element attributable to the nominal session talker. Otherwise, the most salient event (as judged by the transcriber) is given preference. No attempt is made to further code overlapping events.
2. *Transcribe all words.* If a particular word is not recognizable, a guess is made, based on the transcriber's best understanding of the context of occurrence, both sentential and task, in which the word occurs. If a word or phrase cannot be identified with reasonable confidence, then the "++MUMBLE+" marker may be used. If a word is mispronounced but is nevertheless recognized correctly, it is transcribed as if it were spoken correctly. If it is misrecognized, it is transcribed as heard.
3. *Label all other audible events.* At the least level of detail, these can be identified by a cover symbol ("++NOISE+"). We have found, however, that it is useful to label separately those events that occur frequently enough to be of interest in themselves, such as breath noises, or perhaps telephone rings.
4. *If the system recognizes an interrupted word correctly, then the word is transcribed as if it were spoken in its entirety.* This convention is arbitrary and obviously hides

information about interrupted words that are nevertheless correctly recognized. It may need to be revised as we begin to explicitly study such phenomena.

5. *Utterances that produce no recognizer output are eliminated from the transcript.* In our system, this consisted of zero-length "utterances" that result from malfunctions in endpoint detection. A record of these utterances should of course be kept, so that relevant statistics can be calculated.
6. By convention, *any utterances at the beginning of the session that reflect the user's unawareness that the system just went live are eliminated.* This speech consists of interactions with the experimenter and typically reflect the user's unawareness that the period of instruction has ended. Since the user is not "using" the system, this convention is justified. A record of such deletions is kept, however.
7. *Extraneous noises are always transcribed if they affect the recognition.* Otherwise, noises are transcribed only if, in the opinion of the transcriber, they are sufficiently prominent ("loud enough"). Certain noises, in particular inhalations and exhalations at the start and end of an utterance, are not typically transcribed.

The above guidelines could be equally applicable to transcriptions produced at the exploratory stage. These guidelines and related issues are discussed in greater detail in [13].

3.3 Language Analysis and Design

On first examination, we found the transcripts to contain a wide variety of locutions, so wide in fact as to lead one to believe that very little consistency was present. Detailed analysis of the corpus, however, revealed that the language used was quite consistent and that utterances could be divided into a compact set of categories. These are shown in Table 2. Examples of utterances for one of the categories, DATA ENTRY, are shown in Table 3. A more detailed discussion of these categories can be found in [12].

Using the utterance categories as a guide, we can define a manageable language for the spreadsheet task. The proper design of a spoken language should not mean that every locution encountered in a simulation should be made part of the language. Rather, the goal should be to ensure that all major categories are represented. As long as the user has available the proper *modes* of expression, and as long these modes are derived from the actual experiences of users, users should be able to remain within the syntactic and lexical bounds imposed by the designer.

Once this analysis is performed, it is possible to define a system *lexicon* and *grammar*. The lexicon specifies all words that the recognition system must be able to recognize. The choice of a lexicon is fairly critical, since it defines the composition of the training materials for the recognizer. Since training represents a major time and resource investment, it is important that it be done correctly. Grammar is somewhat more flexible, since it represents arrangement of lexical units and can therefore be modified at any point in the development process.

Table 2: Utterance categories in the spreadsheet task.

DATA ENTRY

- **Numeric**
- **Absolute Locate and Enter**
- **Relative Locate and Enter**
- **Symbolic Locate and Enter**
- **Symbolic Locate and Enter (no keyword)**
- **Absolute Locate and Enter (reverse order)**
- **Relative Locate and Enter (reverse order)**
- **Symbolic Locate and Enter (reverse order)**

MOVEMENT

- **Absolute screen movement**
- **Relative screen movement**
- **Explicit screen movement**
- **Screen positioning**
- **Absolute movement**
- **Symbolic Movement**
- **Compound position**

CORRECTION

- **Simple deletion**
- **Keyword at head**
- **Mid-utterance correction**
- **Keyword at end**
- **Replacement**
- **Implicit correction**
- **Prosody**

MISCELLANEOUS COMMANDS

- **Screen refresh**
- **Addition of row/column**
- **Labeling new row**
- **String formatting**

A task language has three components: *control*, *task-specific*, and *generic* subgrammars. This is useful distinction to maintain, particularly if multiple tasks are being implemented. The control component provides access to general system functions, such as undoing a preceding recognition, putting the recognizer into standby mode, or quitting the system. The generic component represents language fragments that could be used without modification in a variety of applications. The language of numbers is a good example of this. The task-specific language includes the non-transportable component of a language, incorporating task-specific lexical

Table 3: Examples of entries in the DATA ENTRY category.

DATA ENTRY

Numeric
sixty-five hundred dollars

Absolute Locate and Enter
uh go down to line six and enter four thousand five hundred

Relative Locate and Enter
on the next line enter seventy-five ninety-four

Symbolic Locate and Enter
go to the cell for my salary and enter the amount six thousand five hundred
under credit card enter the amount zero

Symbolic Locate and Enter (no keyword)
food two seventeen and eighty-five cents plus two hundred forty-six dollars

Absolute Locate and Enter (reverse order)
enter six thousand five hundred in column b row 6

Relative Locate and Enter (reverse order)
enter seven hundred and forty-eight fifty-seven there

Symbolic Locate and Enter (reverse order)
fifteen dollars under bank charges

Table 4: Excerpt from the spreadsheet BNF grammar.

```
<number_type>  ::= <numberstr_act> |
                   <numberstr_act> <decimal_act> <digitstr_act> |
                   <begin_digitstr_act> <decimal_act> <digitstr_act> |
                   <decimal_act> <digitstr_act> |
                   <begin_digitstr_act> |
                   <o> <decimal_act> <digitstr_act>
```

items and grammatical forms.

