A voice interface to a Direction giving program

James Raymond Davis

January 1987

Speech Research Group Technical Memo 2
The Media Laboratory
Massachusetts Institute of Technology

Abstract

This paper describes the user interface to Direction Assistance, a program which provides high quality directions for driving between two points in the Boston area. The interface employs synthetic speech and touch tone input to provide access to telephone callers. The interface is able to diagnose common user errors and explain itself to the user. This paper also discusses some of the problems of synthetic speech for this type of interface. The most significant limitation of presently available speech synthesizers is the lack of ability to specify prosodic features.[1]

Direction Assistance

Direction Assistance [3] is an program that provides directions for automobile travel within the Boston area. The program has a detailed street map covering about 11 square miles, with about 279 miles of streets, including bridges, tunnels, and one way streets. A routing algorithm produces routes that are both short and simple. A text generation program provides natural language descriptions of these routes. A graphics interface displays routes on a map. The task of the voice interface is to make this power available to naive users through the telephone, by using synthetic speech and and touch tone keypad.

The overall structure of Direction Assistance is shown in figure 1.

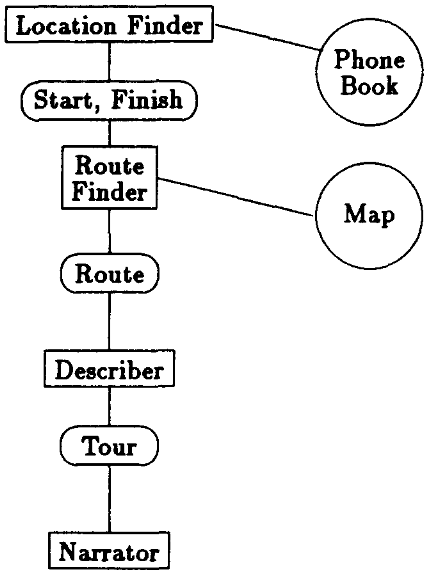

Figure 1: Structure of Direction Assistance. Boxes are programs, ovals are data structures and circles are databases.

The *Location Finder* obtains the locations of the start and destination, the *Route Finder* produces a route which is both direct and easily followed. The *Describer* produces a *tour*, a description of the route, and the *Narrator* recites the tour to the user in manageable chunks. The Location Finder and Narrator are the modules that communicate with the user. Both use a common set of interface routines to provide for a uniform interface.

The Location Finder and the Narrator differ in their behavior, because the Location Finder is input oriented, while the Narrator is output oriented. The Location

[1]This is a revised and expanded version of a paper that appeared in the Proceedings of the American Voice I/O Society conference of September 1986. The interface has changed, hopefully for the better, since the time this paper was written. This memo also supercedes an earlier memo titled "Giving Directions". The principle changes have been to add references to newer work.

Finder needs to obtain two locations, and it does this by asking questions and soliciting input. The user can specify location by giving either a street number and name, or by giving a telephone number (the program's data base includes an inverted phone book). It needs to get several different types of input - answers to yes or no questions, telephone numbers, street numbers and names. It has little to say, other than to guide the user through the input process.

The Narrator, on the other hand, has a lot to say, and needs no input from the user. Its main concern is *flow control*, making sure that it is going as fast as the user can write, but no faster. Both of these modules also have the secondary goals of keeping the user oriented within the interaction and recovering from errors.

The initial message

Direction Assistance is designed to be used by people with no experience with computers or synthetic speech. Almost every American is at least familiar with the concept that computers might talk, but they are not experienced with interfaces built with present day technology. There are lots of ways to go wrong. The interface tries to forestall some of them with its initial message:

```
This is Direction Assistance.  I can speak to
you, but I can not hear your voice.  To tell me
anything, you must use the keypad on your
telephone.  If you get confused any time while
you're using me, hit the key with the star or
asterisk.
```

The initial message conveys two important ideas. First it tells users how to communicate. Until recently, people expected to use the telephone to talk with people. Nowadays, people are beginning to get used to having "conversations" with answering machines, which speak, and then listen. Very few people know how to interact with a program that speaks often, but listens only for button pushes.

