A Robust Parser and Dialog Generator for a Conversational Office System

Christopher Schmandt
Barry Arons[1]

Media Laboratory
Massachusetts Institute of Technology

Proceedings, AVIOS Conference, 1986

---

[1]Currently with Hewlett-Packard Laboratories.

1 Introduction

This paper describes several key components of a speech based dialog system. This system was developed to explore opportunities for speech recognition in an integrated office environment. In order to achieve adequate performance despite speech recognition errors, it was necessary to develop a robust parser and a dialog generator which uses speech synthesis to converse with the user. Although designed for a specific application and recognition hardware, the underlying approach of the parser and dialog generator may be generalized to facilitate human-computer voice interaction in other environments.

2 Motivation

As part of a series of ongoing research projects exploring applications of intelligent voice telecommunications systems [Schmandt 85a, Schmandt 84] the authors designed a voice interface to an office workstation. This system, called the *Conversational Desktop* [Schmandt 85b], used voice to interact with telephone, scheduling, airline reservation, and audio memoranda functions.

The motivation for voice interaction with the office workstation is simply that speech is such a natural way of communicating in this environment. An office has major telecommunication needs which are in a large part voice oriented, and voice may be used in dictating or conversing with co-workers face to face.

As we were employing speech because of its ease of use, it was desirable to use *connected* speech recognition because of its tolerance for more natural spoken input. Connected speech recognition is more difficult than isolated word recognition, due in large part to difficulties finding word boundaries and variation in the pronunciation of words in an acoustical context [Oshika 75].

To use connected recognition effectively, an application must take the output of the speech recognizer, consider it as a very noisy signal due to errors, and filter out the errors from the words which were correctly recognized. At that point, a variety of techniques may be used to try to interpolate the missing words or interact with the user to complete the transaction.

Our approach to the problem used a parser designed specifically for speech and the errors produced by a connected speech recognizer. The parser communicates with a dialog generator capable of phrasing a series of questions in order to gather enough correct input to perform the user's request. This system employed a network of Sun Microsystems workstations, and was written in C under the Unix operating system. For recognition we employed an NEC DP-200 connected recognizer with a 150 word vocabulary.

3 System Components

A parser is used to analyze speech input and detect errors; this analysis is based on a formal description of the *syntax* of the set of input utterances. The parser also generates a description of the input, in a frame-like [Schank 77] representation which is convenient for both the dialog generator as well as action routines embedded in the application itself. What is unusual about the parser described in this paper is that, in addition to the usual syntax rules, it is designed to detect input errors and parse the remaining correct sentence fragments.

When errors are detected, they usually indicate that an insufficient number of the user's spoken words have been recognized to complete the requested action. A rich area of research, just touched in this project, is the use of semantic context to recover gaps in the input. When this is insufficient, the system needs to ask questions to complete its understanding of the user's request.

The dialog generator is passed knowledge about what is missing, in the form of the frame structure (used in a *slot-and-filler* approach) generated by the parser. This structure has fields which can be filled in with specific instances of various parts of speech, such as names, dates, times, etc. While such information could be used to phrase a simple query, such as "When?" or "With whom?", such questions may mask other errors which were syntactically correct (such as recognizing "Gary" for "Barry"). A more graceful approach takes advantage of the information already stored in the frame to simultaneously provide feedback to the user by echoing, such as "*When* do you want to meet with *Barry*?" Because this echoing technique can generate many possible sentences, a text-to-speech synthesizer is used for audio output and to guide the discourse.

4 Requirements for the Parser

The parser must analyze the error-prone speech recognition results and detect errors in such a way as to identify those tokens most likely to be correct (i.e, most likely to have in fact been spoken by the user). It must generate a representation of this postulated sentence, and invoke the dialog generator if the input is incomplete. The parser must also be re-entrant so as to continue analysis of the user's responses to system queries while retaining the current context.

Speech recognition errors can be classified in three categories: substitution, rejection, and insertion. A *substitution* error is one in which some number of words are spoken and the same number are recognized, but one or more of them is recognized incorrectly. A *rejection* error is one in which less words are recognized than were spoken, i.e. one or more input words were simply not recognized. An *insertion* error is one in which more words are reported than were spoken, perhaps because one input word was recognized as several words or perhaps breath noise was matched against a word.

Most conventional natural language parsers [Winograd 83] cannot cope with any of these problems, because they assume well formed input. Rather than detection of errors, their

sole task is to correctly determine the syntactic relationships of the input tokens. This is inadequate for voice input.

Previous *speech* parsers [Levinson 78] successfully dealt with substitution errors, by considering a number of possible choices for each word. Since Levinson's parser dealt with discrete speech in which each word has to be spoken separately, the parser assumed that the number of input tokens was correct and would fail if an insertion or rejection error occurred.