We have developed utilities that allow us to conveniently specify a language. For example, grammar is expressed in the form of a BNF, while lexical entries are specified in terms of phonetic pronunciations. Table 4 shows a fragment of the BNF grammar for the spreadsheet. Table 5 shows some lexical items.

To produce a knowledge base for the recognizer requires creating suitable pho-

netic models. Since we are working with a speaker-independent system, this implies the collection of a sufficient amount of recorded speech. Commonly this means ensuring that at least 10–20 exemplars of each lexical item occur in the training set, in different contexts, spoken by different talkers. We accomplish this by using the BNF grammar to generate a corpus of training utterances that can then be used as scripts for recording. Since we have control over the generation process, we can assure (by differentially weighing production rules) that sufficient instances of each word occur in the corpus. To achieve speaker-independence, we have many different individuals read sets of utterances (say, as many as might fit into a 15 minute session). The recognition system is then trained using this material. For spreadsheet training, we combined several databases, including ones containing calculator sentences, spreadsheet sentences, and ones specific to financial management (the domain used for the evaluation tasks). A total of 4012 utterances was available for training.

3.4 Parser Design

The parser in our original system uses a rather simple context-free grammar, implemented primarily by means of the UNIX *lex* and *yacc* packages. The parsing strategy incorporated into VSC uses a two-stage parsing process to interpret a single string produced by the recognition system and to generate suitable spreadsheet commands. Figure 1 shows this process.

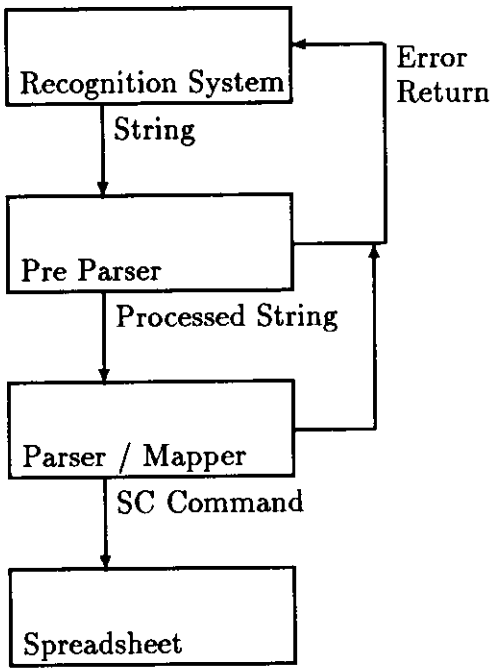
The purpose of the first stage is to produce an initial analysis of the input string, to determine whether it is syntactically correct and to perform certain forms of preprocessing, such as the conversion of spoken numbers into their digit equivalent (e.g., `TWELVE HUNDRED AND THIRTY FOUR` into `1234`). The purpose of the second stage is to complete the parse and to perform the mapping into actual spreadsheet system commands.

Our more recent parser designs have been based on case-frame representations, coupled with a finite-state phrase-based recognition grammar. Case-frame parsers offer comparable execution speeds, but with more powerful language processing, better suited for spoken language.

4 The Recognition System

The voice spreadsheet incorporates Hidden Markov Model (HMM) recognition technology, described more fully in [6]. The units of recognition are words, built from triphone models, which are phones taken together with their immediate context (preceding and following phones). The use of triphone models allows the representation to incorporate information about coarticulatory effects, thereby increasing recognition accuracy. The language designer specifies each word in the lexicon in terms of a pronunciation, as shown in Figure 5. Note that the pronunciation used

Figure 1: The parsing process in the voice spreadsheet system.



is the *most common* form of the word rather than the (citation) form found in a dictionary. Determining the most common form of a word demands some investigation, often made easier by the availability of the wizard experiment recordings. However, the effort is well worth it, since conversational forms of a word are often different from the citation form. Experience with several recognition systems that make use of phonetic representations [6, 2] has shown that using the most common pronunciation provides a significant increase in recognition accuracy.

Phonetic pronunciations can be automatically transformed into triphone forms (as in Figure 6), the units actually used for training the recognition system. For a task such as VSC, the initial lexicon is defined in terms of 48 phonetic labels. The actual number of triphone models trained is 971. Techniques are also available for clustering similar models, to reduce the total number of of models needed. Since this clustering does not increase accuracy [6], we did not perform it for the current system.

The recognizer makes use of grammatical constraints to reduce search and to increase recognition accuracy in the process. These constraints are expressed in the form of a *word-pair* grammar, derived from the BNF grammar. A word-pair grammar provides minimal constraint, since it only specifies which words may follow the current word, and does not incorporate more global constraints, such as would

Table 5: Some phonetic specifications for lexical items.

BEGIN	B IX G IX N
BONDS	B AA N D Z
BOTTOM	B AA DX AX M
BREAK	B R EY KD
BY	B AY
C	S IY
CABLE	K EY B AX L

Table 6: Triphone expansion of monophone lexical items.