The second idea is that there is always help available, and always in the same way, by hitting the "star" key. This is a surprisingly difficult concept to convey. Examination of the log shows that many of users never used the help key, and this was even true for people who daily used computer systems that had "Help" keys. There are several reasons this might be so. First, this initial message is unexpected, and may have caught people by surprise. Second, the message has no direct relevance to the task at hand, and people may have been too impatient to listen to it. Much of the testing was done with a much longer message (nearly one minute) and this surely taxed people's concentration. Finally, people may not have understood what "star" meant. It is hard to find names for the keys without numbers. Some people prefer "asterisk". The "number sign" key is even worse, being called "pound sign", "sharp sign", "hash" or even "tic tac toe" by different people.

Different types of input

There are four types of input Direction Assistance needs from the user: answers to yes or no *questions*, *selection* from a small list of items, *numbers*, and *names*. We now consider each in turn.

Yes or No

The interface poses a yes or no question by stating a possibility and requesting the user to hit any key if it is true, thus for example "If you want to enter a different address, please hit any key now." If the user takes no action within a specified time the answer is no. The wording of the message always includes the consequence of the "no" if it is not obvious. This protocol is easier to use than one that requires an explicit action for either "yes" or "no", but can be slower, since the user has to wait for the full duration allowed if the answer is no. This duration must be long enough for the user to make a decision.

Selection

Selection means choosing one of a small list of items. There are two types of selection. In a *sequential* selection the system slowly reads a list of choices, pausing briefly after each, until the user designates one by striking a key. This is effectively a yes or no question ("Do you want this one?") for each element of the list. In an *enumerated* selection the system reads the entire list, assigning each item a number, then asks the user to hit the key for the number of the desired item. Each type of selection has its advantages and disadvantages.

Sequential selection is simple to use but can be very slow, since the user may have to listen to several undesired choices before hearing the one wanted. A second problem is that the user may not know which choice is the best without hearing them all. For this reason, if the user makes no selection, the selection routine offers to repeat the list.

Enumerated selection is harder to use than sequential selection because the user must remember and enter a number. The compensation for this complexity is that it can be faster than sequential selection. There are two reasons for this. First, the list is read faster, since the system need not pause to await a user action while reading. More importantly, if the user already knows the order of the options she can enter the number immediately, without waiting for the list to be read. An enumerated selection is appropriate when the same list of choices will be presented more than once.

Numbers

There are two types of numbers in this application, phone numbers and street address numbers. Phone numbers are entered in the familiar manner. The system simply collects the next seven button pushes. Street numbers require a delimiter, since the number of digits is not predictable. The delimiter is the number sign, since the only other key is the help key. Both phone numbers and street numbers require some additional checking, described below.

Names

The user selects street names by spelling them with the letters on the keypad. There are three reasons this is difficult: First, people are not familiar with the layout of the letters - they have to "hunt and peck" for letters. Second, the letters Q and Z, and the space character are missing. These are assigned to the 1 key, which bears no letters[2]

[2]In newer versions of the program, Q is on the 7 key, with PRS, and Z on the 9 key, with WXY.

The third problem is that each of the keys has three different letters, and there is no obvious way for the user to designate which of the three is intended. Other systems have resolved this problem by using more than one button push per character, either with a shift sequence (e.g using keys *, 0, and # to select one of the letters.) or by hitting a key N times to select the N'th letter upon it. Direction Assistance is able to bypass this problem because the set of all street names in Boston is much smaller than the set of all digit sequences. A given digit sequence will almost always specify only one street name. If it does not (e.g. "MILL" and "MILK") the system asks the user to make a sequential selection among them. Witten [13] found 8% collisions in a 24,500 word dictionary when using this approach. In this application there are much smaller number of collisions, just eight out of 1000 names. This is both because of the smaller size of the set and because street names are longer than dictionary words. The mean length of a street name is seven characters.

Absence of Context is a problem

A telephone interface is harder to use than a graphics interface (such as a form filling environment or a menu) because there is no *context* present while the user is entering data. The interface always prompts for input, but the user may not understand the prompt because of noise, inarticulate speech, or inattention. Even if the prompt is correctly heard, it must also be remembered, for it is not present during the time the user is entering the data. In a graphics environment the prompt remains visible.

The problems, then, are that the user may not know what to enter or how to delimit it. Neither of these are explicitly indicated, other than in the prompt.[3] If a closing delimiter is expected, and none arrives after a fixed delay, a timeout occurs, and the interface reminds the user that a delimiter is required.