It should be noted that the parser described herein was designed to cope with one particular connected speech recognizer (the NEC DP-200) and as such is constrained by many limitations of this particular device. Features of other recognition hardware could be employed to improve our parser and dialog generator. Particular limitations included:

- The lack of any second guess information for each word or sentence.
- The lack of any measure of the recognizer's confidence in its selection for each word.
- Inadequate subsetting capability. Subsetting allows recognition to be limited to a particular set of words at any moment, thereby improving recognition.
- No indication of the relative difficulty of discriminating between various words in the vocabulary (confusability matrix).

Note: we use the term "words" relatively loosely here. Actually, we refer to an utterance, or a single template in the recognizer's vocabulary. To improve recognition, we often trained a phrase, such as "place a call" as a single, longer template rather than three separate ones. When this phrase is spoken it would generate a single token, and was dealt with in this manner by the parser

5 Parser Overview

A parser designed for speech input must cope with the various types of recognizer errors with no additional information about the input other than the tokens and the order in which they were returned from the recognizer. To achieve this, our parser has two unique aspects:

1. To cope with *insertion* errors, we consider all *subsets* of the tokens returned by the recognizer. This will also detect those substitution errors in which the substituted word is syntactically incorrect. For example, if *"Place a call to Barry"* was spoken, the parser would detect a recognizer error of *"Place a call to lunch"* but not of *"Place a call to Gary."*

2. To cope with *rejection* errors, our grammar accepts syntactically correct

sentence fragments as well as complete sentences. This retains information about what was correctly recognized even though it is incomplete.

There are two major software components in our system. The higher level extracts subsets of the tokens and calls the lower level repeatedly to test them against the grammar. A scoring metric is employed to select that subset which is most likely to correspond to what was spoken. A pruning technique is employed to minimize search time.

The lower level is a set of grammar rules. This level is written for YACC (see section 7) and simply applies a set of rules to analyze its input. While analyzing the output from the recognizer, a frame-like representation of the input is built up, which includes an indication of which slots in the frame are missing to complete the command. This lower level parser returns information that is used by the scoring metric in the higher level.

6 Substring Extraction and Evaluation

Insertion errors cause spurious tokens to appear in the output from the speech recognizer. Substitution errors replace correct tokens with incorrect ones. In both cases, we wish to detect the incorrect tokens so as to continue processing only those believed to be correct. This can be accomplished by considering all substrings (ordered subsets) of the recognizer output, and selecting the best by a scoring algorithm.

For example, if the string ABCD were returned from the recognizer, the following substrings would be considered: ABCD, ABC-, AB-D, A-CD, -BCD, AB--, etc. Each substring is analyzed according to the grammar (see section 7), to determine whether it is syntactically correct. For each syntactically correct substring, a score is computed to determine the most likely match between the input and what the user intended

Any substring which is either a sentence or a sentence fragment is a possible candidate. The correct candidate is chosen by applying a scoring metric based on the following:

1. **Completion:** a complete sentence is preferred to a fragment, as one is more likely to speak a complete command to the machine.

2. **Number:** of two possible substrings, the one with the larger number of tokens will be selected.

3. **Adjacency:** additional weight is given to adjacent tokens. For example, if the original input was ABCD, the substring ABC- has a higher adjacency score than AB-D.

Adjacency is a powerful metric specifically for connected speech, because a significant portion of the problem of connected recognition is *segmentation*, finding word boundaries. If it is postulated that the second token in an utterance is correct, it is more likely that the first and third tokens will also be correct because at least one of each of their boundaries must have been determined correctly [Rabiner 81, Zue 85].

An important point which makes this scheme useful is the definition in the grammar of sentence fragments in addition to complete sentences. This implies that if ABC is a legal and complete sentence, then all substrings including A-C (which has a token missing from the middle) are considered legal, and scored using the same metric. The motivation is correct acceptance of AB if the recognizer returns an incorrect sentence ABX. Even though AB is incomplete, it is an accurate indication of a portion of the speaker's intent and should guide further dialog.

7 Grammar and Knowledge Representation

Each phrase in the vocabulary is a particular *instance* of a small number of *syntactic categories.* For example, "Chris" is an instance of category NAME, and "place a call" (recognized as a single utterance) is an instance of category CMD_NAME, a command which requires a NAME for completeness. In general, this grammar was structured such that the category to which a command belongs indicates the number and types of the arguments to the command. Examples of these classes include CMD_TIME and CMD_NAME_AND_TIME (an instance of which is "schedule a meeting").