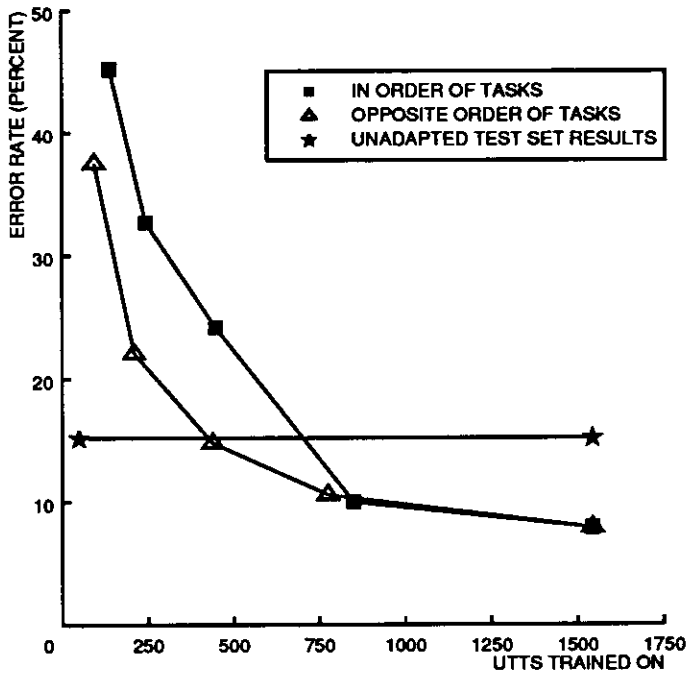
BEGIN	B(SIL,IX) IX(B,G) G(IX,IH) IH(G,N) N(IH,SIL)
BONDS	B(SIL,AA) AA(B,N) N(AA,D) D(N,Z) Z(D,SIL)
BOTTOM	B(SIL,AA) AA(B,DX) DX(AA,AX) AX(DX,M) M(AX,SIL)
BREAK	B(SIL,R) R(B,EY) EY(R,KD) KD(EY,SIL)
BY	B(SIL,AY) AY(B,SIL)
C	S(SIL,IY) IY(S,SIL)
CABLE	K(SIL,EY) EY(K,B) B(EY,AX) AX(B,L) L(AX,SIL)

be provided by a finite-state grammar. For the VSC task, the typical perplexity of the tasks carried out by the user is approximately 52. A finite-state grammar would provide much lower perplexity and would make the recognition problem much more tractable. Other things being equal, it is desirable to construct languages that have low perplexity, since this insures higher recognition accuracy.[2]

An alternate method for increasing recognition accuracy would be to configure a speaker-dependent system. Our experience indicates that for a SPHINX-like system, speaker-dependent training can reduce the utterance error rate by half for the spreadsheet task. This level of performance, however, requires about 1500 training utterances per talker to achieve. Figure 2 shows the effect on utterance accuracy of training a recognizer using different amounts of training data. The data are for a single user (jyk) from the voice spreadsheet evaluation study described below. Sets of 1, 2, 4, 8, and 15 scripts (each containing about 100 utterances) were used for training. A separate set of 5 scripts collected a month after the main experiment was used as the test set. Between 500 and 700 utterances, depending on how recently the speech was collected, are needed to match speaker-independent performance.

[2]Perplexity is defined as the geometric mean of the branching factors in an utterance. Perplexity is used as an estimator of the difficulty of recognition for a particular language. The perplexity of arbitrary English text is estimated to be about 200; the perplexity of a tightly-constrained task language can be as low as 5, a value characteristic of many commercial recognition systems. While perplexity gives an indication of how difficult the recognition process might be for a particular language, it is only one determinant of difficulty. The confusability of items in the vocabulary, as well as its size will also affect recognition difficulty.

Figure 2: Speaker-dependent training for the spreadsheet task.



5 Interface Design

The proper design of a speech interface requires attention to a number of factors, including attention management, error correction, and interaction control.

Spoken communication does not have a direct physical focus in the way that keyboard entry does. In keyboard interaction, there is little ambiguity about the user's intent to communicate—the keyboard is mechanically activated by direct contact. There is no such explicit link in the case of speech. When a user speaks, the utterance may indeed be directed at the currently active application, but it could just as easily have been directed to another human in the environment or have been the result of thinking aloud. The problem is that the system does not have the concept of *attention*; it does not know when it should be paying attention to a potential input. There are a number of solutions to this problem.

The simplest is a "push to talk" system, where the user needs to perform some explicit mechanical act, such as depressing a button during the duration of the utterance or depressing a key just prior to the utterance. These solutions, while

dealing the problem at hand, are not satisfying, since they require the user to perform an incompatible or even disruptive act in order to signal an input to the system. We have experimented with the use of "attention" words that allow the user to activate the recognizer via a single keyword, such as `LISTEN`. If the recognizer is in the standby mode, it will not actively process speech until it encounters the attention word, subsequent to which it will transmit recognitions to the application. If we also provide a keyword for switching off attention, e.g. `STANDBY`, then the user has been given explicit control over the system's "attention" in a manner that is consonant with the modality. (We also have found it useful to have the system enter the standby mode if the quality of recognition falls below some level, on the assumption that the user has redirected his or her attention to another listener in the environment).