Handling Hangup

The interface is more complicated than it needs to be, because there is no way to detect whether the caller has

[3]One possible solution is to employ acoustic icons, as discussed in [4]. The system might play a faint dial tone to indicate that phone number was expected, and ambient street noise to indicate that an address was expected.

hung up the phone. A user might hangup through frustration, or to recover from an error, or by accident. The only sign of a hangup is that no input is received. This means that every routine that expects input must have a timeout, such that if no input occurs, the user is deemed to have hung up. This timeout adds a further complication, since the system should not hang up if the user is merely slow, so in every case where a timeout occurs, the system first asks the user to indicate her presence by striking any key. This is described by [5] as a channel checking dialogue. This dialogue can be a problem, too, since it introduces a timing condition in input: when the dialogue occurs the user may be uncertain which keystrokes count as part of the channel checking dialogue and which are considered to be part of the numeric input.

The Location Finder

The task of the Location Finder is to get the locations of the origin and destination of the traveler. Logically, the task is as shown in Figure 2. The actual interaction can be more complicated than this. Interaction complications arise from deficiencies in the program's database, inherent ambiguities in input, and user errors. This section discusses these problems, and concludes with an annotated transcript of the Location Finder, showing error recovery.

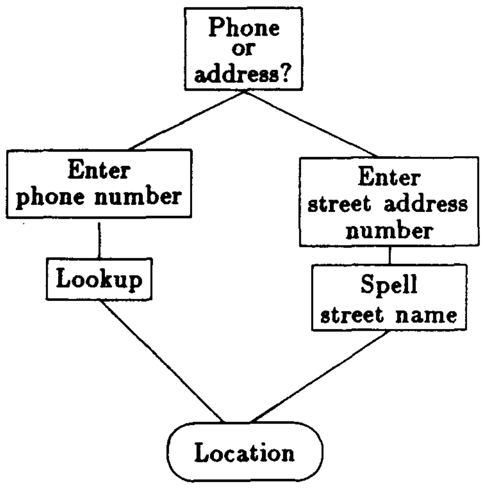

Figure 2: Location Finder flow

Handling Ignorance

The first source of complication is that Direction Assistance is not omniscient. The database covers only a limited area of Boston, and is not complete even within that area. The user may enter an address outside the known area, or a phone number that is not in the phone book. In the former case, the user will probably hang up, since the program can't give directions to a place it doesn't know, but in the latter the user may be able to supply an address. So the first complication is the need to recover from a failure to comprehend a location.

The program has a few special case checks for phone numbers that are outside the local area. If the first digit of the number is 1, the program informs the user that the 1 is superfluous. The first three digits of a phone number specify the local office, and are sufficient to tell whether the number is outside the area covered. If the first three digits are not recognized, the program informs the user that the number isn't known, suggests the possibility that the user was entering an area code, and asks for the number again. This is useless, of course, if the user really intended a distant number, but helpful for typing errors.

Handling ambiguity

The second problem is that a street address may not be unique. The interface accepts only the name of a street, and not the *type* (e.g. "Street", "Avenue", "Place", etc.), so an entry such as "10 Beacon" is not unique. This problem can can also occur when the user enters a phone number, if the phone book entry does not include a street type. This situation is an example of *partial incomprehensibility* as described in [5]. The system informs the user of the problem, and asks for a sequential selection from the alternatives.

A street address can be made precise either by asking for the town or neighborhood (by place) or by asking for the type of the street. Some ambiguities are better resolves by place, and others by type. If there are three instances of a street, with two in one city, and the third in a second city, then asking for the city is not enough to resolve the reference fully. If the three streets are all of different type, it is better to ask for the type. The interface uses whichever will yield the most information. A sequential selection is employed.

Places can be named by city or by neighborhood. When resolving by place, the system always uses the least specific name for a locale that suffices distinguishes the street from others. Thus if two streets are different cities, the city name is used, but if both are in the same city, the neighborhood name is used. The neighborhood of a street is obtained by using the zip codes of the street as an index into a database of neighborhoods, organized as a tree.

The interface could have been designed to always collect this additional information, but that would have imposed a cost on every user. In addition, this method is faster, since the clarification dialog requires at most two sequential selections, instead of spelling both street type and city name.