This categorization was stored in human readable form in a vocabulary file, which includes the prompts used to train the recognizer and generate dialog. The file also contains symbolic constants suitable for programmatic access to the categories and instances for each word (figure 1).

```
/**Utterance          Type          Instance **/

monday               -t DAY        -i 1
tuesday              -t DAY        -i 2

10                   -t HOUR       -i 10
11                   -t HOUR       -i 11

chris                -t NAME       -i CHRIS
barry                -t NAME       -i BARRY

clear                -t CMD        -i CLEAR

place a call to      -t CMD_N      -i PHONE
hangup               -t CMD_N      -i HANGUP

schedule a meeting   -t CMD_NT     -i MEET
```

Figure 1: Sample portion of a vocabulary file. Note that Types and Instances are referenced through symbolic constants.

This classification was used in the source code for a parser compiled under YACC, which would analyze each of the possible substrings. YACC is a standard Unix utility which

converts a context-free grammar into a set of tables for a simple automaton which executes an LALR(1) parsing algorithm [Aho 77]. YACC generates a parser based on the supplied grammar rules; when one of the rules is recognized, then user supplied code, an action, is invoked.

Note that the categorization of commands by the types and numbers of their arguments allows the parser to incorporate semantic knowledge as well as syntax, and also conveniently reflects the level of lexical description used for the recognizer. Thus, when one speaks "Schedule a meeting with Barry," the recognizer should match against two templates, "schedule a meeting" and "Barry." This is also reflected in the frame representation; although there are slots for both name and time, for example, some commands may require only one or the other of them.

```
sentence        : CMD
                | CMD_N name
                | name CMD_N
                | CMD_NT n_and_t
                | n_and_t CMD_NT
                | name CMD_NT time
                | time CMD_NT name
                ;

n_and_t         : name time
                | time name
                ;

name            : NAME
                | NAME AND NAME
                ;

time            : DAY tod
                | tod DAY
                ;

tod             : HOUR
                | HOUR MIN
                ;
```

Figure 2: Simplified example of YACC specifications. Rules for sentence fragments and user defined action routines are not included.

When each rule is executed, associated C routines set variables to be used by the scoring algorithm and fill slots in the frame abstraction to reflect the particular instance of the rule. For example, the 'CMD_N name' rule (figure 2) applied on "Place a call to Chris" sets the 'command' field to PHONE and the 'name' field to CHRIS.

While the command syntax may seem limited when decomposed into such rules, the vocabulary and grammar actually afford quite a bit of flexibility by allowing commands to be specified in several ways. For example, times could be of the form: Wednesday at 3, tomorrow afternoon, 3 o'clock tomorrow, etc.

In addition to defining complete sentences, as in the examples above, the grammar also contains rules for *fragments*, or incomplete sentences. Examples of such might be "Barry tomorrow afternoon," "tomorrow," or "place a call." A speech parser must recognize such fragments, because the recognizer may make a rejection error, and return such a fragment even if the user spoke a complete sentence. No further processing of the input is done at this level, as it is up the substring generator and scoring metric to accept the best choice, even if it is incomplete, for further consideration.

8 Dialog Generation

A robust parser is designed to extract as much information as possible from error-prone input. Detecting the errors allows the correct information in the input to filter through, but usually enough has been lost that it is not possible to act on the user's request yet.

The system was designed around a *conversational* model, in which dialog is employed to clarify ambiguous or incomplete input. As the parser was designed specifically for speech input, the dialog generator employs speech output. Since the number of sentences which could be generated is quite large, a text-to-speech synthesizer was used instead of pre-recorded replies (which are more intelligible).

The conversational model allows for a human initiated major task, or transaction, with a series of machine initiated sub-tasks, or questions, to clarify the user's intent. With each query, new information is gathered and added to the current frame. When a frame is finally complete, it can be passed off to an action routine which will perform the user's request.

As part of this model, we incorporated the concept of *indirect echoing* [Hayes 83] as a confirmation technique. A query by the computer contains *as much information as possible* about what has been assumed to be correct. Recognition errors may result in input which is syntactically correct but erroneous, and otherwise undetectable, such as substituting one name for another with a command requiring a person. Indirect echoing is an efficient way of alerting the user to such errors.

The parser, under guidance of the substring selector and scoring algorithm, produces both the frame, with slots filled by specific instances from the vocabulary, and a simpler structure which indicates what information is missing for this particular parse path. Because discrete speech is easier to recognize than connected speech, the dialog generator initiates a series of questions, each designed to elicit a single word response. Each question is phrased so as to echo as much as is known, or rather assumed, to be correct in the utterance.

For example, if the user said "Schedule a meeting with Chris Friday afternoon" and the recognizer reported "Schedule a meeting ... Friday." the first question generated would be "With whom do you wish to meet on Friday?" The query is generated from the frame information as a text string, and sent to the speech synthesizer to be spoken.

The dialog generator can also be used to generate queries that are not directly related to completing a user's command. After a user's request is completed, an *incomplete* set of tokens can be programmatically passed to the parser, and hence to the dialog generator. This will cause a *new* question to be generated, initiating further dialog.

