1-1988

A user interface management system for voice I/O applications

Brian A. Nadworny

Rochester Institute of Technology

School of Computer Science

A User Interface Management System for Voice I/O Applications

by

Brian A. Nadworny

A thesis, submitted to
The Faculty of the School of Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science.

Approved by _____ 10/14/1988

John A. Biles, Rochester Institute of Technology Date

_____ Oct 7, 1988

James Wilson, Eastman Kodak Company Date

_____ 17 Oct 88

Peter Anderson, Rochester Institute of Technology Date

Abstract

With the use of voice technology increasing, a need for proper end-user interfacing is necessary. To deal with voice I/O applications, a user interface management system (UIMS) is suggested. This will alleviate the frustrations of poorly designed interfacing. It will also remove the burden from the systems designer of implementing the end-user interface and allow him/her to concentrate on the application. The end-user interface will then be designed by an expert in dialogue design with the assistance from the UIMS. This thesis presents a UIMS model to be used with voice applications.

Table of Contents

Table of Figures

Chapter I - Introduction

An increasing number of people are being introduced to the world of computers daily. The dynamics of the interaction between people and computer applications determines the frequency of their usage and the frustration level of the user. A system called a User Interface Management System (UIMS) has been developed to help control such interaction.

A UIMS is a system tool used to establish dialogue for the purpose of satisfying the communication needs of the end-user. The result of utilizing this system tool is the decreased dependency on developing specific dialogue software for individual applications; the application developer no longer needs to consider how to implement the communication needs and can now concentrate on the application itself.

A variety of input/output (I/O) devices can be used with a UIMS. Of special interest is voice I/O, which is one of the newest devices to become integrated into computer systems.

Currently, the voice industry has been described as "solution in search of a problem" [PECK84] [COHE85], a technology looking for an application. People have not yet seen a real need for voice I/O [RUBI84]; however, some companies are currently trying different marketing techniques to increase the usage of voice I/O in the interface market. In this author's opinion, voice I/O, with

the appropriate software tools, will become one of the accepted devices used for man-machine communication. It has been shown that end-users would like to be able to speak to their systems and be understood [MART87]. Voice I/O products will take hold when the middle ground between the system developers and the end-users is established.

This thesis provides background information on UIMS and voice I/O interfaces. In addition, a design of a UIMS utilizing voice I/O is submitted. Some of the problems of voice interaction and possible solutions are discussed. This information should be of interest to interface designers and end-users. Understanding the capabilities and tools that are available in a UIMS system will help bridge the gap between interface designers and end-users and bring about the development of a well-designed interface for the application.

Terminology

Before proceeding, it is necessary to define or differentiate among the following terms: end-user, dialogue designer, systems designer, application designer and UIDS.

End-user: the last person on the chain of software development to use any software packages or systems.

Dialogue designer: a specialist in interface dialogues and types of interfaces; person who is knowledgeable in creating a proper interface to satisfy the end-user and fulfull the needs of the application [COHE85]. Though the interface designer is a user of a UIMS, reference to him/her will be made by his/her title and not as the "user" to avoid confusion. A more detailed description is presented later in Chapter V.

Systems designer: a person skilled in the designing, building, and implemention of the final application system.

UIMS designer: the person who creates the UIMS for the dialogue designer to use.

A User Interface Development System (UIDS): is a system used to create a UIMS. The term UIDS was developed since all user interface management systems that have been created emphasized development and not management of user interfaces [HILL86]. Some people prefer the term UIDS since it reflects more of the repetitive process of development of a UIMS, but in this thesis the term UIMS will be used for all aspects of the development and implementation of the system.

Chapter II - User Interface Management System

UIMS Definition

A User Interface Management System (UIMS) serves as a mediator between the end-user and the application package (see Figure 1). A UIMS can also be viewed as a system that acts as a translator in the dialogue between the end-user of a system and the application part of that system. The design of the UIMS is to "encourage interdisciplinary cooperation in the rapid development, tailoring and management of the interaction in an application domain across varying devices, interaction techniques and user interface styles" [BETT87]. All feedback and presentation of data to the end-user is handled through the UIMS. Therefore, the UIMS must be aware of both the needs of the end-user and the application. The UIMS is also capable of handling the dialogue between multiple applications (see Figure 2).

Figure 1. Simplified version of the UIMS. The arrows represent the flow of information.

Figure 2. The flow of information between multiple applications through a UIMS.

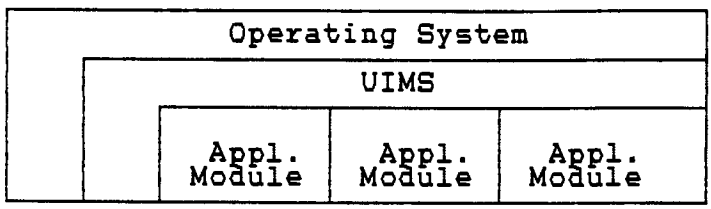


Figure 3. Structure of an externally designed UIMS.

UIMS responsibilities

The UIMS is responsible for three groups of information that are passed via the UIMS. There is the information generated by the end-user for the application, the data generated by the application for the end-user, and the data generated by one part of the application for another application. The UIMS designer must consider these three different perspectives and their impact on the different types of data handling that must be built into the UIMS. Since the data going to and from the application will most likely stay in the same format, the UIMS can simply reorganize it if necessary and then send it to the next application. All data going between the application and the end-user will be reformatted so that each side can be presented with a format that is most appropriate for it. Therefore, it is important that the UIMS know what is being

transferred and where it is going at all times.

UIMS layers

One way of designing a UIMS model is in layers to handle the flow of information. Each layer has a specific function in the translation of the information [COHE85]. From the end-user's and application's perspective the information that it would receive would be in the proper format and would not have to be translated in any way to be used. The central layer, which is normally percieved to be the UIMS, is the dialogue manager [LANT87]. This is the layer in which the dialogue designer does most of his/her work and is the major concentration of some UIMSs [HIX86]. On either side of the dialogue manager are layers for translating the dialogue to formats that the end-user and application can understand.

UIMS support to the end-user

The UIMS should provide support to the end-user for several capabilites. The support should include: help for the end-user, embedded training, error avoidance and recovery, and defaults [BETT87]. It is up to the dialogue designer to provide these features, which are supported by the UIMS. The end-user sees only what is presented to him/her by the UIMS. The data and queries generated by the application program are presented in an understandable manner to the end-user via the UIMS.

External design of the UIMS

An externally designed UIMS allows it to have complete control over the application. It also allows the application to concentrate on the computational parts of the system as opposed to viewing the UIMS as an abstract device handler with which it tries to communicate. This occurs in primitive graphical packages, where full control is coming from the application software, which must not only handle the application, but the graphical interface to the user as well. The externally designed UIMS also alleviates problems when attempting to separate the user interface from the semantic component [TANN85]. An externally designed UIMS would be able to use the user interface to control the application modules, to have access to all parameters and data, and to pass this information to the user in the form of semantic and syntactic feedback. As described in [BETT87], "A true UIMS provides support for external control, while a toolkit only supports internal control."

Comparison between a UIMS and other software tools

A UIMS is similar in many ways to other major components of a computer system. It is a set of tools that can handle the specific communication needs between the end-user and the application. Similar to an operating system, the UIMS handles and maintains transparency to the end-user so that he/she is unaware of all the complex transactions that occur and only concerns him/herself with

the actual application of the computer system.

As shown in Figure 3, the UIMS exists between the operating system and the application. It is possible to have the UIMS control the end-user interfacing to the operating system. Since the operating system is also an application, from the perspective of the end-user, the UIMS could control all messages going to the end-user. This would maintain a consistent interface to the operating system that would handle the needs of a variety of end-users.

The UIMS can be compared to other software tools such as a Data Base Management System (DBMS) and graphical packages. A DBMS is used for both the development of a data base system and in the actual use by the end-user. Like a DBMS, it communicates from the application program to the detailed levels of data interaction. The inner workings of the DBMS, graphical packages, and the UIMS are transparent to the end-user. The main difference between a UIMS and other management systems, such as DBMS or graphics components, is that a UIMS is in control of the application, whereas other components are controlled by the application (by application calls).

When a UIMS is involved in a system, the entire application system is controlled by the UIMS. Unlike other software packages, the UIMS is operated externally to the application software (see Figure 3). Normally, the application software will execute other transactions and

determine when they will execute, for how long, and what parameters they will use. Since the UIMS is external to the application software, it is the UIMS that controls the application and determines what it will execute in the application.

Benefits of a UIMS

One of the main benefits of a UIMS is that it removes the burden of end-user interaction concerns from the application designer so that he/she may concentrate on the application. The interface to the end-user will be written by the dialogue designer while the application is being written by the application designer. The section that connects the application to the UIMS can be worked on jointly by both the application and the dialogue designers.

Another major benefit of using a UIMS is that it is possible to design the UIMS to handle a variety of applications and act as a consistent interface to the end-user. The UIMS should be able to be interfaced to an application without redesigning all or major parts of it [ENDE83]. This will also eliminate the lack of consistency in user interfaces by using the same or similar user interface for a number of different applications.

The benefit of a voice UIMS

In reviewing recent reports on the current trends of UIMSs [OLSE87], the necessity of the UIMS is questioned.

The current UIMSs might not fulfill all of the expectations of the original concept. After reviewing the material on current and hypothetical user interface management systems, which are centered around graphical systems, this author feels that a contribution can be made from the transfer of the UIMS to other areas of interface technology, specifically voice. With the trends in voice technology expanding, there will arise a need for the application/UIMS dichotomy that has been experienced in the graphics world. Misunderstandings of voice technology by end-users and the misrepresentation of the technology by the developers and media have led to a void in the interface. A UIMS could solve some of these problems without creating new ones.

A UIMS system can help the dialogue designer to decide on the best method to use when dealing with an end-user. This would allow rapid changes of the dialogue to be more compatibile with the end-user of the system [WONG85]. It would also allow the interface designer to view his work in action during development. This would give him/her immediate feedback on his/her design. The UIMS system can also be used to keep track of all types of input and later be used to statistically analyze all the information to find the areas most and least used in the dialogue. It may also be used to suggest areas to be modified or strengthened.

A UIMS system is especially important with voice since there are very few voice packages that handle the aspect of easy-to-design user interfaces for an application. In the

graphics world there are many packages that contain tools to talk with the graphics interface. These tools allow the end-user to have full control over the graphics and input devices. The voice industry has made little effort to provide support for voice in the same way the graphics industry has with the development of graphical interface tools. Most of the concetration shown in the voice products has been towards accuracy rather than useability. There have been no standards set up for interfacing voice to a system. This makes transporting voice related software from one system to another difficult. In the graphics world there are many different standards, but they are similar enough to make transportation easier than in the voice world.

UIMS limitations

A UIMS is not the answer to all problems in the communication dilemma between humans and machines. There are some things that a UIMS cannot do. A UIMS cannot compensate for end-user environment and random interference. The environmental influences that affect an end-user's input or perception of output must be planned for by the interface specialist. The interface specialist working with the system designer must come up with a plan for the most appropriate setting in which work is to be performed. It is the obligation of the dialogue designer to design an interface that can handle all possible input choices. The

dialogue designer must also arrange for training of the end-users if applicable so that they become familiar with the types of input and are adept at working with the interface.

Summary

In summary, a UIMS is a system that is beneficial for the end-user. It handles the dialogue control between the end-user and the application. It also makes it easier to design the complete system since it has been broken down into different levels of complexity, which can be handled by two distinct groups: the system designers and the interface specialists. The UIMS physically exists between the operating system and the application software and logically exists between the end-user and the application. The placement of the UIMS over the application is called an external design. This has been said to be a true UIMS [BETT87], while an internal design would be called a toolkit or user interface toolbox. The entire application is controlled by the UIMS, which allows the application to concentrate on the areas not involved with end-user communication. The UIMS cannot handle all problems of human-computer interaction. However, it will aid greatly in voice I/O, and is easier to build than having the application handle everything. The UIMS encourages the consideration of all aspects of human-computer interaction, that might not occur otherwise.

Chapter III - Communication and Feedback

The transfer of communication techniques to computers

The human factors of a dialogue are important to consider if the dialogue is to be accepted or rejected by an end-user. Methods that the end-user will be using to communicate with the application are important. Equally important is the way that the results from the application will be presented to the end-user.

To help the communication between humans and machines, it is sometimes important to transfer some of the methods used for human-to-human communication to human-to-machine communication. When communication occurs on a human-to-human level there are signs that the information has been received and understood. This could be as subtle as a nod of the head, a returned glance, or a humming sound. It could also be a word for word repetition or an interpretation of what was said. This is called feedback of information. This and other human-to-human communication techniques should be incorporated into man-machine interfacing to maintain a certain degree of comfort similar to the manner in which humans communicate [RICH84] [CHAP65]. It will not be until the progress of artificial intelligence has reached a certain level when we can have the full interaction between humans and machines as we have in human-to-human communication.