User Errors

The most troublesome part of the Location Finder is the selection of the type of location to be entered (phone or street address). A user might forget this step, or might enter to wrong value. The interface handles all these cases. If the user skips the selection, then the next keystroke will be part of either a phone number or a street address number. If the number doesn't begin with 1 or 2, the selection routine will detect the error. If it does, than it will appear to be a valid selection, and the system will then prompt the user for one of these two things. The prompt itself may be enough to inform the user of the error. If not, there are four cases:

The user is entering a phone number, and the first digit is 1. The system is expecting a phone number, because the user hit 1. The user begin to enter the phone number. After three digits have been read, the interface checks that the exchange is known to the system. If it isn't, an error message is issued, and the user must begin again.

The user is entering a phone number, and the first digit is 2. The system is now expecting an address number. The address number input routine assumes that street numbers in Boston never exceed four digits, so when the fifth digit is entered, an error message is issued.

The user is entering a street address number, and the first digit is 1. The system is expecting a phone number. When the user strikes number-sign to delimit the number, the phone number routine complains with the message "I thought you were entering a phone number, but you just entered a number sign, which you use only for addresses. If it was an accident, you can continue with the phone number, otherwise, enter number-sign again, and then you can enter the address.".

The user is entering a street address number, and the first digit is 2. Effectively, the first digit of the street address number is lost. The system does not detect this error. When the user strikes number sign, the system repeats the number. The user has a chance to abort the entry during the spelling of the street name, as described below.

This error checking is only possible because phone numbers and street address numbers have different syntax. A phone number is always seven digits, and the first three are chosen from a limited set. A street address number is always less than five digits, and is always delimited by a number-sign.

User Initiated Corrections

The user may detect an error while still entering data. The most frequently detected error is to find oneself in the wrong mode, that is, the user intends to enter an address, but the system expects a phone number, or vice versa. It is difficult to allow the user to correct mistakes, because there is no spare key to use for corrections. Star is reserved for help, and number-sign is sometimes is used for a delimiter (for numbers and names). The best solution is to use different abort conventions for the different types of input. The number sign key is always an abort key for phone numbers, and the zero is always an abort for names. For street numbers one aborts by entering star twice. This complication seems to be only way to provide the needed features.

Since the error correction mechanisms are not elegant, the initial prompts do not describe them. Instead, they are described by the "help" routines. If the user does not know how to correct an error, she will either hit "help", or will pause long enough for a timeout to occur, and this in turn will remind her to hit "help".

If the user makes a mistake that is not syntactically invalid (enters the wrong phone number, spells a different street) there is no way for the system to detect the error. Nor does the system confirm the location with the user. It was felt that adding a confirmation step (a yes or no question) would be too troublesome for the user. (In retrospect, this was a mistake.)

An annotated transcript

What follows is an annotated transcript of a dialog with the Location Finder. This dialog shows a few of the error recovery mechanisms in the interface. Audio output is shown in this font, keypad input **like this**, and comments are in italics.

Tell me where you are now. You can give me either a telephone number or an address. If you know the number of your phone, and it is not a payphone, then enter 1. Otherwise enter 2 and I'll give you further instructions.

This is an example of an enumerated selection.

1

Enter the phone number

867

The user realizes she's made a mistake, but doesn't know what to do. A long pause ensues, and the input times out.

For help enter *

The first layer timeout handler simply reminds the user how to get help.

*****

You are entering a phone number for your origin. So far you've entered 867. If you make a mistake, or if you don't want to enter a phone number, enter number sign.

#

This restarts the Location Finder. Enter 1 for a phone number, 2 for an address.

The system uses a shorter prompt the second time. This prompt also serves as implicit confirmation of the abort from entering a phone number.

1

Enter the phone number.

8761111

876-1111. 245 First Street, Cambridge.

Confirms the entry and gives the location.

Tell me where you want to go. Enter 1 for a phone number, 2 for an address.

Shorter prompt. The user intends to enter an address but forgets to preface it with 2.

3

Wait! If you want to enter a phone number, you must first dial 1. If you want to enter an address, you must dial 2.

The enumerated selection routine detects this error, unless the number begins 1 or 2, in which case other handlers detect the error.

2

Enter the number of the address, followed by number sign.

33

The user forgets to enter a closing delimiter. Pause and timeout.

So far the number is 33. If you're through, enter number sign. Otherwise, please continue. If you make a mistake enter star.

#

33.

Echoes the street number

Spell the name of the street, followed by another number-sign. Spell the entire name, but don't include words like Avenue or Street.

33462676

33 Edinboro St, Boston.

Only one name is "spelled" that way, and only street in the area is named "Edinboro". No disambiguation is required.