A second aspect of attention control has to do with the tendency of intended utterances to map imperfectly onto acoustic utterances. An *acoustic utterance* is defined in terms of physical attributes and consists of a continuous region of acoustic energy delimited by some silence. Actual complete utterances (a *logical utterance*), as a human might understand them, may consist of one or more acoustic utterances, particularly if the user pauses in the middle of an utterance while mentally formulating what he or she will say next. A related phenomenon occurs when the user runs together several logical utterances into a single acoustic utterance. If this is allowed, then the system gives up a valuable input constraint, somewhat like the one exploited by discrete-word recognition systems. A proper solution to the potential lack of a one-to-one correspondence between acoustic and logical utterances is beyond the scope of a simple interface design and would require the use of a sophisticated parsing algorithm.

A third component of speech interface design is some provision for error recovery. Speech recognition systems will, for the foreseeable future, produce errors. Even if perfect recognition is achieved, the user will occasionally utter unintended commands and will need to restore the system to a previous state. We have investigated several approaches to the problem of providing error recovery facilities.

In an early experiment, we compared time-to-completion for a voice spreadsheet task using two different forms of confirmation. In one mode, recognized inputs were acted upon immediately and the user had to recover from errors by re-entry of commands. In a second mode, users had the opportunity to edit an ASCII version of the utterance, presented in an editor buffer, before it was acted upon.

Table 7 shows the times to completion for these two confirmation modes as well as a reference time-to-completion for keyboard entry, taken from [11]. There is a clear advantage for the explicit confirmation mode. The explicit confirmation mode, however, is more awkward to use, since the flow of the interaction is interrupted by the requirement for a (keyboard) confirmation. Because of this, and because of the desire to implement a "pure" speech system, we chose the implicit confirmation model for the system we implemented and therefore did not provide an explicit

Table 7: Comparison of confirmation styles.

Input Modality	Task Completion Time (min)
keyboard only	12:39
voice and keyboard	15:16
voice only	19:21

mechanism for error recovery. It is our belief that the implementation of error recovery should if possible keep the error correction protocol entirely within the speech modality. Failing that, it should be a multi-modal system that can function with minimal interruption in the flow of interaction, since such interruptions involve shifts of attention and increase the effort required to interact by speech.

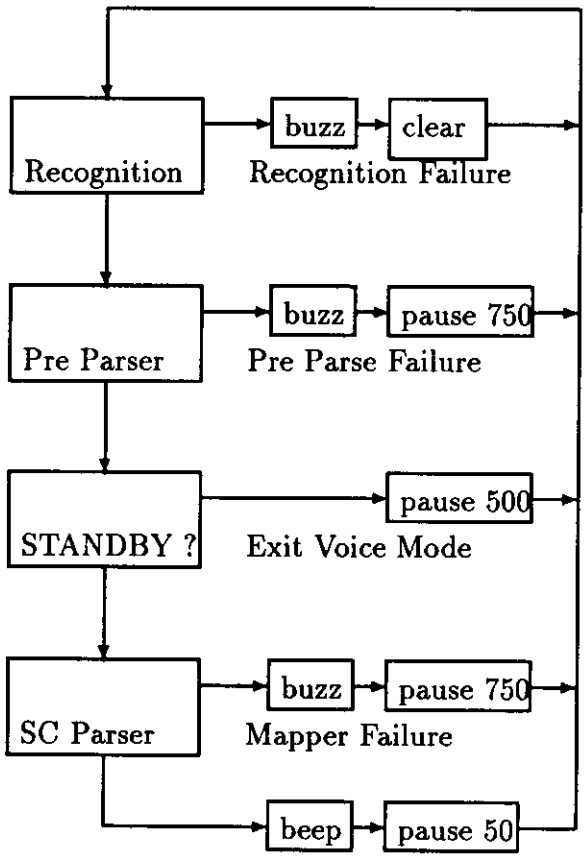
5.1 The Structure of the Interaction Cycle

Figure 3 shows a detailed flow diagram of the VSC interaction loop, showing recognition, parsing, and execution components.

The system can be in one of two major input modes: Listening and Execution. The system does all signal acquisition and recognition in the Listening mode and locks out speech input during the Execution phase. The structure of the Listening mode is shown in Figure 4. Immediately upon entry into the input loop, the system displays the "`Wait...`" token in the information buffer (line 2 of the spreadsheet display) and initializes the signal processing module. This step takes an average of about 200 ms, though conditions will occasionally force a longer delay. Once the system is actually listening, the "`LISTEN`" token appears in the display buffer, indicating to the user that speech input is possible. The system waits until the onset of speech energy is detected and immediately begins the recognition search. The transition to this state is indicated by the appearance of the "`Recognizing...`" token. Termination of the recognition process is signalled by the appearance of the recognized string on the screen. Since recognition is not real-time, this change in state may occur some noticeable interval after the user has finished talking. Note that this protocol provides the user at all times with a clear indication of recognizer state. This is particularly desirable, if the temporal uncertainties associated with speech processing exceed certain limits. In a pilot study [14], we found that users seem to cope well with delays of up to 150–300 msec. With longer delays (exceeding 600–800 msec) users found it easier to pace their interactions by means of these external indicators.