Advantages of feedback

Feedback from voice systems implies providing the end-user information that input was received by the system. Feedback is an important concept in communication, and a voice interface should include it in its design and implementation [CHAP65] [SCHU85]. With voice entry systems it is very important to reassure the end-user, especially a new end-user who is unfamiliar with the concept of speech recognition, what is happening at all times.

There are some situations that use voice input without feedback. The U.S. Postal Service uses speech recognition with a limited vocabulary of only 10 digits. Since the vocabulary is small and the recognition errors are minimal it is felt that feedback would only slow down the process [SCHU85].

It should be the decision of the dialogue designer to establish what kind of feedback should be used in what situation. This should be done after a thorough examination of the interaction needs of the end-user.

Types of feedback

There are two major types of feedback that can be used in a voice entry system: visual and auditory feedback [SCHU85]. Tactile and other types of feedback will not be discussed here since they are not as frequently used as visual and auditory feedback.

Visual Feedback

The L.E.D. as feedback

An example of visual feedback is a Light Emitting Diode (LED). Once an end-user has voiced an entry the LED can light up to show that some input has been received. However, one LED is usually insufficient to completely reassure the end-user that the correct input was received. It only informs the end-user that input was received. The end-user has no idea, from the indication of the LED, that the correct match, word for spoken word, was made. The end-user is only aware that some match has been made and that the input was not rejected. A rejected input might be indicated by the color of the LED or the rejection might be simply the failure of the LED to light up. After the end-user develops confidence in the reliability of the system, feedback would become secondary in nature and the input would be assumed correct. In certain applications where it is easier to modify an input later on in the application, this type of system might be sufficient.

Using a series of LEDs for feedback

Another type of visual feedback can be provided by using a series of LEDs. These may be used to display to the end-user the volume of the input. This will provide the end-user with a method that he/she may use to control his/her volume when using a speech recognition unit for voice entry. This is critical for recognition since the

internal representation of the voice might change with the increase of the volume. On some voice entry units there are three LEDs for description of volume: a lower, middle and top [KURT85]. The lower LED (green) represents normal background noise and speech input. If the lower LED is not lit up when input is present, then this tells the end-user that the microphone he is using may not be on or he/she is not using the proper volume when speaking into the microphone. If the middle LED (yellow) lights up, it means that the volume of the end-user is tolerable and he can proceed with the input. The top LED (red) indicates that the incoming signal is too strong. In this instance the end-user could either lower the gain on the microphone or lower his/her voice [KURT85].

Use of a CRT screen for feedback

One of the most frequently used types of visual feedback is the terminal screen or Cathode Ray Tube (CRT). There are several types of responses that can be presented on a CRT. The actual text of what the end-user said could be printed, or a symbol that corresponds to the input value could be presented for verification. An example of this could be if the end-user said: "ONE", the display might read: "1" or "ONE" or "I". A graphical representation of what the end-user has said may also be displayed on a CRT. If the end-user mentions a part number, instead of displaying the part number the CRT could display a diagram

of the part. This is very helpful when working on a Computer Aided Design (CAD) package with a voice input system. Instead of drawing parts with the package, predefined parts in a library could be called for display by voicing the proper part. These parts can then be manipulated within the CAD system package.

Auditory feedback

Auditory feedback is the other technique used for "handshaking" with the end-user. On the lexical level, the system could provide feedback by simply providing a repetition of what has been. This could be done if the end-user is wearing a headset with microphone and headphones or by using an external speaker. When the end-user speaks, the input is sent to the recognition unit which can repeat the word back to the end-user. It should be noted that certain studies have been done showing that auditory feedback was too slow to be used as feedback for voice input [SCHU85]. However, if the circumstances do not allow for visual feedback, such as with eyes-busy hands-busy applications, this may still be faster than other types of data entry and feedback.

Voice synthesis as auditory feedback

One type of auditory feedback is synthesized voice response or text-to-speech [DECT85] [CATE83]. After the end-user has given a command, which has been converted to ASCII, it can then be sent to a voice synthesizer for

feedback to the end-user. Due to the time factor needed for the computation of the input and translation to the output, this method often slows the end-user down. Its benefits are that the end-user is assured that the actual commands spoken into the voice entry unit were correctly received, as opposed to the above method of just listening to one's own voice. The dialogue designer may even design the system so that a correct response is given to the command instead of just an echo of the user's command. An example of this would be if the end-user gave the command "on," the reply might be "system is on" to indicate the command was received and an appropriate action had taken place. Once the end-user is reassured that the system is receiving the input accurately, this method of feedback could be turned off and changed to a more simplified auditory or visual feedback such as the singular LED. A more technical discussion of voice synthesis will be presented later in this thesis.

Using a tone for feedback

A tone might be useful as a simplified auditory feedback to an end-user on the lexical level. If the end-user hears the tone, he is aware that the recognition unit has received the input. This is similar to the singular LED type of feedback. There also can be a variety of tones used for different feedback needs of the end-user. A low pitched tone might represent a valid input. A high-pitched tone might represent an input that was not

recognized for any reason. This method of feedback allows a faster speed of response than a voiced output response. The disadvantage is that there is no verification on the semantic level. If the end-user says word "A" and it is recognized as word "B" the recognition unit still views the word as a valid word and sends a valid tone to the end-user. If this type of mistake can be corrected at a later time, perhaps during an editing period, or is not crucial, then this type of feedback may be sufficient. If this type of error will cause some type of problem to either the data or to the system, then the designer should use a different type of feedback. It is important to measure the complex levels of responses needed to handle the speed and accuracy constraints of the application. This should be designed in the initial considerations of the system to determine what factors will be important to the end-user.

Summary of feedback

No matter what types of feedback are chosen, it is important to remember that some "handshaking" must be supplied to the user in any application. The types of feedback can be determined by the dialogue designer or selected by the end-user. When the end-user is comfortable with the accuracy of the system, he/she may wish to increase the speed of input by changing the feedback from an echo of his/her words to a simple tone or LED. Whatever the choice may be, it is crucial to have proper feedback on all levels of communication with the system.

Chapter IV - Voice I/O Technology

Justification for the use of voice I/O

There are many benefits to using a system that allows for voice communication with a computer. Its benefit is most realized in situations where the end-user is engaged in multiple activities. An example of this might be a parts inspector who can view and manipulate a part and still enter his/her findings via voice to the computer. He/she can also listen to instructions from the computer using a voice output system and not have to look up from his/her work to read the next instruction. When using other methods of interfacing to the computer, it is virtually impossible to engage in dual activities [CHAP77]. Certain handicapped groups also can benefit from this type of interface [GULI84] [KING84]. There are reading devices available for the visually impaired and devices used to help hearing-impaired in speech training and word recognition.

Categorizing speech input and output

Voice communication with a computer can be broken down into six categories. These categories may overlap but for the purpose of dialogue implementation and current technological limitations they have been divided as such. The first is voice or speech input using continuous speech recognition. This is where there are no limitations placed on the end-user as to how to speak to the system. This would be very similar to the way that humans communicate

with each other. The second category is isolated utterance speech input. In this case a restriction is placed on the end-user to artificially pause between each word as he/she speaks. The third category is connected speech recognition where some of the features of isolated and continuous are joined together. The fourth category is called speaker recognition. This is where the recognition of the speaker is more important than what was uttered.

The last two categories concentrate on output from the computer to the end-user. The fifth category is speech synthesis, which is the process of converting computer formatted data to a speech sounding audio signal. Some of the products that use this today can represent speech very accurately. The sixth category is stored speech. Stored speech can be used as both an input and output method. Speech can be transformed from an analog signal to a digital signal and placed into computer memory for playback at a later time by the reverse process. This method is very similar to the way a tape recorder works but without any moving parts.

Any combination of the above input and output methods may be used as an interface with a computer system. Depending on the system and the application, concurrent voice input and voice output might be beneficial to the end-user.

A more detailed description of voice I/O technologies follows.

Voice Input

The concept of speech recognition from past to present

Speech recognition as a concept has been around for quite a while. The first mention of speech recognition was in the play R.U.R. by Karel Capek [CAPEZ1] [WIEG84]. The play, originally performed in 1921, begins with the dictation of a letter to a robot capable of voice recognition. Ideas from this play and other science fiction stories planted the seed for the development of voice communications between man and machine. Unfortunately, the technology that has been developed today does not match the expectations of the end-users who have been exposed to perfect speaker independent continuous speech recognition devices in science fiction stories. This promotes frustrated end-users who expect more from the technology than the technology is ready to offer [JOOS86].

Isolated utterance

Voice input can be broken down into four main categories. The first is called isolated utterance and is used for word recognition by the computer. The end-user of a voice recognition system that uses isolated utterance must place artifical pauses between each word as he/she speaks. This places a constraint on the end-user's rate of entry. The end-user must say each word separately from the others and allow a long enough pause between each word for the computer to successfully identify it. This method is used

because of the difficulty in properly identifying the location where one word stops and another begins.

Connected speech

The second category is connected speech [RUBI84]. This method removes the necessity of artificial pauses between each word from the isolated utterance method but normally limits the vocabulary that can be used by the end-user. The vocabulary is usually restricted to numbers (digits) when connected speech is used. Only two of the numbers sound similar to each other, five and nine, making it easier to recognize where one number ends and another begins. If the input to a system requires only numerical input on a single digit basis, then connected speech is better to use than isolated utterance. This method is faster than isolated utterances since the pauses between each word are gone, but the end-user is restricted to single digits only.

Continuous speech

The third category for speech recognition is continuous speech. This is where the end-user may speak to the system in a comfortable and "normal" manner. No restrictions are placed on the end-user regarding speed of entry. This is also the goal of speaker recognition systems and the method that most end-users prefer. This is a very difficult process since people blend words together. Systems confuse similar sounding phrases such as "I scream" and "ice cream."

There is also a need for artifical intelligence to analyze what is trying to be said so that it is not misinterpreted. An example of a sentence that when spoken out loud seems to have a new meaning is "It isn't easy to wreck a nice beach" [DREY86] (it isn't easy to recognize speech).

Speaker recognition

The fourth category is speaker recognition. This is a method of recognizing the actual end-user that is using the system and not what has been uttered. Usually the features of the end-user's voice will be analyzed for anything that will distinguish him/her from another speaker. Some parts of the word recognition areas may be used in speaker recognition to identify special passwords used by the end-user. One main advantage in speaker recognition is that no two speakers talk exactly the same. The voice is as unique as fingerprints. One main disadvantage is that no person can duplicate exactly any word that he/she has uttered. Some people have a high degree of variability in their speech; therefore making it difficult when providing training and recognition [GARD85]. Colds, fatigue, and stress can also have an affect on recognition accuracy [RUBI84] [PAUL86].

How does speech input work?

In the four methods for speech input two word identification methods are used. Both of these methods are

used independently but may be joined to improve recognition quality. The first method is called template matching. This is where the word is matched with a predefined template that represents the voice pattern of the word [GOLD85] [SCHO85]. The actual process is done by first building a "vocabulary" of templates of all of the words that will be recognized. The word that is uttered by the end-user is then compared with each of the templates and a percentage of an exact match is computed.

Each word of the vocabulary is analyzed for a set of features. These features might include pitch, tone, amplitude, and other speech features and their values are placed into a template for the word to be matched. When the end-user utters a word to be recognized, its feature values are extracted and placed in a feature vector representation. This representation is then matched statistically against all of the templates in the vocabulary for a match. Features of a word are more robust to the subtle variations in the way a given person says the same word on different occasions [WIEG84].

The utterance is matched to the templates and ranked by percentage of the match that occurred. If it is an exact match then it is assigned 100%. The percentages are then placed in order and the template that matches with the highest ranking percentage, closest to the original word uttered, is chosen as the word that was uttered. Thresholds are set to determine an acceptable match. An example of a

threshold would be if the match had more than 80% of the features matching then that would be acceptable to the end-user as an adequate match. A threshold set too high might have many rejections where there are no exactly matching templates.

There are cases where all of the closest matching templates are below the threshold that has been set. In these cases a best match could be chosen as the match that comes the closest. Some applications may set the threshold to accept almost everything and then make the decision of the match by the best match or closest match. It might even accept the top three choices and let the end-user decide which was the correct word in cases of a close call.