The Narrator

Recall that the output of the Describer is a *tour*, a sequence of actions to make to get from start to finish along the route. The task of the Narrator is to read the lines of the tour, one at a time, while the user writes them down. It is necessary that the system be able to repeat lines and to go back to the beginning of the directions. After each line, the user makes an enumerated selection for one of three actions: repeat the line, move to the next, or start over. This is the most important use of enumerated selection in the interface, since this selection is made many times per session.

Difficulties of Speech Synthesis

Synthesis of speech by rule is essential for this application, because the text to be spoken is not known ahead of time. But synthesis by rule has drawbacks. Synthetic speech is harder to understand than recorded natural speech, and understand synthetic speech imposes extra more cognitive load on the listener [7,8]. Text to speech rules mispronounce some words. This section examines these problems and the solutions used here. This interface used a Digital Equipment Corporation Dectalk, but all available synthesizers have similar problems.

The problem of intelligibility is most acute when reading directions. Names must be decoded on an acoustic phonetic basis alone. Neither syntax, semantics, nor pragmatics provide any constraint on names. The names of streets are usually unfamiliar to the user. Distances are similarly unconstrained. The Narrator copes with this by spelling all names in a line whenever the user asks the line to be repeated.

The Dectalk's pronunciation rules are good but not infallible for ordinary English. Perhaps a dozen words required either explicit phonetic pronunciations or explicit stress. For example, in this application, the word "address" is a noun, not a verb. A more annoying problem is that the rules are quite poor on proper names. Spiegel found that about 29% of surnames were pronounced incorrectly [12]. I found a similar result: The Dectalk's pronunciation was unacceptable for 220 of the approximately 1000 names. For these words, the interface uses the Dectalk's exceptional pronunciation dictionary facility. Ken Church has suggested that pronunciation of proper nouns might be improved by attention to etymology [1].

Prosodics

Prosodics refers to variations in pitch and timing of words. It can be difficult to obtain natural prosody with the Dectalk. Errors of prosody may simply sound unnatural, though comprehendable, or they may introduce great confusion into the interaction. Prosodics affect both the lexical content and the state of the entire interaction.

It is a fact of English intonation that the final word in a compound street name is stressed (e.g. First 'Avenue, First 'Boulevard) *unless* the word is "Street". Lists of letters also require special care. Since isolated letters are easily confused, they must be read slowly. Pitch is adjusted up at the beginning of the list, and downward just before the last element of the list.

Rhetorical constructions also required extra work on their prosodics. The error message given above ("If you want to enter a phone number, you must first dial 1. If you want to enter an address, you must dial 2.") uses a parallel construction. To be most easily understood, it should be spoken with accent on the digits.

Prosodics are also used to indicate the state of the discourse. The error message just mentioned is prefaced by an exclamatory "Wait!". This message is an *interruption*, not a response, so the emphatic tone produced by the exclamation mark tells the user that something unusual has happened, that something is wrong. In this case the Dectalk's prosodics were just what was needed.

A second discourse function of prosodics is the marking of turn endings. In this application, the interface is the only party speaking, so the issue of turn endings reduces to informing the user when input is required and also when input is not required. In natural conversation people use a mixture of cues from syntax, pragmatics, and prosodics to recognize the end of a speaker's turn, and the opportunity or obligation to reply [9] [10]. The limited prosodics of synthetic speech deprive the interface of one powerful method of indicating the state of the conversation. This leads to a problem most obvious when the system asks a yes or no question. A typical question, e.g. "If you want to enter a different address, hit any key now. Otherwise I'll ask for a new street name." is best expressed as two sentences, one for each alternative, but users often answer as soon as the first sentence is complete, lacking any cue that indicates that the turn is not finished.

Turn taking is even more of a problem for the Narrator. The interface requires the user to hit a key in order to hear the next set of instructions. Since an instruction may be several sentences long, the user has no syntactic clue as to when a chunk has ended. As a result, users do not always understand that the system is waiting for them instead of the other way around.

Further work on prosodics

Prosodic tuning is not difficult. The Dectalk provides a variety of means to affect the pitch and timing of utterances, including inserting explicit pauses between words or inserting stress or de-emphasis markers, or by altering the comma pause duration, or using the "pitch change" characters. The difficulty is simply that so much of it is required, and the means are not very powerful. The Dectalk does allow the programmer to explicitly designate duration and pitch for a phoneme, but this facility is impractical, since any attempt to do so imposes a monotone pitch on all unlabeled phonemes. It is not possible to supply a pitch contour for only a portion of the utterance.