An utterance that successfully passes through the recognition and parsing stages will result in a spreadsheet action, indicated to the user by a harmonic beep (F_0 250 Hz) of 100 ms duration. The various failure modes will generate a 100 ms buzz (F_0 100 Hz). These two signals are distinctive and give the user initial information about the success or failure of the current input. A short pause, of 750 msec, follows

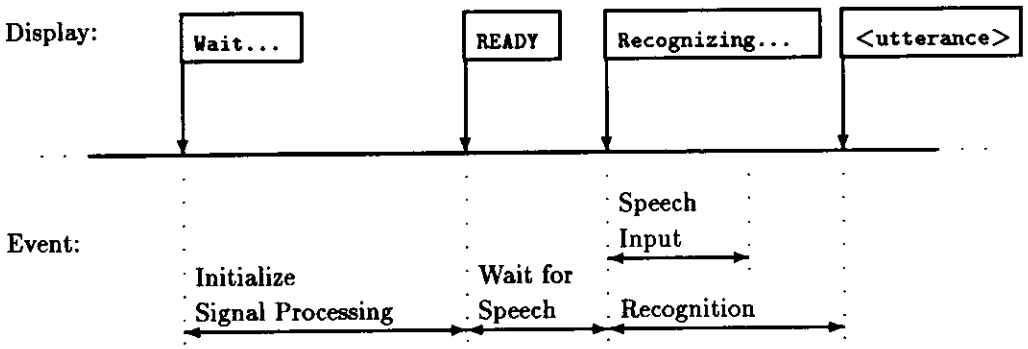
Figure 3: Interaction flow in VSC



parser failures, so that the user has the opportunity to glance at the display buffer and note the nature of the failure.

We found that the inclusion of such detailed information about system state was useful to the user, given the operating characteristics of our system. A system functioning without delays and capable of high-accuracy recognition might not need to rely on as many state indicators, since successful completion of an individual command is highly probable. As a result, the cost of error interception and recovery can be amortized over a large number of interactions. Once systems with such characteristics are available, a systematic investigation of this issue can be undertaken.

Figure 4: Detail of the Listening Mode.



6 System Evaluation

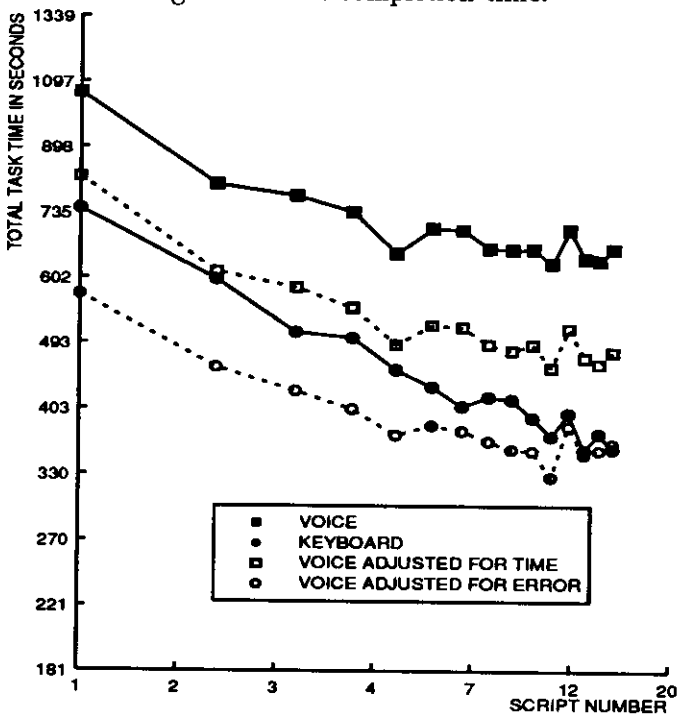
To be informative, the evaluation of a spoken language system needs to take into account a number of dimensions, some relevant to comparisons with other input modalities, others useful for understanding the performance of the speech system itself. The potential advantages of speech input resolve to ways that it can improve the performance of some complex task. Is the user capable of performing the task more rapidly when input is by voice? Are there fewer errors in the completed work? Does using a speech interface entail less effort for the user?

Understanding the performance of a spoken language system requires the examination of a number of traditional measures, as well as some new ones, inasfar as they impact system performance. While word accuracy has been considered as a base measure of system performance, in a task situation utterance accuracy becomes more important,[3] since it has a direct impact on task throughput—each erroneous utterance requires either re-entry of the information or correction followed by re-entry. The usability of a system is also affected by the degree to which the system language is appropriate for the task and the degree to which it manages to encompass the language that the user chooses to bring to the task.

The following discussion draws on a user study of the spreadsheet system described in this report. In this study [10], eight users completed a series of thirty spreadsheet tasks, alternating speech and keyboard input. Each task required the entry of about 40 items of personal financial information into a predefined work-

[3] Word accuracy (w) is related to utterance accuracy (u) by $u = w^n$ where n is the average utterance length, in words. Systems for which interactions are limited to single words or short utterances are reasonably characterized by word accuracy. Systems that are characterized by long input sentences will exhibit much poorer utterance accuracy. The presence of extraneous sounds, as observed for the spreadsheet system, can significantly degrade system response, in comparison to evaluations performed on "clean" speech.

Figure 5: Task completion time.