Phoneme recognition of speech

The second method for word recognition is phoneme recognition. A phoneme is described by Webster's dictionary as "a member of the set of the smallest units of speech that serve to distinguish one utterance from another in a language or dialect." A phoneme recognizer breaks apart the word into its phonetic parts [MANG86]. It then looks up the resulting phoneme string in a phonetic dictionary that translates the word to its correct spelling. This method is based on the theory that any phoneme, when pronounced, is basically the same regardless of the speaker [MANG86]. That is to say a "b" is a "b" no matter who says it. It also allows for an almost infinite vocabulary list since there

are no restrictions to the vocabulary used as in the previous two methods. The only vocabulary restriction is the dictionary list which is used to translate the phonetic representation of the word to its correctly spelled counterpart.

End-user training

In all of the above three methods there must be some kind of system for training the voice patterns so that the computer will have something on which to base its pattern matching. One type of training is to simply read a list of vocabulary words into the system [BULK86] [RUBI84]. The list of words might be a phonetic word list. This list includes words that represent all of the spoken charactersitics of the end-user's natural language. The words are then broken down into phonemes that are separated and used later for comparison. Therefore, any time the end-user reads the list and pronounces a sound, the computer will store that sound for later comparison with similar sounds spoken by the end-user. If this is done several times per word, an average of the voice patterns is calculated and used as the template for future comparisons. This process builds an acoustic-phonetic model of the speaker's voice [GOLD85]. Retraining any of these words, if poor recognition occurs, is accomplished by simply going back into training mode and repeating the misinterpreted word a few more times to increase the accuracy of the

template. A problem that might occur in this type of training is that errors accidently entered into the system are taken. This can lead to unnecessary stress on the end-user when training the system [FINK86]. There are systems which constantly update the template when a good match occurs while the system is being used.

Once an end-user has been trained, the template from the training can be used for recognition. It has been shown that even after a short period of time the voice pattern remains relatively stable [GARD85]. This same study has shown that merely having an experience using voice recognition systems will improve recognition performance [GARD85].

The problem of end-user dependent recognition

When an end-user trains a voice entry system, the system typically will respond only to his/her voice. All of the templates that are stored in the vocabulary of the system are set up to match the end-user that trained it. This is not to say that a person with similar voice qualities could not be recognized using someone else's template. Chances are that one person could use the template of another person and be recognized but not with the same amount of accuracy as the original speaker. This is called a speaker dependent system. That means that the system has been trained to recognize only one end-user at a time and that his/her template must be installed in order to

have the highest accuracy in recognition of the speaker.

Speaker independence

If a group of end-users need to use the system simultaneously or interspersed with other end-users there are several ways of accomplishing this. One method is to have several stored templates in the system at the same time. The end-users simply have to identify themselves to the system so that it can access the correct set of templates. Another method is to have the end-users alternate in the training of the system so that the template average is based on multiple end-users and not only one [BULK86]. This method generally may not yield as high an accuracy rating as a single speaker system [RUBI84], but it will lend a degree of speaker independence to the system. Some systems are sold with a speaker independent option, but the drawback is a restricted vocabulary consisting of digits 0 through 9 and "yes" and "no" [VN5185].

Multiple training and environmental impact on training

Many times it is necessary to have the end-user train the system multiple times. When the system is being tested, it may be in an experimental area without natural background noises. When the end-user then uses the system in its natural environment, recognition may not be as high. This is because the training was done in a noise free environment and the actual use of the system is in a noisy environment.

The noise from the environment should be included during the training of the end-user on the system so that it may be included in the template of the word. This will increase the accuracy of the recognition since the same "noise" pattern will be entered with the word to be recognized. The "noise" is actually a combination of both the spoken word uttered by the end-user and the background sounds that exist around the end-users' environment.

The problem of background noise

There are other methods to resolve the background noise problem. One is to have a noise cancelling microphone. This is a microphone with built-in filters to cancel out the sound frequencies that are not in the speaker's voice range. Although they cannot cancel out all of the background noise they can greatly reduce the amount of extra sounds that might affect the results of the template matching. The noise cancelling may also be done in software using signal processing algorithms on the incoming voice signal to remove the unwanted noise. Software signal processing is slower than the hardware when implemented.

Other conditions that cause poor recognition

Besides being a training problem, using a voice entry system in a noisy environment may also cause recognition problems [CHAP65]. It also has been found that people who are under great stress while using the voice entry devices

have poorer recognition due to changes in their voice [RUBI84]. Pilots who give voice commands have been known to alter their voices under stress which lowers their successful recognition rate [PAUL86]. Although it is impossible to train workers of this nature under complete battlefield conditions, it is one factor to be considered when implementing a voice entry system in a high stress environment.

Where to implement a voice entry system

Probably the best environment in which to implement a voice entry system is where the end-user has both his/her hands and eyes busy on the task. Inspectors of equipment must keep their eyes on their work and manipulate the actual devices they are inspecting with both hands. This makes it difficult to manipulate any input device requiring viewing or tactile manipulation. A poorly implemented user interface in this situation would slow down the workers by having them constantly go back and forth from the task to the input mechanism. If voice entry is used they can have their full attention on their task and still enter the necessary data or commands into the system. Speech recognition systems should not be used to replace an already existing system but the system should be designed to include speech if appropriate [SCHO85].

Summary

In summary, there are four main categories of voice input or speech recognition. The first is isolated utterances, which forces the end-user to speak to the system with pauses between each word. This limits the speed of the end-user. The second category is connected speech, when the end-user is allowed to enter a fixed vocabulary of words without the artificial hesitation of isolated utterances. It does put a severe vocabulary restriction on the end-user, usually limiting it to only digits. The third category is continuous speech, where the end-user may speak at a normal pace to the computer with no hesitation. This is also the preferred method of communication. The fourth category is speaker recognition. This category varies from the others in that the identification is not placed on the word uttered but on the end-user that uttered it. The techniques used for recognition are the same as previously mentioned except the emphasis is to identify the speaker and not the word spoken.

There are two methods used to figure out which word was said by the end-user. The first is template matching. This is where pre-defined templates have been set up or "trained" by the end-user to be matched against anything that may be said in the future. This is done by a statistical analysis of features extracted for the vocabulary. The second method is done by phoneme recognition of the parts of the word. This is done by breaking down the word that was entered into

its basic phonetic parts to be analyzed. The phonetic representation is then mapped through a dictionary to a correct spelling. In all of these methods there must be some kind of system training so that the system will have a basis for later comparisons. This training is usually accomplished by multiple repetitions of vocabulary lists by the end-user. The training should take into account the background noise normally found in the working environment. This will have an effect on the recognition qualities when the system is actually used. All of these factors and restrictions will have a bearing on how the end-user perceives the system and how successful the system will be.

Voice Output

Voice output is the presentation of computerized data to an end-user using a method that is perceived to be voice. In this way the end-user hears the computer system "speak." There are two methods that will be discussed on how the computer is able to present material to the end-user in this manner. The first method is voice or speech synthesis, and the second is stored speech.

Voice synthesizers

There are problems in using text-to-speech voice synthesizers. In the English language written words are not phonetic. This fact is the most common reason that a simple text-to-speech system will mispronounce words. One solution to this is to store a list of commonly used words and their corresponding phonetic representation. When the word that is to be pronounced is on the list of words, the word is replaced with the phonetic representation of the word, which will be correctly spoken by the speech synthesizer. The computer cannot store all potential words since all proper names, for example, could never be stored in the computer. Also there are words that can be pronounced different ways, depending on the context.

One common method used in determining the correct pronunciation of words is by heuristic rules. The computer system can have a list of pronunciation rules with which it can decide the correct way of translating the word to

speech. For example, if the last 4 letters of a word were 'tion', then it should be changed to 'shun'. Although this might not apply to all words it will cover the most common words and will lead to fewer mispronounced words. Products like DECtalk have more than 500 rules and tools to help pronounce a word correctly [DECT85].

Stored speech

Stored speech or digitized voice is the second method for generating voice output from a computer. It is actually unfair to say that the computer creates the voice output in this method because the computer only acts as a voice recording machine, allowing the end-user to temporarily place a copy of his/her voice patterns into memory for later recall. Using this method, the end-user may store and playback his/her voice just like using a taperecorder. Using this technology the voice is digitized or converted into a digital representation so that the computer can manipulate and store it. This method can also be described as voice input, but since the computer treats the incoming signal only as data it is more useful to the end-user as an output mechanism.

Stored speech has many uses. It is commonly used by the telephone company for directory assistance by having the sentence "The number is..." read before the number that is given by the computer [DECT85]. It is also used in financial reporting systems by reading the sentence "Your

bank balance as of yesterday was..." before the balance is given [DECT85].

Stored speech is also used in telephone answering machines to receive incoming calls and to relay information to different end-users. Similar to a normal answering machine, the computer can store the voice of a telephone caller for later playback by the owner of the system. It can also store messages that are accesible to a caller by using a code typed into the telephone with a touch tone key pad. Pre-recorded messages can be played back depending on what code was entered.

Recognition of synthesized speech

There is a concern, when using speech output systems, that the end-user will not be able to understand what has been said. Studies have shown that with experience, practice, and training, the end-users can improve their recognition capabilities of synthetic speech [NUSB83]. They have also shown that there is a perception difference between natural speech and synthesized speech [PISO85]. One can blame these factors, which must be compensated for in the dialogue, when responses from the system are not recognized by the end-user. With proper training on the system, the end-user's recognition can be improved.

Chapter V

Design Considerations for Building a Voice UIMS

There are several things to consider when building an interface from the end-user to an application. They include everything from the layers of translation to the testing of the end-user. It is important to make sure that the end-user will be satisfied with the end product. It is also important that the dialogue designer will be able to use the tools provided for him/her in a convenient manner. All of the parts to the interface development life cycle must cooperate with each other in order to produce an acceptable working interface.

When developing a UIMS, it is important to realize that the "end-user" of the UIMS is the dialogue designer. It is the dialogue designer that must be satisfied using the UIMS tools. The internal workings of the tools can be designed and implemented by a systems designer. However, the interface for the UIMS tools should be designed by a dialogue designer. It seems recursive that the dialogue designer should design something for himself/herself, but who better would know his/her needs.

The following chapters include a discussion of what is necessary when designing and implementing a UIMS for a voice I/O system. The topics that are discussed are: human factors issues, the language that the dialogue designer will be using to communicate with the UIMS, and a detail of the tools that the dialogue designer would need to build the

dialogue. All of these components are necessary to understand and implement the Voice UIMS.

"User-Friendly" Interface

It is not sufficient to know the technology of voice I/O and programming to create an application interface. There are many concepts, related to human factors engineering, that must be considered and studied before implementing a system. A closer examination of some of these topics are presented here.

The term "user-friendly" is an over-used descriptor in today's world. The public is being told that any item with "user-friendly" markings will be a pleasure to use and will require minimum effort on their part to learn. This makes it very attractive to the naive end-user who knows nothing about computer systems but who still wants to have his job made easier through the use of a computer. Unfortunately this label is sometimes used just to sell the item and is not what it was advertized to be.

End-user frustration

Even with the development of tactile, visual, and audio interfaces, the end-user of a computer systems may still face frustrations and misunderstandings of how to interact with his/her application. The operation of these devices is sometimes difficult or inappropriate to use and may tend to confuse the end-user instead of helping him/her.

Some end-users, after they overcome their initial fear and frustrations of using the system, become completely disappointed and annoyed with the system and give up on it

altogether. Others want to use the system. It shows that the system was designed and implemented correctly so that the end-user is satisfied. Hence, the goal of the development process is met.

There are many features that allow the end-user to feel more comfortable when interacting with an application. Having an on-line help system often makes the job easier. When an end-user runs into an unknown area of the system or forgets the next step, the system should be able to guide the end-user back on the right path. If the end-user makes an error, it should be presented in a non-threatening manner, and suggestions should be offered on how to alleviate the dilemma. Controls and feedback mechanisms should be set up in a comfortable and easy-to-use manner. If the end-user must operate the system all day, it should be designed so that he/she is subjected to the least amount of mental and physical strain as possible.

A poorly designed human-computer interface will result in significant operational costs of computing systems due to many effects on the end-user, including extensive training requirements, high error rates, low productivity, and the poor emotional attitude of their end-users toward computers.

A human-computer interface should have the following properties [JACO84]:

* It should be easy to understand. It should not present material to the end-user that is not directly related to the task.

* It should take less effort to produce than to write the software that would implement it.

* It should be easy to check for consistency.

* The specification technique should be powerful enough to express nontrivial system behavior with a minimum of complexity.

* It should separate what the system does (function) from how it does it (implementation). The technique should make it possible to describe the behavior of a user interface, without constraining the way in which it will be implemented.

* It should be possible to construct a prototype of the system directly from the specification of the user interface.