It is not surprising that the Dectalk's rules fail to provide natural prosody, since its production would require full comprehension of the matter to be synthesized, and this is clearly not to be expected. What would be a reasonable next step for products like the Dectalk, or systems that use them, would be to allow the programmer to explicitly mark the utterance for, e.g. rhetorical construction, and to employ intonational rules to synthesize the correct pitch contours for the desired effect. Research is required to discover both useful representations for this information and rules for synthesizing the correct intonation given the required marking. Such work in underway here at the Media Lab and also at Bell Labs [6,2].

A second main area of research is to provide a more natural method for listener control of speaker output. When people listen to spoken directions they typically use much more subtle means to let the speaker know how fast to go, or when to pause - they make inarticulate noises which we usually notate as "uh-huh" or "mm-hmm", sounds we call paraverbals or *back channels* [14]. A program capable of recognizing and responding to listener's paraverbals would be much more natural than one which used keypad presses. This is now a subject for research at the Media Lab[11].

Conclusion

Synthetic speech and key pad input are sufficient for construction of a robust and easy to use interface that can gather a wide range of types of data input. Research into synthesis of prosody by rule will make programming interactive systems easier. Until such systems are available, speech synthesizers should allow programmers to specify prosody explicitly. Recognition of listener paraverbals will make more natural interfaces possible.

Acknowledgments

Direction Assistance was written by the author and Thomas Trobaugh during the summer of 1985. I am grateful to those who tested the interface and made useful suggestions, and to Thinking Machines Corporation for supporting this work. Further development has occurred at the Speech Research Group at the Media Lab. At this writing, one version of Direction Assistance is part of the "Age of Intelligent Machines" exhibit touring science museums across the United States, and another is installed at the Computer Museum in Boston. These installations differ from the one described in this paper in that they do not have a telephone number to address database.

Bibliography

References

[1] Ken Church. Stress assignment in letter to sound rules for speech synthesis. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 246–253, July 1985.

[2] James R. Davis and Julia Hirschberg. Automatic generation of prosodic support for discourse structure. In *Proceedings of the Association for Computational Linguistics*, page (submitted), 1988.

[3] James R. Davis and Thomas F. Trobaugh. *Direction Assistance*. Technical Report 1, MIT Media Laboratory Speech Group, Dec 1987.

[4] William W. Gaver. Auditory icons: using sound in computer interfaces. *Human Computer Interaction*, 2(2):168–177, 1986.

[5] Phillip J. Hayes and Raj Reddy. Steps towards graceful interaction in spoken and written man-machine communication. *Int J. Man-Machine Studies*, 19:231–284, 1983.

[6] Julia Hirschberg and Janet Pierrehumbert. The intonational structure of discourse. In *Proceedings of the Association for Computational Linguistics*, pages 136–144, July 1986.

[7] Paul A. Luce, Timothy C. Feustel, and David B. Pisoni. Capacity demands in short-term memory for synthetic and natural speech. *Human Factors*, 25(1):17–32, 1983.

[8] David B. Pisoni, Howard C. Nusbaum, and Beth G. Greene. Perception of synthetic speech generated by rule. *Proceedings of the IEEE*, 73(11):1665–1676, Nov 1985.

[9] Harvey Sacks, Emanuel A. Schegloff, and Gail Jefferson. A simplest systematics for the organization of turn-taking for conversation. *Language*, 50(4):696–735, 1974. Reprinted in*Studies in the Organization of Conversational Interaction*, J. Schenken, ed., Academic Press 1978.

[10] Deborah Schaffer. The role of intonation as a cue to turn taking in conversation. *Journal of Phonetics*, 11:243–257, 1983.

[11] Chris Schmandt. Employing voice back channels to facilitate audio document retrieval. In *Proceedings*, page to appear, ACM Conference on Office Information Systems, 1988.

[12] Murray F. Spiegel. Pronouncing surnames automatically. In *Proceedings of 1985 Conference*, American Voice I/O Society, Sept 1985.

[13] Ian H. Witten. *Principles of Computer Speech*. Academic Press, 1982.

[14] Victor H. Yngve. On getting a word in edgewise. In *Papers from the Sixth Regional Meeting*, pages 567–578, Chicago Linguistics Society, 1970.