For example, the user might initiate an interaction with "Schedule a flight to Chicago Friday morning." Note that the machine tracks the user's whereabouts, so it is not necessary to give the city from which you are leaving. The computer would first confirm this request, perform the appropriate action, and enter the event into its calendar database. The command "Schedule a return flight from Chicago" would then be passed to the parser, initiating the query "When would you like to return from Chicago?", as flight scheduling commands require a place and a time for completion. In effect, incomplete user input is simulated to cause the proper prompt to be generated automatically by the dialog generator.

9 Context in Sentence Completion

In many cases it is possible to fill in the slots of an incomplete command through pre-existing knowledge in the system. No attempt was made to create a general knowledge based system, but rather to apply simple rules which extract information from the context of the dialog and current state of the system. Use of this information can reduce requirements on the speech recognition hardware and tends to make the system more conversational.

For example, the command to disconnect a telephone conversation normally requires a name (e.g. "Hangup Chris"), as our system assumes multiple audio connections. If there is only one current connection (to Chris), then saying "Hangup" is sufficient, as the name can be deduced from context of the command. This not only lets the user say the short form of the command, but effectively increases the recognition rate if the name is spoken but not properly recognized. In a similar vein, "Schedule both of us a meeting" can be applied to the person on the phone, or to someone with whom one is currently meeting.

A more interesting example is illustrated by the "When is my flight?" command. If a destination city is not explicitly stated, the next airline reservation that occurs in the calendar is reported. However, if the user is on the phone with a person who lives in another city, the schedule is first scanned for flights to that city (unless there is a flight departing imminently, in which case it is reported).

10 Conclusions

A robust parser can be built from a fairly simple set of building blocks by designing it for the specific types of errors encountered in speech recognition. A descriptive grammar keyed more toward function rather than strict syntax facilitates both the writing of rules by the programmer and the generation of queries by the dialog subsystem. Use of indirect echoing in these queries by the computer helps the user detect and correct errors which the parser cannot find.

It should be pointed out that this particular parser was designed to cope with the limitations of a particular recognition device. Many other sources of information could enhance the decision algorithm if available. Some of these include: a confusability matrix for the vocabulary, knowledge of the lengths of utterances or stress in the sentence to weight probabilities for each word, or second guess results on the words recognized.

The authors believe, however, that many aspects of this parser/dialog generator combination are generalizable and could be utilized in a variety of human-computer voice interaction scenarios.

11 Acknowledgement

This work has been supported by NTT, the Nippon Telegraph and Telephone Corporation, as part of ongoing research into intelligent voice telecommunications. Further hardware support was supplied by NEC, Sun Microsystems, Speech Plus, and Digital Equipment Corp.

12 References

[Aho 77] Aho, Alfred V. and Ullman, Jeffrey D.
Principles of Compiler Design.
Addison-Wesley, 1977.

[Hayes 83] Hayes, P. and Reddy, R.
Steps Toward Graceful Interaction in Spoken and Written Man-Machine Communications.
Int'l J. Man-Machine Studies 19:231-284, 1983.

[Levinson 78] Levinson, S.E.
The Effects of Syntax Analysis on Word Recognition Accuracy.
Bell System Technical Journal 57(5):1627-1644, 1978.

[Oshika 75] Oshika, B.T. et. al.
The Role of Phonological Rules in Speech Understanding Research.
IEEE Transactions on Acoustics, Speech, and Signal Processing ASSP-23(1):104-112, 1975.

[Rabiner 81] Rabiner, L. and Levinson, E.
Isolated and Connected Word Recognition -- Theory and Selected Applications.
IEEE Transactions on Communications 25(5):621-659, 1981.

[Schank 77] Schank, R.C. and Ableson, R.P.
Scripts, Plans, Goals, and Understanding.
Lawrence Erlbaum Press, 1977.

[Schmandt 84] Schmandt, C. and Arons, B.
A Conversational Telephone Messaging System.
IEEE Trans. on Consumer Electr. CE-30(3):xxi-xxiv, 1984.

[Schmandt 85a] Schmandt, C. and Arons, B.
Phone Slave: A Graphical Telecommunications Interface.
Proc. of the Soc. for Information Display 26(1):79-82, 1985.

[Schmandt 85b] Schmandt, C., Arons, B., and Simmons, C.
Voice Interaction in an Integrated Office and Telecommunications Environment.
In *1985 Conference Proceedings.* American Voice Input/Output Society, 1985.

[Winograd 83] Winograd, T.
Language as a Cognitive Process - Syntax.
Addison-Wesley, 1983.

[Zue 85] Zue, V.W.
The Use of Speech Knowledge in Automatic Speech Recognition.
Proceedings of the IEEE 73(11):1602-1615, 1985.