A study of the end-user

Studying the end-user is as important as understanding the task being performed. One can either observe the end-user or become the end-user himself/herself. Participation in the activities is a strong investment into the working knowledge of the end-user; however, this is sometimes difficult due to the specialized skills an end-user might use in order to accomplish his tasks. When this is the case, questions and observations are helpful. By observing the end-user, we can see what tasks he/she is performing and how he/she is going about performing those tasks. It is advisable to use media, such as video tape

cameras and tape recorders, for later playback of the tasks for in depth study. There are special equipment used by ergonomists for the study and measurement of time and performance of the end-user. An example of this might include a video recording of the end-user performing the task with a time mark in tenths or hundreds of a second displayed on the screen.

Decisions should be made to separate the parts of the task to be handled by the computer and the parts to be handled by the end-user. Limitations have to be placed on these task assignment, especially when they reach past the bounds of either the technology of computing or the physical and mental limitations of the end-user. An expert from both disciplines, a systems analyst and a user interface designer, should be represented so that the system can be properly divided into the parts that will handle the interface and the parts that will handle the application.

There should be enough information to resolve certain questions: how long would it take to learn to operate the system?, how fast does the system operate?, what are the typical mistakes made by the end-user?, how often do these mistakes occur?, and how satisfied is the end-user with the task?

An in-depth study should be done to determine the characteristics of the end-user so that the most appropriate interface may be designed for him/her. Certain demographic information should be found out about the user. These

should include: what kind of person is the end-user?, what kind of a work environment does he/she have?, is the end-user willing to change the methods he/she is using presently?, is there a lot of training necessary to perform this task?, what is the educational background of the end-user?, and what are the attitudes of the end-user towards computers? These questions and others should be thought out before attempting to design a system for a specific end-user. It is very important that a user interface designer know what type of end-user he/she is dealing with. There will be a definite payoff when the user interface is implemented if these questions are considered and the system is implemented correctly according to the end-user's needs.

Speech as a natural interface

It has been thought that since humans communicate mainly by speech, it should also be a natural manner in which to communicate with a computer. It has also been shown that this is not necessarily true [VANP85]. Although it is natural to talk to other humans, it is not natural to talk to inanimate objects and have them respond without the aid of another human.

The dialogue designer

The creation of a user interface requires the skills of a specialized person. This person is an expert in end-user

communication styles. The term used for this person in this thesis is a "dialogue designer."

The dialogue designer has a background in both computer systems and human factors. He/she can understand the needs of an end-user and work with the design team to help create a convenient and useful "user-friendly" system. The dialogue designer must be aware of both the limitations of the computer system and the needs of the end-user. He/she should be a part of the sytem from conception to make sure that the interface will be an intergral part of the design and not something that is added on at the last stage.

The dialogue designer does not act as a liaison between the system designer and the end-user but works as a team with the system designer and may help with the communications between the team and the end-user. In theory a dialogue designer should have good communication skills and good design skills. These skills will lead to a better design with less end-user frustrations.

The dialogue designer needs tools to perform his/her work. These tools should aid in the design and promote good design practices. They should also be practical for the job of designing interfaces. The tools should be packaged in a way to give the designer access to any of them conveniently and easily so that the process of design is not distracted by the method of getting it done. This is where a well designed UIMS is useful.

It is very important to keep a happy medium by

understanding the needs of the end-users and the limitations of the application. This is the job of the dialogue designer. In the development cycle it is important for the interface designer to know what the end-user is expecting of the system and what the system can do. He/she can explain to the end-user if certain functions are impossible for the application to accomplish. He/she can also get information for the system designer which indicates the end-user's expectations of the system and what functions should be implemented.

Language Specifications

In order to specify to the computer what actions are to take place during a dialogue, a formal language may be set up.

The interface language must have certain desirable features to aid in the creation of a UIMS. It must have the ability to check for consistency [BETT87]. It is important throughout the design to have a consistent interface so that the user can go from one part to another in the interface and be able to transfer the skills of operating one area to another.

The interface language should also have the flexibility to show the state of the dialogue. This would allow the creator of a dialogue to identify certain paths the user would take when dealing with the interface. This could also help the designer in following all paths of the dialogue when debugging or testing the dialogue.

The third feature would be the readability of the language. The language should conform to a semi-natural language structure that could easily be understood without having to learn a new vocabulary. The designers and builders of the user interface do not have to be programmers and should not have to know the intricacies of programming to build the user interface. Using a form of natural language structure will increase the readability of the user interface for later modifications and maintenance.

A language should be simple to use and understand in

order to be part of a dialogue design tool. This "ease of use", should expedite the formation of a user interface dialogue since the language should not inhibit the production of the interface.

The naming conventions of the language should avoid putting restrictive limitations on the variables and structures used. This will allow the dialogue designer to create the interface more towards his/her own style. This lends to greater flexibility and should also carry over to the point of readability.

The dialogue creation language should be able to operate in two forms: interactive mode and compiled mode. The interactive mode will allow the user to test the dialogue while it is being designed and built. If speed and efficiency are vital, then the dialogue can be operated in compiled mode.

Design of Tools

There are several tools that a UIMS designer needs in order to build a UIMS. All of these components working together comprise the major working function of the interface to the dialogue designer.

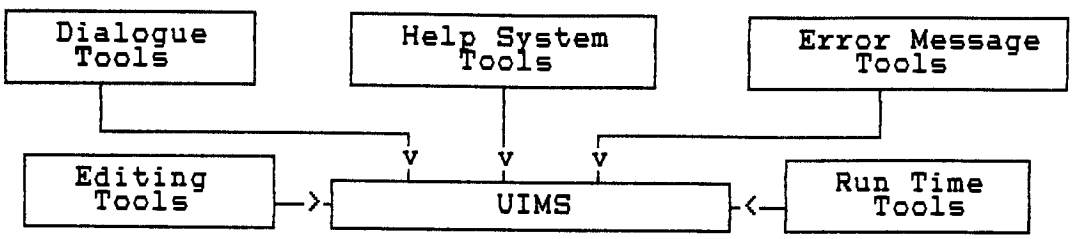


Figure 4. The components that support the UIMS.

Figure 4 is a simplified diagram of the system used to build a UIMS. This dialogue is built in stages where each part of the dialogue is created through interactive design and experimentation. This gives the dialogue designer the flexibility to change the design of a dialogue while the application is simultaneously going through its developmental changes. The tools in this system will guide the dialouge designer to develop a working dialogue with all of the important functions of the interface properly implemented. The following is a description of the design tools necessary to build a UIMS.

Editing Tools

The most visible part of the UIMS tools is the editor. It is the editor that guides the interface designer to implement a proper interface for the end-user. The editor handles the creation of the total interface dialogue and the maintenance of it (see Figure 5). The only part it does not handle is the actual running of the dialogue. The editor must be flexible enough that it does not frustrate the dialogue designer, yet structured enough that it will guide him/her to the completion of a dialogue.

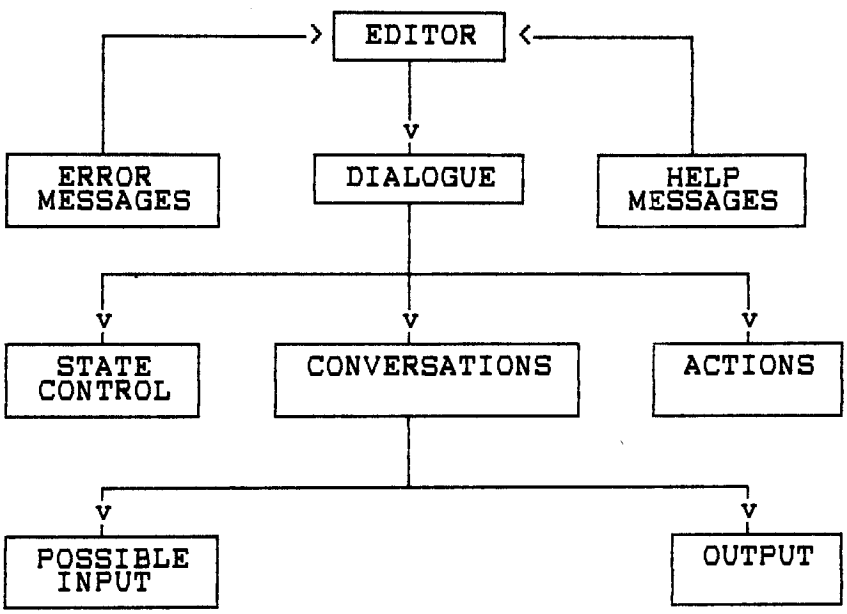


Figure 5. The data flow controlled by the editor.

Custom formatted editor

The editor should be set up in such a way that it is configurable by the dialogue designer. Many programming

editor functions can be selected by the dialogue designer in the fashion that he/she prefers. The assignment of function keys is an example of flexibility in an editor. The dialogue designer can choose which functions keys represent which functions. The function 'delete' can be configured to: the first function key, <CTRL> D, the actual word 'delete', or many other combinations and selections. The dialogue designer should have the ability to choose which messages he/she wants to see and in what format they will appear. Certain levels of messages can be selected for display or be ignored. An example of this could be the ability to turn off all warning messages. This could prevent messages, that would appear while creating the dialogue, from bothering the designer. He/she could also re-word the messages to reflect his/her personal style.

Error-free dialogues

The editing system is in constant contact with the error checking system to maintain data integrity. Every time the dialogue designer enters a command or data the input is sent to the error checking system for evaluation. The dialogue designer is warned that what has just been previously entered is not correct or is a possible error. This feature can be turned off at the request of the dialogue designer for faster data entry, but it is not recommended. The error checking is done on both a lexical and syntactical level. This refers to the errors that are

due to the misuse of the editor and also to the improper creation of the dialogue. Wherever possible the dialogue is checked for consistency.

Editing features

The editor should have all of the basic features of most editors with multiple ways of manipulating the dialogue of the editor. The editor should be able to copy, move, insert, and delete parts of the dialogue as it is being written. There should be a feature to allow the dialogue designer to design a template for each part of the dialogue and have the editing system prompt him/her for each appropriate section with that template. This feature would also add to the consistency of the final dialogue. The editor should also allow the dialogue designer to view any or all of the dialogue for reference at any time. This should help the dialogue designer develop the flow of the dialogue. While the dialogue designer is viewing the dialogue he/she should be able to continue editing without having to go back and forth between viewing, editing or any other modes [TESL81].

Entire dialogues may be used as templates by specifying the name of an existing dialogue to be used as a template for the creation of a new dialogue. This feature is useful when creating a dialogue which is similar to an already existing dialogue. It can also be used to test multiple dialogues for the same application without having to modify

the dialogue between tests.

Presentation

At any time the dialogue designer can see all of the possibilities that can be entered for each selection. As the dialogue designer builds the dialogue, the parts that have already been built can be viewed as a guide for the construction of the rest of the dialogue. This is useful since it is difficult to remember the many different names already assigned in the dialogue.

Having different methods of displaying the design greatly enhances the creative ability and judgement of the designer. Since there are more perspectives on the dialogue it is possible to see the dialogue from more than simply a linear view point. An example would be the ability to view the design as a drawing of circles and arrows depicting the flow of information through the dialogue. When a dialogue is designed on paper it is sometimes represented in this manner. Having the ability to view the dialogue this way could help the interface designer follow its paths easier. Depending on the method the dialogue designer chooses to use to create the dialogue, the speed in which the dialogue is created may be greatly increased. Suitable hardware for this would be a graphical display screen with a pointing device for each location. As each location is built a location-label with a circle around it could be drawn on the screen. To and from each location would be arrows that show

the flow of the dialogue. This representation also helps the dialogue designer identify locations that have not been fully attached to the dialogue because of missing entry or exit points.

Since it is difficult to display large dialogues completely this way it is advisable to have two different modes of displaying the locations in the dialogue. The first method is a detailed diagram of one location only. This diagram could include a labeling of all paths. The second method is an overview of the whole dialogue. Since screen space could be a problem, parts of the dialogue may be viewed at a time. It is recommended that a "zoom" feature be implemented so that the dialogue designer may see as much or as little of the dialogue as necessary.

Dialogue Tools

The dialogue tools are the heart of the UIMS. It is the dialogue that controls the perception of the interface as well as flow of interaction with the application system. Since the dialogue plays a crucial role it must be set up to allow the most amount of flexibility to accommodate the end-user.