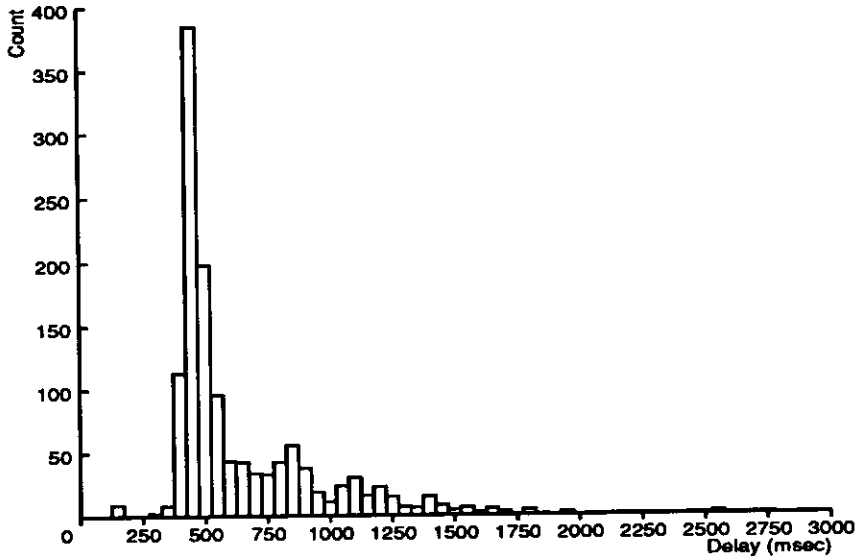
sheet. Items could be numbers or expressions. The VSC program was instrumented to allow collection of timing and recognition data.

6.1 Task Completion Time

The effectiveness of a spoken language system as an interface can be evaluated by comparing the time it takes a user to carry out a set of standard tasks by voice with the time it takes to complete the same task using some other input modality, keyboard in our case. Figure 5, taken from [10], shows total task times for spreadsheet tasks performed by voice and by keyboard. Overall, these tasks take longer to carry out by voice. These data, however, confound a number of variables, in particular the performance of the speech system itself, with the effectiveness of spoken input.

Two aspects of recognition system performance were poorer than desired: system recognition time, and recognition error. Due to the nature of our hardware implementation, average system response was on the order of 1.9 times real-time, the excess over real-time being attributable to speed limitations in the search component of the system (additional delays were introduced by the requirement that the end of utterance be delimited by approximately a 200 ms silence, and by a potential pipeline delay in the signal processing component of over 400 ms; inter-

Figure 6: Distribution of system response delays.



Notes: A total of 1330 delay intervals are shown here, pooling a dozen sessions from several talkers. The bin size is 50 msec. Less than 1% of the delays exceed 3 sec. These are not shown in this histogram.

mittent delays were also introduced by network communication software). While engineering solutions exist for many of these delays, they could not be avoided in our experimental system. Figure 6 shows the distribution of delays for a typical session. Most delays fall into the range of 400–600 ms, though some individual delays are quite long. Some understanding of such delay distributions can be of use in evaluating the usefulness of a speech system, particularly for applications that require a fairly tight interaction loop, as is the case for a spreadsheet. Users of systems that are characterized by a less coupled interaction style (database retrieval may be an instance) may tolerate significant delays in response [3].

The system under evaluation also exhibited a less than desirable error-rate, on the order of 25%, meaning that one out of every four utterances spoken by the user was misrecognized and required reentry or correction. This level of performance is not acceptable for a production system. We believe that for a task such as the spreadsheet, utterance accuracy rates of at least 90% are essential for usability and rates better than 95% are highly desirable.

We can estimate the impact of real-time performance and high-accuracy recognition by subtracting the time lost to non-real-time response and error recovery

from the voice task time-to-completion. If we do so, we find that the voice-operated spreadsheet can allow for more rapid work than keyboard input, indicating that on the simple criterion of time-to-completion, a voice-operated spreadsheet would represent a real improvement over keyboard input. The dotted lines in Figure 5 show the recalculated values. In the present case, we estimate that a system capable of operating in real-time and providing at least 85% utterance accuracy would produce equivalent times-to-completion for voice and keyboard. Whether this is actually the case, of course, would require implementing and evaluating such a system.

The potential advantage of voice input is supported by an analysis of action times calculated separately for movement and entry instructions. In both cases, commands are entered more rapidly by voice. The median action time (including preparation and entry times) for numeric and formula entry is, for voice 1906 msec, and for keyboard, 3301 msec. The total time for movements (including all delays) is 200.4 sec for voice and 202.6 sec for keyboard. The reason that total task completion times are excessive is the consequence of recognition errors and the resulting loss of time taken by reentry and correction.

In addition to examining the consequences for task performance, we also need to evaluate how well the recognition system performs.