Transition Network

A Transition Network (TN) is one way of organizing the dialogue between an end-user and an application in the UIMS [JACO84] [GREE86]. The network itself is a simple pointer mechanism from one state in the dialogue to another [GREE86] [SALT83]. This is demonstrated by the following diagram (Figure 6):

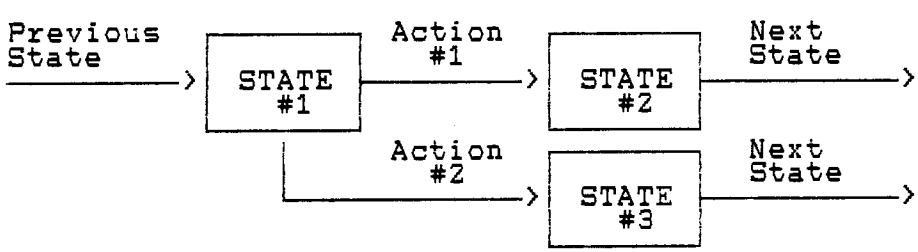


Figure 6. A representations of a Transition Network.

Depending on the action that the end-user chooses the TN will pass from one state to another and determine the next part of the dialogue. Before proceeding on to the next state, the input and variables of the dialogue are checked

to see if all dialogue-related criteria have been met.

Each state also has a set of actions to be performed by the UIMS. An action might be a null action, while the application is waiting for enough input to be built up to act upon it. It might be to call an application routine when all of the parameters have been collected.

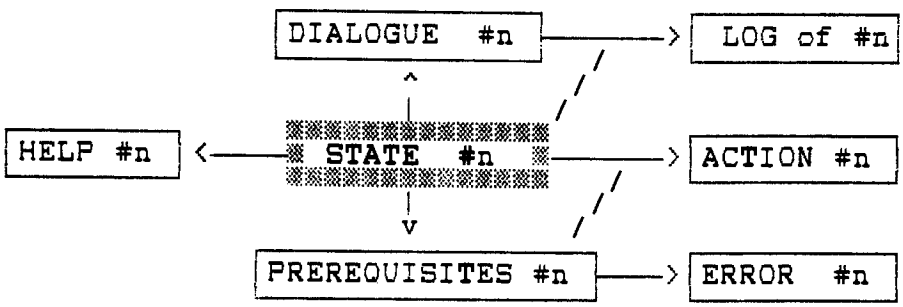


Figure 7. Relationships between the state and its components.

As shown in Figure 7 the components of a state are linked to the state, which in turn directs the flow of the dialogue. The state represents a resting area between the transition from one part of the dialogue to another. The actual mode of communication, voice or keyboard, to and from the end-user is irrelevant from the perspective of the state and its components. The messages that are received and sent out from the state are processed through another layer of the UIMS that handles the actual I/O. The state represents the core of the interaction between the application and the end-user.

The dialogue can be built into a TN making all

possible variations on the dialogue possible. Each state is divided up into actions that can either branch off into linear or recursive action. An example of a linear action is a series of application calls that are necessary to check and translate a specific input. A recursive action would be an action that is dependent on the result of another action. The reason that the term recursive is used is that the action that is dependent on the other, might be the same action deeper in the dialogue. This would make the process of parsing the dialogue a recursive process.

Location in the dialogue

It is important that the dialogue designer know where he/she is in the dialogue he/she is designing. To help the dialogue designer, there must be some indicator to show where in the dialogue he/she is. This can be accomplished by flag setting, labels, state indicators, or pre-assigned dialogue sections. The dialogue designer can assign a unique label to each part of the dialogue to maintain its structure. These labels will be used as a reference point to branch to/from other parts of the dialogue. The labels may be viewed at any time using the editor functions for viewing parts or all of the dialogue. This can be presented in either graphical or list format. Indicators for current location will be kept in the record keeping system.

Dialogue control flow

The dialogue is directly linked with the need for information from both the end-user and the application. When the request for information has been satisfied, the dialogue can be directed to take appropriate action. These actions could be the setting of variables, or involve calling an application routine or prompting the end-user for more information. These needs must be designed properly in the UIMS so that all locations in the dialogue may have separate command-response capabilities.

Dialogue actions

After each response by either the end-user or an application the UIMS must take some action. These actions are specified in the dialogue description. There are two basic types of actions that can be performed by the UIMS. After a response is validated the actions are performed. The first action that may occur is the assigning of a value to a variable. The second action is to transfer to another location in the dialgoue.

Dialogue variables

The variables of the dialogue system are in two forms. The first is used to match the types of the response from the end-user or application to a previously defined response. This is done through a list of words that can be compared throughout the dialogue. The word lists are set up

in a very similar manner to the list concept in LISP [ALLE79] where a list of words may contain the name of another list. The dialogue designer may list all possible input responses that are acceptable for certain parts of the dialogue. There is also available a null list so that the input may be ignored if this is necessary in the dialogue design. These are set at design time and may not be altered while running the dialogue.

The second form of variable in the dialogue is for assignments. This variable can be manipulated the same way programming variables are. They can be assigned values for storage of parameters. They may also be manipulated mathematically to serve as counters and flags. These variables are set up to provide as much flexibility in the dialogue design as possible so the dialogue designer does not have to venture outside of the UIMS for external support. This is what is known as an augmented transition network [GREE86] [SALT83]. The variables are directly linked to each state for manipulation through the dialogue. An example of this might be if the dialogue designer wishes to see how many times the end-user responded to a prompt. If it was greater than three times then the necessary information has been received and the dialogue can proceed to another location.

Data validity

Data validity is accomplished by having the input to

the UIMS match a list of valid terms. This is automatically done by the UIMS since the dialogue cannot proceed unless proper identification of what was entered has been established. The UIMS will pass on the dialogue a flag that something was entered but it has not been satisfactorily identified.

Timers

Very often the timing of a dialogue is crucial. After a certain amount of time has elapsed the dialogue may prompt for help or further assistance. If the item is not important, it can be skipped after a period of time and the dialogue can be transfered to another more important area. The dialogue design tools should be created to include a way of manipulating the dialogue based on the passage of time.

End-user interface

The end-user interface exists between the end-user and the heart of the dialogue tool. This interface takes the input from the end-user and translates it into a format which the UIMS can handle. In the case of voice input devices the analog to digital conversion is handled along with the conversion to text. The end-user interface also handles the translation and formatting of the output. In the case of voice output it is important to designate what style should be used as well as the actual text that is to be said. The parameters that may be set for voice output

should include: type of voice (high pitched, low pitched, etc.), speed of utterance (how rapidly the message is said), and inflection of voice (how natural sounding the voice will appear to be).

Application interface

The transfer of data to the application is handled by the application interface. This part is co-designed by the application designer. This design includes: the names of the applications, the number of parameters in each application, the order of the parameters, the types and format of the parameters (numeric, character, etc), the values that will be returned from the application, and any other details that could affect the transfer of information to and from the applications.

Help system tools

The help system guides the dialogue designer through the UIMS development and operation. Whenever the dialogue designer is unsure of what option to use or what type of entry should be made, the help system should be available from any point in the UIMS developmental cycle. All the dialogue designer has to do is inform the system that extra help is required to continue from a certain point.

The help system provides help to the dialogue designer on several levels. The first level of help is actually not part of the help system at all. This is the prompts that are used by the rest of the system to guide the dialogue designer through the creation process. If these prompts are not sufficient enough to be understood at any point in time he/she may request additional help from the system. The help that is first shown is a reminder of what is to be done. If the reminder is still not sufficient for the dialogue designer to continue then a more detailed explanation may be provided by requesting for more help.

The help system is set up on multiple levels. The first level is the reminder level. This is a quick reminder of what is to be performed. This might be a one-word or small sentence reminder to aid the process of development. If this is not sufficient, the next level of the help system includes the format of what is to be entered as well as an example for study. If this is still not adequate for the user the next level is a users' manual describing in detail

where he/she is in the process and what he/she needs to do to continue.

A new user to the system might request the system to prompt at the second level at all times until instructed otherwise. This will avoid the necessity of having to request for more help during the learning process.

The most basic part of the system for help should be the system itself. The prompts and requests to the user should be self-explanatory and guide the user unambiguously as possible.

The help system is based on where the user of the UIMS is in the system. If he/she is using the editing section then all editing commands should be explained based on the location in the editing system and the type of help requested. If he/she is in the record keeping system then an explanation of how to manipulate all the commands in that system should appear.

The architecture of the help system is quite simple. It is based on a table look-up of different explanations corresponding to the locations in the UIMS subsystems. For each action that the user of the UIMS might make there should be an explanation telling him/her how to make it. The help system should also be able to explain in further detail if the first explanation was insufficient.

Error Messages Tools

Error messages could be one of the most important aspect of human/machine interfaces [BROW84]. They give the essential feedback necessary to the user of the system when something is not as it should be. The presentation of these errors should be in an informing and non-confusing manner. The error messages should be used to guide the dialogue designer through the workings of the system without frustrating him/her. They should be presented in a pleasing manner and above all should be correct [BROW84].

The error message should be informative and provide a helpful solution to the problem [SHNE86]. It should make available "hints" to the next selection item for the dialogue designer. If the process that lead to the error must be repeated then the error message should state that is the case. If the correct term need only be entered again to continue then the error message should present that information only, and in a non-threatening manner.

Error messages will guide the problem solving techniques of the dialogue designer. Erroneous or misleading information could hinder the eventual solution. If the error message is cryptic it will not be effective because after the dialogue designer has read it he/she will still be in the same situation of not knowing what has gone wrong. A wrong "guess" of the type of error by the system could also lead to further confusion with the UIMS.

Some errors can be prevented if the dialogue designer

is restricted to only certain input. One example is a selection from a menu. The dialogue designer is presented with a group of choices to select from. Any variation from those choices will result in a reminder that only those choices that are presented on the menu are valid choices. The menu selection works the same way as it does at a restaurant. Only those displayed on the menu are valid choices to order. Anything else will not be served.

Another method of error prevention is by having templates of the input presented and only patterns which match these "correct" templates will be accepted. This is where any input not matching the valid input will be immediately rejected and a prompt for the input will re-appear. This can only be used in very limited circumstances such as tutorials where there is only one selection.

One major flaw in many systems is the lack of consistency of the system [NORM84]. If an error message format is decided upon for the system then consistency must be maintained. The dialogue designer will adapt to this method and will expect this to be the format for all error messages.

Too much similarity in commands could also lead to problems. When attempting to operate a seldom used command that is similar to another command the end-user might find himself/herself using the more frequently used command instead by habit. Commands with different functions should

make sure that there are no overlapping sequences in them to reduce confusion.

If multiple commands are being performed simultaneously, then all of them should be apparent to the end-user. Reminders of all currently active tasks should be available to the end-user so that he/she does not forget about them.

There are many different types of errors that must be handled by the system. The two basic types are errors that are caused by the end-user misunderstanding the system and the errors caused by the system due to a poor design of the error handling routines. All error messages should be presented in a consistent and informative method to insure complete cooperation from the end-user. The error handling part of the system should be designed from the begining with the rest of the system development.

Run Time Tools

Application interface

The UIMS is constructed in layers to relieve some of the burden of translation from the end-user to the application as shown in Figure 8. The layer that exists between the dialogue system and the application is the application interface. The application interface handles the calls to applications from the dialogue system.

The application designer must work closely with the dialogue designer to determine the basic needs for communication between the application and the end-user. Although they are separate entities, they are still part of the whole system. They must also design an interface standard that will be used to translate the information going to and from the application.

The dialogue system gathers all of the paramters that have been generated by the end-user and the dialogue. It then organizes the parameters in the proper order for the application. From the perspective of the UIMS this is a prompt to the application for information to determine the next action to take in the dialogue. The returning information from the application will be viewed by the UIMS basically the same as the input from the end-user.

The application interface then translates the parameters from the format in the dialogue system to the format expected by the application. It seems unreasonable for the application to have to translate information from

the UIMS every time it is sent. The parameters should be in a format that can be used by the application. The same holds true for the information that is returned by the application. It should be translated back into a format that the UIMS can use without extra manipulation.

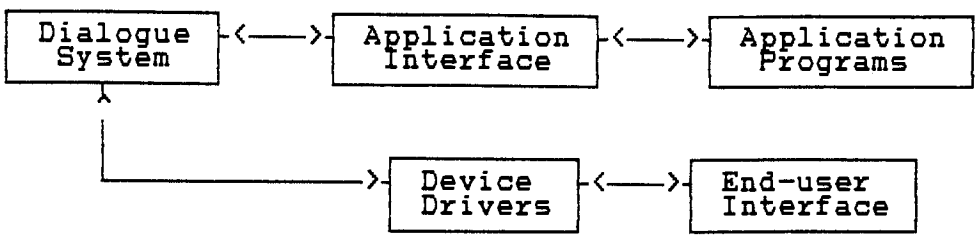


Figure 8. Dialogue system interface to end-user and application.

What are record keeping tools

Record keeping tools are a set of tools that can keep track of how the end-user is interacting with the system dialogue. This information can be very useful to the dialogue designer. It allows the dialogue designer to know what parts of the dialogue are being used the most and what parts are not being used at all. It also allows for statistical analysis of the dialogue so that optimizations can be made for a smoother more efficient dialogue.

These tools keep track of all possible paths and choices that the end-user may take. From these choices the record keeping system will record, at run time, what choices the end-user made, what order he/she made them in, and how many times the choices were made. From this information the dialogue designer can request a description of how the

end-user is using the system which can be used for data analysis purposes.

Components of record keeping tools

One of the record keeping tools is a statistical analysis tool that allows the dialogue designer to view the dialogue numerically. This includes histograms, bar charts and numerical representations of the dialogue. When the end-user is working with the system all commands from the end-user and paths through the dialogue can be optionally recorded for later analysis.

Data logging

The data logging section of the record keeping system allows the dialogue designer to select what parts of the dialogue he/she wants to concentrate on for data collection. During run time the events that occur are logged for analysis later by the dialogue designer. This allows the dialogue designer to request, for example, all application routines that were invoked, how many times, and in what order. This would give the dialogue designer some insight into the relationship between the dialogue and the application routines. He/she can also determine which part of the dialogue is being most heavily used and which areas of the dialogue are being ignored.

Data presentation

Data presentation takes the data that has already been collected and possibly transformed by statistical analysis and formats it in a presentable way to the dialogue designer (see Figure 9). This is an easier way to view the data than to merely have the data printed out in its raw format.

Statistical analysis

The data that is collected by the data logging part of the record keeping system can be analyzed statistically without leaving the UIMS design system. In the record keeping system the dialogue designer can selectively choose different statistical methods for grouping and counting the data to elucidate what is occurring in the dialogue. In the event that the dialogue designer wishes to view this data using another statistical analysis package outside of the UIMS tools, then the data should be stored in a manner that would make it easy to do so. The statistical analysis tool is included in the UIMS tools for convenience and availability.

Event flagging

The event flagging section of the record keeping system allows the dialogue designer to be alerted when a given set of events occur. This is useful for locating events that should not occur in the dialogue or parts of the dialogue that need special attention or further explanations (see

Figure 9).

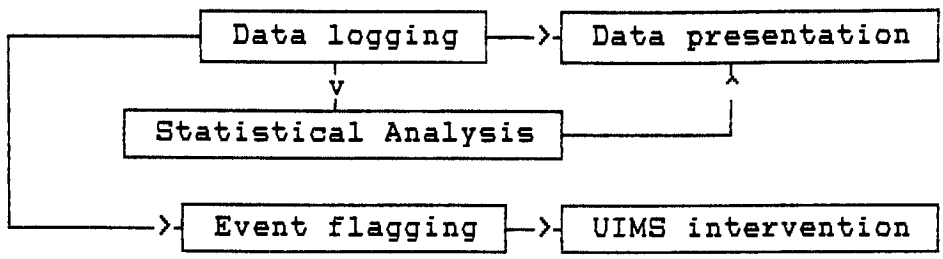


Figure 9. The Record Keeping System. The arrows represent the flow of information.

Chapter VI - **Implementing the Voice UIMS**

The following sections contain descriptions of the design that was implemented for the voice UIMS. Detailed specifications are described as well as the screen and data base layouts. This is a description of the middle layer of the voice UIMS, which includes the dialogue tools, editing tools, and some error checking.

In order for the dialogue designer to communicate with the voice UIMS, a descriptive language was developed. This language is based on the needs of developing a voice I/O dialogue. This system follows in intent, as closely as possible, the recommendations that previously were described throughout this thesis. Only the most important features were implemented due to time limitations.

Vocabulary and features of the voice UIMS

There are certain terms used in the system that were developed during the research and development process. These terms will be defined here so that the UIMS can be better understood.

Location labels

It was necessary to define a method of locating any part of the dialogue. To clarify this to the dialogue designer, a dialogue location label was developed. This was a character string of up to 50 characters that could be used to describe any location in the dialogue. The number of characters was decided arbitrarily but based on two factors. The first was that the name should not be too long that it becomes a burden when referred to. The second is that it allows enough characters to make a unique and distinct description of any location in the dialogue.

Commands

A method was also devised to group different words together that may be passed to an application for evaluation. These words entered by the end-user are normally in two forms. The first are commands and the second are data. The commands are words that have an action that an application can responded to. Data are words that are used for the collection of information to be used by the application at a later time.

The command is basically a way of grouping words to make it easier to compare them with the input from either an end-user or an application. For example, since red, blue, and yellow are all colors they could be grouped under a command by the name of 'color.' When the end-user utters one of these words they could be checked by the dialogue. In the dialogue the specific color is not crucial, but the fact that the uttered word was a color might be important. This information could then be passed to the application where the exact color might be relevant.

Built-in Commands

There are three built-in commands supplied by the system that can be used by the dialogue designer. The first is the #ANY command. This tells the system that any word that is uttered by the end-user is valid. In other words, the condition will always be evaluated to true if the end-user utters something. This is very useful to use in the dialogue design for two reasons. The first is determine if a response was given. The second purpose of the #ANY command is when it is used as a default command. After all of the other commands have been compared and there was no match with the word that was uttered by the end-user the default could be the #ANY command. This will signify that something was said but not what was expected by the system. This situation can then be handled by the dialogue.

The second of the built-in commands is the #NONE

command. This is the opposite of the #ANY command. It is useful when no response is necessary by the end-user for the dialogue to continue. The dialogue system can determine when the end-user does not say anything.

A more advanced feature related to the #NONE command is the #TIME command. This is a command that allows a certain amount of time to elapse before the dialogue resumes [SHNE86]. This is very helpful when the end-user does not know what to do next. After giving the end-user enough time to respond the dialogue system can prompt him/her further so that a proper response can be made. The dialogue designer should also be able to indicate a default time that could be used throughout the system unless indicated. It is recommended that testing be done on each dialogue to determine the amount of time the end-user needs at each location in the dialogue.

The #IF statement

The third built-in command is the #IF statement. This is used to test the value of a variable. It is the only command that does not test the input. The if statement has the following BNF format [TREM76]:

<#IFstatement> := #IF <variable> <comparison> <variable>

<variable> ::= <alpha> | <variable> <alpha> |
<variable> <digit>

<comparison> ::= = | <> | < | > | <= | >=

<alpha> ::= A|B|C|...|Z|blank | <digits>

<digits> ::= 0|1|2|3|4|5|6|7|8|9

Actions

When a command evaluates to be true then a corresponding action or group of actions occurs. Actions have two formats. An action can be a transfer from one location in the dialogue to another. This is done by a GOTO statement. The GOTO statement has the following format: **GOTO** [location label]. The second format an action might have is a variable assignment statment which is described below.

Variables

In the design of the dialogue it is often necessary to keep track of what has previously happened in the dialogue to make future decisions. It was for this reason that variables were implemented into the system. A variable name is represented by a character string of up to 30 characters. The only restriction in the system is that a variable must be preceeded with a dollar sign ($). After that any characters including spaces are allowed. One of the benefits of having this type of naming convention is that it allows the dialogue designer to have the flexibility of using the variable names as documentation. An example could be a variable named: "$ # of times color was entered". The variable can have two types of values, these are numeric and character. A character string in a variable may be up to 50

characters long. A numeric value in a variable may be up to three digits long. The reason that numeric values are only given three digits is because it was not expected that any loop counter in the dialogue need to count more that 1000 times. The original purpose of the variable was to keep track of the dialogue. In certain instances it is necessary to count how many times the end-user has gone through a certain part of the dialogue. This can be done by incrementing the numeric value in a variable.

Variable assignment statements

There are several formats for assigning a variable a value. If the response from the end-user is to be stored in a variable then the following format is used:

<variable assignement> := <variable> = <assignment>

<assignment> ::= <variable> | #RESPONSE | <string> |
<number> | <variable> <operation> <number> |
<variable> <operation> <variable>

<variable> ::= <alpha> | <variable> <alpha>

<number> ::= <digit> | <number> <digit>

<string> ::= " <alpha> " | " <string> <alpha> "

<alpha> ::= A|B|C|...|Z|blank | <digits>

<digits> ::= 0|1|2|3|4|5|6|7|8|9

<operation> ::= + | -

In the above formats the variables after the equal sign may be the same variable name as the first or different.

Future implementation of variables

Prompts with variables imbeded in them, may be implemented in the future, to give more flexibility and consistency to prompts. For example if there are several prompts that are similar, except for one word, that word may be assigned to a variable. Each time the prompt is given that variable may be changed to reflect the current need of the dialogue. The variable may also be used to reflect back to the end-user what has been said. If there is any ambiguity as to the correct word uttered by the user the phrase "Did you say..." [VANP85] may be used inserting the response in the place of the ellipsis. This might be in the format "Did you say <variable>?."

Hierarchical Order

The order in which commands are evaluated is important. To specify the sequence in which the commands are evaluated an order is assigned to them. This order specifies which commands are to be evaluated before others. This is crucial if certain events are to take place as the consequences of a command evaluating to be true. Certain variable tests might need to be made before the evaluation of one command causes the dialogue to branch to another location. This is a valid reason to determine the order of the actions.

Data Base

The UIMS was implemented in the format of a data base. This decision was based on the needs of the UIMS for the manipulation of the dialogue and availability of the software. The data base was designed as eight separate data bases linked together by key fields (see Figure 10). The data base was designed for minimizing space and maximizing relationships between the different parts of the dialogue.

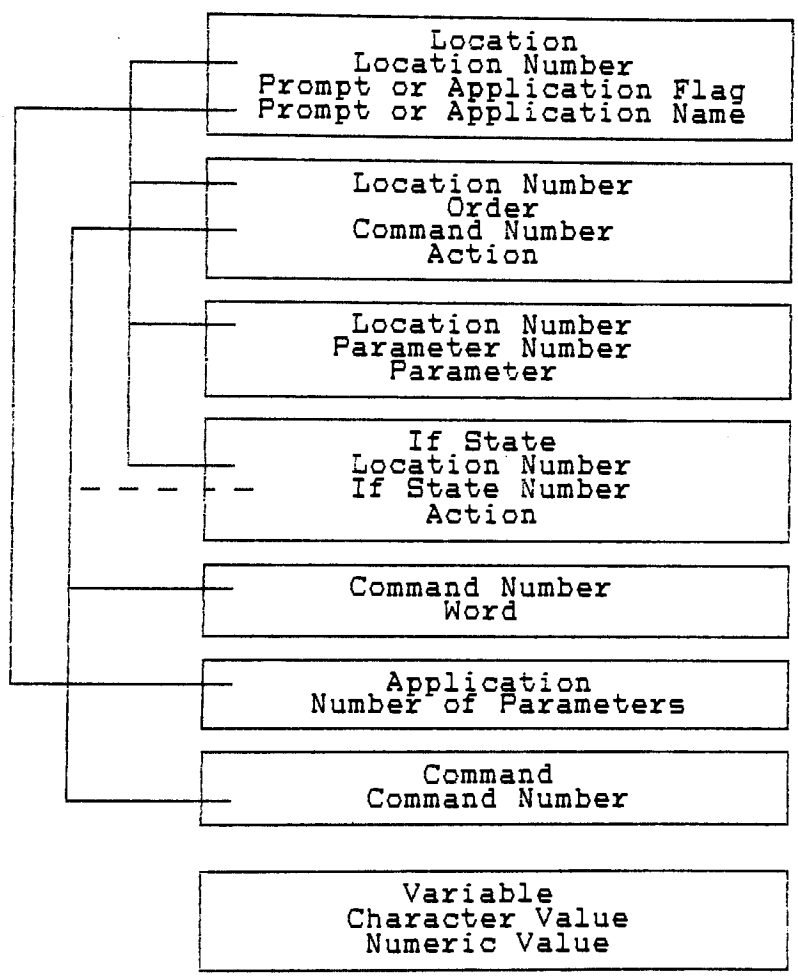


Figure 10. Relationships between data bases.

The structures used in the data base are as follows:

1. LOCATION -

Field Name	Type	Width
Location	Character	50
Loc_Number	Numeric	3
P_OR_A	Character	1
Prmpt_Appl	Character	50

The first data base to be designed was the location data base. This data base consists of four elements. The first is the name of the location (Location). In order to identify a place in the dialogue it was necessary to give the dialogue designer an area to label each location. It was decided that 50 characters were enough space to sufficiently elaborate where in the dialogue he/she was. In order to save on memory, the location label was replaced by a location number (Loc_Number), in all other data bases that reference a given location.