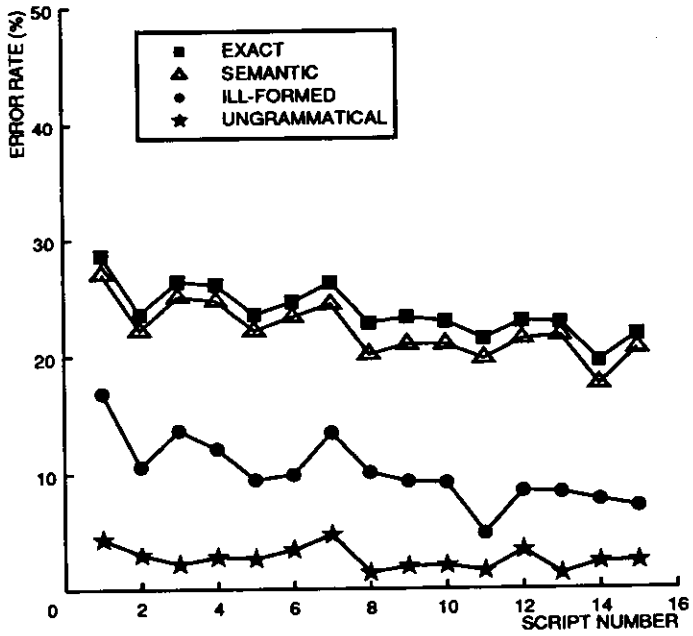
6.2 Speech Recognition Performance

To analyze recognizer performance we captured and stored each utterance spoken, as well as the corresponding recognition string produced by the system. All utterances were listened to and an exact lexical transcription produced. The transcription conventions are described more fully in [13], but suffice it to note that in addition to task-relevant speech, we coded a variety of spontaneous speech phenomena, including speech and non-speech interjections, as well as interrupted words and similar phenomena.

The analyses reported here are based on a total of 12507 recorded and transcribed utterances, comprising 43901 tokens. We can use these data to answer a variety of questions about speech produced in a complex problem-solving environment. Recognition performance data are presented in Figure 7. The values plotted represent the error rate averaged across all eight subjects.

The top line in Figure 7, taken from [10] shows **exact** utterance accuracy, calculated over all utterances in the corpus, including system firings for extraneous noise and abandoned (i.e., user interrupted) utterances. It does not include begin-end detector failures (which produce a zero-length utterance), of which there were on the average 10% per session. Exact accuracy corresponds to utterance accuracy as conventionally reported for speech recognition systems using the NBS scoring algorithm [8]. The general trend of recognition performance over time is improvement, though the improvement appears to be fairly gradual. The improvement indicates

Figure 7: Mean utterance accuracy across tasks.



that users are sufficiently aware of what might improve system performance to modify their behavior accordingly. On the other hand, the amount of control they have over it appears to be very limited.

The next line down shows **semantic** accuracy, calculated by determining, for each utterance, no matter what its content, whether the correct action was taken by the system[4]. Semantic accuracy, relative to exact accuracy, represents the added performance that can be realized by the parsing and understanding components of a spoken language system. In the present case, the added performance results from the "silent" influence of the word-pair grammar which is part of the recognizer. Thus, grammatical constraints are enforced not through, say, explicit identification and reanalysis of out-of-language utterances, but implicitly, through the word-pair grammar. The *spread* between semantic and exact accuracy defines the contribution of higher-level process and is a parameter that can be used to track the performance of "higher-level" components of a spoken language system.

[4]For example, the user might say "LET'S GO DOWN FIVE", which lies outside the system language. Nevertheless, because of grammatical constraints, the system might force this utterance into "DOWN FIVE", which happens to be grammatically acceptable and which also happens to carry out the desired action. From the task point of view, this recognition is correct; from the recognition point of view it is, of course, wrong.

The line at the bottom of the graph shows **grammaticality** error. Grammaticality is determined by first eliminating all non-speech events from the *transcribed* corpus then passing these filtered utterances through the parsing component of the spreadsheet system. Grammaticality provides a dynamic measure of the *coverage* provided by the system task language (on the assumption that the user's task language evolves with experience) and is one indicator of whether the language is sufficient for carrying out the task in question.

Grammaticality can be used to track a number of system attributes. For example, its value over the period that covers the user's initial experience with a system indicate the degree to which the implemented language covers utterances produced by the inexperienced user and provides one measure of how successfully the system designers have anticipated the speech language that users intuitively select for the task. Examined over time, the grammaticality function indicates the speed with which users modify their speech language for the task to reflect the constraints imposed by the implementation and how well they manage to stay within it. Measurement of grammaticality after some time away from the system indicates how well the task language can be retained and is an indication of its appropriateness for the task. We believe that grammaticality is an important component of a composite metric for the *language habitability* of a spoken language system (SLS) and can provide a meaningful basis for comparing different SLS interfaces to a particular application[5].

Examining the curves for the present system we find, unsurprisingly, that VSC is rather primitive in its ability to compensate for poor recognition performance, as evidenced by how close the semantic accuracy curve is to the exact accuracy curve. On the other hand, it appears to cover user language quite well, with only an average of 2.9% grammaticality error[6]. In all likelihood, this indicates that users found it quite easy to stay within the confines of the task, which in turn may not be surprising given its simplicity.

When a spoken language system is exposed to speech generated in a natural setting a variety of acoustic events appear that contribute to performance degradation. Spontaneous speech events can be placed into one of three categories: **lexical**, **extra-lexical**, and **non-lexical**, depending on whether the item is part of the system lexicon, a recognizable word that is not part of the lexicon, or some other event, such as a breath noise. These categories, as well as the procedure for their transcription, are described in greater detail in [13]. Table 8 lists the most common non-lexical events encountered in our corpus. The number of events is given, as well as their incidence in terms of words in the corpus. Given the nature of the task, it is not surprising to find, for example, that a large number of paper rustles

[5] *System habitability*, on the other hand, has to be based on a combination of language habitability, robustness with respect to spontaneous speech phenomena, and system responsiveness.

[6] Bear in mind that this percentage includes *intentional agrammaticality* with respect to the task, such as expressions of annoyance or interaction with other humans.

Table 8: Incidence of (some) non-lexical spontaneous speech tokens.

585	++RUSTLE+	4	++PHONE-RING+
206	++BREATH+	4	++NOISE+
43	++MUMBLE+	4	++DOOR-SLAM+
18	++SNIFF+	4	++CLEARING-THROAT+
13	++BACKGROUND-NOISE+	4	++BACKGROUND-VOICES+
11	++MOUTH-NOISE+	2	++SNEEZE+
10	++COUGH+	1	++SIGH+
6	++YAWN+	1	++PING+
5	++GIGGLE+	1	++BACKGROUND-LAUGH+

Note: The first column given the percentage and the second column the actual number of tokens for the given non-lexical token. There are 43,901 tokens in the corpus.

intrudes into the speech stream. Non-lexical events were transcribed in 893 of the 12507 utterances used for this analysis (7.14% of all utterances).

The **ill-formed** curve in Figure 7 shows the proportion of transcribed utterances that contain extraneous material (such as the items in Table 8). This function was generated by calculating grammaticality with both non-lexical and extra-lexical tokens included in the transcription. As is apparent, the incidence of extraneous events steadily decreases over sessions. Users apparently realize the harmful effects of such events and work to eliminate them (conversely, the user does not appear to have absolute control over such events, otherwise the decrease would have been much steeper).

While existing statistical modeling techniques can be used to deal with the most common events (such as paper rustles) in a satisfactory manner, as shown by [15], more general techniques will need to be developed to account for low-frequency or otherwise unexpected events. A spoken language system should be capable of accurately identifying novel events and dispose of them in appropriate ways.

7 Summary

We have described in detail the design and evaluation of a spoken language system whose operating characteristics approach the minimal requirements for a natural and easy to use system, incorporating speaker independence and continuous speech capabilities.

The design process is time-consuming and requires careful attention to the development of a speech language and the training of of the speech recognition component. The design of the speech interface must take into account characteristics

of the speech device, such as delay and recognition error, that may make it difficult for a user to work with.

An informative evaluation of a spoken language system must provide information about the system's performance as an interface in comparison with other modalities, as well as a detailed characterization of its characteristics as a recognition system.

8 Acknowledgements

The work described in this chapter has benefitted from the contributions of many people. I would particularly like to mention Bob Brennan, who implemented an initial version of the voice spreadsheet; Joe Polifroni, who performed the Wizard experiments and carried out the language analyses; and Michelle Sakamoto, who conducted the evaluation experiment.

References

[1] Chapanis, A. *Interactive Human Communication: Some lessons learned from laboratory experiments.* in: **Interactive Human Communication: Some lessons learned from laboratory experiments**, by A. Chapanis, edited by B. Shackel. Sijthoff and Noordhoff, Rockville, Md, 1981, pp. 65–114.

[2] Cohen, M. H. *Phonological structures for speech recognition.* University of California, Berkeley, April 1989.

[3] Gallaway, G. R. *Response times to user activities in interactive man/machine computer systems.* **Proceedings of the Human Factors Society–25th annual meeting.**, 1981, pp. 754–758.

[4] Hauptmann, A. G. and Rudnicky, A. I. *Talking to computers: An empirical investigation.* **International Journal of Man-Machine Studies**, vol. 28 (1988), pp. 583–604.

[5] Hauptmann, A. and Rudnicky, A. *A comparison speech versus typed input.* in: **Proceedings of the June DARPA Speech and Natural Language Workshop**. Morgan Kaufmann, San Mateo, CA, 1990, p. in press.

[6] Lee, K.-F. **Automatic Speech Recognition: The Development of the SPHINX System**. Kluwer Academic Publishers, Boston, 1989.

[7] Luzzati, D. and Neel, F. *Dialogue behaviour induced by the machine.* in: **Proceedings of Eurospeech 89**, edited by J. P. Tubach and J. J. Mariani. CEP Consultants, Edinburgh, UK, 1989, p. 601.

[8] Pallett, D. *Benchmark tests for* DARPA *Resource management database performance evaluations.* in: **Proceedings of the ICASSP**. IEEE, 1989, pp. 536–539.

[9] Pausch, R. and Leatherby, J. H. *A study comparing mouse-only input vs. mouse-plus-voice input for a graphical editor.* in: **Proceedings of the American Voice Input Output Society Conference**. AVIOS, 1990.

[10] Rudnicky, A. I., Sakamoto, M. H., and Polifroni, J. H. *Spoken language interaction in a goal-directed task.* **Proceedings of the ICASSP**, April 1990, pp. 45–48.

[11] Rudnicky, A. *The design of voice-driven interfaces.* in: **Proceedings of the February** DARPA **Speech and Natural Language Workshop**. 1989, pp. 120–124.

[12] Rudnicky, A., Polifroni, J., Thayer, E., and Brennan, R. *Interactive problem solving with speech.* **Journal of the Acoustical Society of America**, vol. 84 (1988), p. S213(A).

[13] Rudnicky, A. and Sakamoto, M. *Transcription conventions for spoken language research.* no. CMU-CS-89-194, Carnegie Mellon University School of Computer Science, October 1989.

[14] Rudnicky, A. I. and Quirin, J. L. *Subjective reaction to system response delay: A pilot study.* January 1990.

[15] Ward, W. *Modelling non-verbal sounds for speech recognition.* in: **Proceedings of the October DARPA Workshop on Speech and Natural Language**. Morgan Kaufmann, San Mateo, 1989, pp. 47–50.