It was also necessary to define each location as one of three possible types. The first is a prompt to the end-user. The second is a call to an application program. The third is a transitory state where tests could be made affecting the dialogue which are not dependent on further input from either the end-user or the application. This was signified by the letters "P" or "A" in the prompt or application flag field (P_OR_A). If the location was only

used to check variables then a blank " " was inserted in the field.

The last field in the location data base is the prompt or application field (Prmpt_Appl). This is the field used to hold the prompt to the end-user or the name of the application to be called. Since both the prompt and the name of the application are alpha-numeric they were able to be placed into the same field. Limitations of the DBASE III system did not allow multiple structures occuring in the same data base location. It was also necessary to minimize on space for the system so any fields that could share functions were helpful.

2. ACTION -

Field Name	Type	Width
Loc_Number	Numeric	3
Order	Numeric	2
Com_Number	Numeric	3
Action	Character	50

The action data base was designed to store possible actions that may occur in any location. At each location it was possible to receive information from either the end-user or an application. This information was then compared with a command to identify what action should be taken next. If a comparison was done and the information passed to the UIMS matched what was expected at this location then all actions

that matched the expectation were done.

The location number (Loc_Number) identifies what location this action was related to. It matches the unique location number that is in the location data base.

The order field (Order) tells the dialogue system, which order to compare the commands from the end-user or application. This is important since an order for the transfer from one location to another, in relation to the setting of variables, must be established.

The command number (Com_Number) identifies what command should be matched against the input. Each command number is matched to a command in the command data base.

There are also three built-in commands that are used by the system to identify special cases. The first built-in command is the #ANY command. All built in commands are identified by a "#" preceeding it. The #ANY command signals the dialogue running routine that any input from either the end-user or the application is acceptable. This is very useful to transfer from one location to another without worrying about vocabulary. It is also helpful in keeping track of non-matching commands. In this situation none of the pre-defined vocabulary words have matched for a given location it can then be left to default with the #ANY command.

The second build-in command is the #NONE command, which is the opposite of the #ANY command. If no input was given by the end-user this command is activated. This is

useful if the user has not responded in any way.

The third built-in command is the #IF statement. This command is the only way of testing variables that have been set in the dialogue. Sometimes it is necessary to have certain actions occur based on the internal workings of the dialogue and not necessarily on the responses from external sources. An example of this would be the third time an end-user has gone through a particular part of the dialogue it might be necessary to direct the conversation to another part of the dialogue since no progress is being made in that particular location. This is done by setting and testing variables in the dialogue.

The action field (Action) is used to store the action to be taken if the command is evaluated to be true. In order for a command to be evaluated as true, the input from either the end-user or the application must match any of the words stored in the command data base corresponding to a command. The action can be either a GOTO which changes the location in the dialogue or an assignment statement which changes the value of a variable.

In the event that the command is an #IF statement then the action field stores the #IF statement number that corresponds to this location. Each #IF statement at a location is stored individually in the #IF statement data base. It is linked to the action data base through the location number (Loc_Number) and the action field (Action). In this case the action that is to be taken is stored in the

#IF statement data base.

3. PARAMETER -

Field Name	Type	Width
Loc_Number	Numeric	3
Param_Num	Numeric	3
Parameter	Character	30

The parameter data base was designed to hold the names of the variables that will be transfered to an application.

The location number (Loc_Number) identifies the location to which this applicatino call is related. It matches the unique location number that is in the location data base. Since the location data base already stores the name of the application to be called it would be redundant to store that information again in the parameter data base. Instead the location number is stored which can be used to get the name of the application from the location data base.

Since there may be several parameters passed to an application and order of the parameters is important, each record in the parameter data base has assigned to it the order number of each parameter (Param_Num). This is to guarantee the correct order of the information when passed to the application when it is called.

The last field in the parameter data base is used to hold the name of the variable corresponding to a parameter. During the course of the dialogue the dialogue designer will

set certain variables to be passed as parameters to an application. In order to signify which variable is to be passed, the names of the variables are stored in the parameter field (Parameter).

4. COMMAND -

Field Name	Type	Width
Com_Number	Numeric	3
Word	Character	30

The command data base is used to store the words that are related to a command. Each command is made up of one or more words that represent a group of related concepts. To group these together they all have a common command number (Com_Number). The command name is stored in the command reference data base. Each command name has a unique number to which it is related. This command number is used to group all related words in the command group.

Each word that is related to a command is stored in the word field (Word). The word may be a single word or small phrase, of up to a few seconds in length, that can be uttered into a voice input device. The dialogue could be set up to accept the phrase "Login please", even though it is two separate words they are accepted as one command word. Number groups can be set up in a similar manner.

5. APPLICATION -

Field Name	Type	Width
Applicat	Character	30
Num_of_Par	Numeric	2

The application data base is set up as a reference data base to know how many parameters are in a given application. The name of an application is stored in the application name field (Applicat). The number of parameters the application had was stored in the number of parameter field (Num_of_Par). The field was only given a width of two numbers since 99 parameters seemed more than sufficient.

6. COMMAND REFERENCE

Field Name	Type	Width
Command	Character	30
Com_Number	Numeric	3

The command reference data base is used to store the name of the command for a group of related words. To save on space this is the only data base that holds the name of the command (Command). Each command is assigned a unique number (Com_Number) that can be referenced in other data bases.

7. VARIABLE

Field Name	Type	Width
Variable	Character	30

Cvalue	Character	50
Nvalue	Numeric	3

The variable data base is used to store the variables that are used in the dialogue. The name of the variable is stored in the variable name field (Variable). Variable names have been alloted 30 characters to allow for a fully descriptive name which can include symbols and spaces. The only restriction that a variable has is that it begins with a dollar sign ($).

During the course of the dialogue, the variables are assigned a value. Since the dialogue designer was presumed to be a non-programmer, the concept of variable declaration would be unknown to him/her. Therefore, all variables can be assigned numeric or character information. Since the type of data is not known until run-time, the variable data base has two fields that are reserved for both numeric (Nvalue) and character (Cvalue) data.

8. #IF STATEMENT

Field Name	Type	Width
Ifstate	Character	50
Loc_Number	Numeric	3
Ifs_Number	Character	3
Action	Character	50

The #IF statement data base was designed to hold the

#IF statments and the actions that are taken if the statement evaluates to true. Since the action from each location is different for each #IF statement this data base was created. If all #IF statements were treated identically for each location the #IF statements could have been added to the action data base. The current program design flags the action data base that a comparison is not a command but an #IF statment. The active data base is then switched from the action data base to the #IF statement data base.

The first field in the #IF statement data base is the actual #IF statement used for variable comparison (Ifstate). This is a character string formated for #IF statements.

The second field is the location number (Loc_Number). This is the same location number as previously described. It is used to identify the location to which the #IF statement is related.

In order to identify each #IF statement at a given location a corresponding #IF statement number (Ifs_Number) is assigned to each #IF statement. This is a unique number which is used to group together all actions under a given #IF statement. To identify which #IF statement was connected to the location the #IF statement number was linked to the action field in the action data base. Since the action field is a character field the #IF statement number was stored as a character for ease in comparisons.

The Action field (Action) holds the same information as the action field in the action data base. It informs the

system at run time what action should take place if the #IF statement evaluates to true.

Screens for Editing and Building Dialogues

All screens in the UIMS follow the same format for consistency. The pattern used for the screens is shown in figure VI-2. Centered at the top of the screen is the title of the screen. This is to help identify where in the editor the dialogue designer is currently.

Screen format

Directly under the title is an area for supplementary information. This area is used to display already defined parts of the dialogue. These serve as important reminders to determine where the dialogue designer is in the dialogue. An example would be if a dialogue location label was just named then it would be displayed on all screens that are related to that location. This way the dialogue designer will know what part of the dialogue he/she is currently designing.

Beneath the supplementary information display is the data and query display. If the screen is a query screen then the questions will be displayed in this area. If the screen is a data display screen then all the data is shown in this area.

The next area on the screen is the warning and information area. This lets the dialogue designer know if there is more data to be displayed and asks if he/she would like to view it. This area also lets the dialogue designer know if any data entered in the data area was incorrect and what was wrong with it.

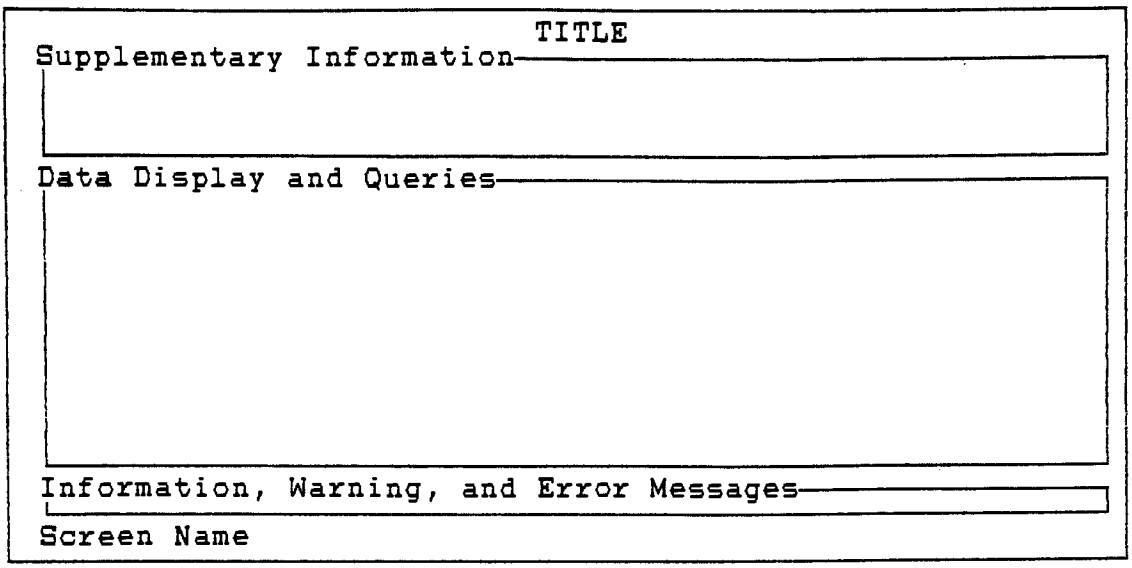


Figure VI-2. Screen Layout

There are 18 main screens, used for the editing system, defined in detail below. The screen names are located on the bottom left corner of the screen. Samples of the screens can be found in the appendix.

MENU01 - This screen allows the dialogue designer to add parts of previously defined dialogues to the current dialogue. The dialogue designer is allowed to add Commands, Applications, and Locations from another dialogue.

MENU02 - This screen is for the naming of the Dialogue. It asks the dialogue designer for a name of the Dialogue. If the Dialogue already exists then the dialogue designer is prompted to make sure he/she wants it modified. If the Dialogue does not exist the dialogue designer is questioned if he/she wants it created.

MENU03 - This screen is a display screen that allows the

dialogue designer to view all of the Location labels that have been previously defined.

MENU04 - This screen is the Main Menu for the editing/developing system. This screen has the following choices: 1. Add, modify, or delete a Location. 2. List all Locations that have been previously defined. 3. Add, modify, or delete a Command. 4. List all Commands that have been previously defined. 5. Perform a diagnostic check to the Dialogue. 6. Run the dialogue. 7. Exit the program.

MENU05 - This screen is used for the naming of a Location. If the dialogue designer needs assistance, all of the previously named Locations can be displayed for reference from this screen.

MENU06 - This screen is the Location Menu. It allows modification of all parts of a Location in the Dialogue. This screen has the following choices: 1. Define the current Location as a Prompt to the end-user. 2. Define the current Location as a call to an Application. 3. Add, modify or delete an Action for this Location. 4. List all Actions that have been previously defined for this Location. 5. Set or view the priorities for the Commands in this Location. 6. Delete this Location. 7. Create or go to another Location. 8. Return to the previous menu (MENU04.) 9. Exit the program.

MENU07 - This screen allows the dialogue designer to give the prompt to be sent to the end-user.

MENU08 - This screen allows the dialogue designer to identify which Application will be called in a Location. All of the available Applications are displayed for choosing.

MENU09 - This screen is used to enter the variables to be passed to an application as parameters. Each parameter is numbered so that correct ordering is established.

MENU10 - This screen displays all of the currently defined commands for a dialogue.

MENU11 - This screen gets the name of the command to be viewed or modified. If the command does not exist this screen will inform the dialogue designer that a new command is being generated.

MENU12 - This screen is the Command Menu. It allows the dialogue designer to choose one of the following choices: 1. Modify the current Command. 2. Delete the current Command. 3. Return to the previous menu(Menu04). 4. Exit the program.

MENU13 - This screen allows the entry of words to be grouped under a given Command.

MENU14 - This screen allows the dialogue designer to set the starting Location for the dialogue. If the starting location is to be changed this screen will accommodate the dialogue designer so that he/she can start from

any point in the dialogue. It also allows the dialogue designer to use the previously set variables from the last time the dialogue was run. The screen also prompts if the dialogue designer wants to run in a diagnostic mode by displaying the Location labels at each Location and the value of the Variables after the transfer from one location to another.

MENU15 - This screen prompts for the name of the Command or #IF statement that is to be used for comparisons. After this screen, all Actions that are added to the Location will be under this Command or #IF statement.

MENU16 - This screen displays all of the Commands and their Actions that are related to a given Location. This screen is used for checking to make sure that all possible situations have been covered in the dialogue.

MENU17 - This screen allows the dialogue designer to add, modify, or delete an Action that is related to a Command. All of the Actions that have been previously defined are displayed one at a time to be inspected and approved. Any changes may be made to them and any new Actions may be added to the bottom of the list. An entry of a blank line for an action signifies to the system, that there are no more actions to be entered.

MENU18 - This screen allows the dialogue designer to establish a priority for Commands and #IF statements. The comparisons will be performed in that order. All

of the Commands and #IF statements are displayed with the current priority. The dialogue designer may move the cursor over each priority and change it to which ever order he/she wishes. When the dialogue designer is done, he/she moves the cursor past the last priority and the order that is displayed is stored in the data base. The new priority is then re-displayed for verification.

Editing System

When the dialogue designer requests the system to create a new dialogue, he/she is prompted for the name of the dialogue. This name will be used for all files that are created related to this dialogue. Since it is running under an MS-DOS environment, the limitations of the file names are up to 8 characters. The file names are divided as such: the first seven characters are the name of the dialogue and the eighth character indicates from which part of the dialogue it comes. Due to DBASE III limitations, the file extensions will always be the same.

Presentation

The dialogue designer is presented with a screen to be filled in for each part of the dialogue. This form limits the dialogue structure so that there is consistency throughout the dialogue. At any time the dialogue designer can see all of the possibilities that can be entered for each selection.

As the user builds the dialogue, the parts that have already been built can be viewed as a guide for further construction. This is useful since it is difficult to remember the many different names already assigned in the dialogue.

Error Message Tools

The error system constantly monitors the information entered by the dialogue designer as the dialogue is being built. At any deviation from what is expected, the error system will inform the user of what the problem is and offer a solution on how it can be corrected.

In all of the modules in the UIMS there are constant checks to determine if what has been entered is appropriate for that system. Each time the dialogue designer enters a system command, the system informs the error system to check the input to make sure it is correct.

Dialogue diagnostics

One of the major benefits of the error system to the dialogue designer is the dialogue checking routine. In this routine the entire dialogue is checked as a whole. This allows the user to see if there are any missing or misplaced parts of the dialogue. Checks are performed to see if each part of the dialogue is accessible. If a location in the dialogue is unaccessible then this is flagged as an error. The only time there is no access for a particular location is when the location is the initial location of the dialogue.

A check is also made to see that there is at least one exit from each location. Since all dialogues are basically cyclical, there does not have to be a designated exit from the dialogue. It is possible for the final state in the

dialogue to be a transfer from the end to the beginning of the dialogue in order to start the dialogue over again.

If there exists a transfer from one location to a non-existing location in the dialogue, a warning message is displayed to inform the dialogue designer. This can also occur if the dialogue designer is only partially finished with his/her design and wishes to test it for errors. If a misspelling of a location label occurs in a branch to a location, then that location will be flagged as a non-existing location.

The dialogue designer has the choice of ignoring all warning messages and running the current system knowing that this situation exists. This is not advisable but it is allowable. It is the responsibility of the dialogue designer to avoid the areas of conflicts indicated by the error checking tools.

Run Time Tools

When the dialogue designer has reached a point in the dialogue where he/she feels the dialogue should be viewed in operation, the run time system may be invoked. This is done through option number 6 (Run the Dialogue) on menu04.

Run time parameters

After the run time system is invoked the next screen to appear before the dialogue is executed is Menu14. The first parameter of the run time system that is presented for change is the starting location of the dialogue. There are advantages in having this capability. One of the main advantages of this function was to have the option of starting the dialogue from any location. If the dialogue was previously run and the dialogue designer wishes to continue the dialogue from the last accessed location, then he/she may change the starting location to the last accessed location.

The next prompt that is displayed is for the use of the previously set values for variables. If the dialogue has been executed and variables have been assigned values, those values may be maintained. This is useful when re-entering the dialogue from a new location without having to start the dialogue completely over again. If new features are going to be tested, then it might be advantageous to continue from the point from where the dialogue was exited, with the previously set variable values.

The final prompt from Menu14 is used if the dialogue designer wishes to view the internal workings of the dialogue while it is being tested. This feature will display the location label in the top left corner of the screen as each location is entered. After all comparisons are done, it will display the values of all variables to ensure that all assignments are being made correctly and in their proper order.

Prompts to the end-user

Prompts to the end-user are displayed on the prompt screen, which is a simple screen with two fields. The top field is where the prompt, as designed by the dialogue designer, is displayed. The second field is the input field, where the response from the end-user may be entered.

Application interface

When the application is called by the dialogue, the information is displayed on the application screen. The application screen takes the place of the application interface temporarily. This screen displays the name of the application, the parameters that have been passed to the application, and an entry field that is used to enter the return value from the application.

The application screen allows the dialogue designer to test all possible return values for any given situation in the dialogue. It also allows the dialogue designer to view

all of the values that were assigned to variables, corresponding to each parameter. This is another method of determining if the order of the variables used for parameters is correct. It also verifies that the correct information is being passed to the application.

Response analysis

The response that has been entered from either the end-user or an application, determines the next action in the dialogue. Each word that is entered is compared to the list of words related to each command. If it is found, then the value of that command is returned to the run time system to determine the next action. This analysis also sets the flags for the #ANY and #NONE commands so that all comparisons may be made. If the word entered did not match any command words, then an invalid term value is returned.

#IF statment analysis

If the command is valid, and an #IF statment is encountered in the dialogue, it is sent to the #IF statment parser for evaluation. This parser will determine the values for all variables and make the proper comparisons to determine if the statement evaluates to true or false. This informs the run time system to either execute or ignore the next action. There are six separate comparisons that can be performed by the #IF statement tool (=,<>,<,>,<=,>=).

Variable analysis

If there is a variable assignement in the dialogue it is handled through the variable assignment and parser tool. This tool evaluates all of the values that are in a variable assignment statement and performs the correct operation on each. The result is then placed in the variable to be assigned. The only mathematical operations that can be perfomed by the variable analysis tool is addition (+) and subtraction (-). These have the capability of counting events and keeping track of occurrences in the dialogue.

System Constraints

The UIMS only handles the voice technology that exists in the current commercial market. Most commercially available voice input systems are isolated utterance and are used with microprocessors based computer systems. More enhanced versions of the system are discussed throughout this paper but not implemented in the final design due to timing and system constraints.

Application constraints

The voice I/O used by the UIMS is handled as part of the dialogue and not as data. Each word uttered by the end-user is considered as part of the dialogue and not merely passed on to the application for evaluation as data. This limits the UIMS dialogues to not include listening-typewriters and similar applications. Dialogues for these applications may be designed on the UIMS but is not the primary consideration for this UIMS.

Hardware specifications

The UIMS was designed on a microprocessor based system. The actual hardware configuration is a Digital Equipment Corportation (DEC) Rainbow 100b. The hardware includes an 8088 processor and a Z-80 processor with 2 single-sided quad-density disk drives and 320k memory. This choice was based upon availability and not preference.

It is desireable that this UIMS be transferred in the

future to a microprocessor based system with a hard-drive and more memory. The future hardware may also support voice I/O for more appropriate dialogue testing. Accomodations have been made to simulate dialogue testing with the current system. The motivation behind recommending that the implementation should be done on a microprocessor is due to the recent trend towards these machines.

Software specifications

The software for this system was written in DBASE III. This decision was made after evaluating the avaliable software tools and the needs of the UIMS. Since the system is used mostly for keeping track of interrelated data it seems logical to choose a software package that was designed for that purpose. The graphical part of the package was not implemented in this version since it is an unsupported feature of DBASE III. The implementation of the graphical part of the system was weighed not as important as implementing the central layer of the UIMS.

System interface

The system interface is handled through the system hardware. The input is handled through a keyboard for text entry and the output is done on a CRT.

Voice UIMS implementation

The central layer of the voice UIMS was implemented.

This included all tools necessary to implement and test a working dialogue. The end-user presentation coding was not implemented due to the capabilities of the software tools. This limitation was also true for the application interface. These two sections are presented through supplementary screens which allow the dialogue designer to send information to the UIMS from either the application or the end-user. This was handled through the run-time tools. A main error routine was not impelemented due to the structure of DBASE III programming language. The design process would have been slowed greatly had this been implemented.

Record keeping tools were not implicitly built into this version of the UIMS. Since the dialogue was written with a data base program, it is possible to use the features that are in DBASE III to view the dialogue as it is proceeding and to obtain information about the dialogue in this manner.

A working run time system was implemented to allow the dialogue designer to view the dialogue in action. This feature also included a screen that allowed the dialogue designer to see all of the variables that had been assigned and their values. It was also possible to view the names of the location labels at each location in the dialogue. During application calls the dialogue designer could view all of the values that were being passed as parameters and respond to them from the perspective of the application.

This implementation of the UIMS was able to handle

conventional dialogues that use only voice as the interface. There were no problems when designing dialogues on this system though some dialogues were easier to implement than others. There have been no indications that the tools that were implemented were not sufficient for the purpose of the experiment. The basic building blocks necessary for building a voice I/O dialogue have been implemented although the techniques used to implement them were not necessarily the most appropriate. It is believed that with further testing and development this UIMS can be developed into a very powerful and useful interface tool for voice I/O applications.

System Walk-through Test

After the system was completed it was tested three ways: a walk-through, a function test, and a simulated user test. The first was a systems test to ensure that all parts of the system were working. An overall inspection of each software module was done. This included a visual inspection of the software with a walk-through of all modules. A consistency check was done to ensure that all modules would handle data the same way and that all code was consistent throughout each module. A comparison was done between the design and the implementation to be sure that what was finally implemented was the same as what was originally designed.

Functional Test

The second of the major tests was a functional test of the system. This included testing each module for all possible types of acceptable input to ensure its robustness. Each module was also tested with erroneous data to ensure the system would be able to handle some types of errors that the user would be likely to enter. A test of all major paths in the system was done to see if it was possible to transverse the dialogue the way a user would without any difficulty. A full feature test was done to make sure that all implemented features worked the way they were described to work.

Simulated user test

The last check of the system was a simulated user test, where a dialogue was built using the system to see if the system worked as a whole. This included the use of as many features as were necessary to implement a dialogue.

The Dialogue Designer Experiment Setup

In order to test the voice UIMS it was necessary to design an application to be implemented. The goal was to design an application short enough so that implementation of the dialogue would not require a lot of time, simple enough for the dialogue designer to understand, and yet complex enough so that the system could be tested for all of its capabilities.

Test Application Description

An application was designed with a limited vocabulary using a data base application. This application consisted of a shoe store inventory database that was accessible by an end-user through a voice I/O system. The application was purely fictional and designed merely to test the UIMS. The application program was designed with five different categories of input. These were 1. the gender of the customer (looking for men's or women's shoes), 2. the size, 3. the width, 4. the color, and 5. the style of the shoe. The application returned a zero (0) if the shoe was in stock and a one (1) if the shoe was not in stock.

Test Application Vocabulary

A vocabulary was built for the voice input with a variety of words to satisfy the needs of this experiment. The words included: men, women, male, female, the numbers from six to eleven, the letters A to E, double-E (EE),

black, brown, tan, white, work, dress, and casual. At any time during the session, the dialogue designer could build a larger vocabulary or change the existing one to his/her satisfaction.

Test Application Documentation

A two page summary of the system commands and terminology was written so that the dialogue designer could understand the basic building blocks of the UIMS. A one page description of the application was written so that a purpose for the dialogue could be established. These are included in the appendix.

Results

The Initial Dialogue

After the dialogue designer had read through the documentation of the system and had a brief explanation of the system, he proceeded to build a dialogue. His original dialogue was drawn out on paper in a series of boxes with labels explaining the purpose of each box in the dialogue. Each box represented a location in the dialogue. The flow of the dialogue was indicated by lines and arrows which connected the boxes. After an initial attempt at the dialogue on the paper, the dialogue designer proceeded to implement the primitive dialogue which he had designed. He then continued increasing the dialogue using the voice UIMS dialogue design tools.

The "Straw Man" Design

The initial design that had been done by the dialogue designer was described as his "straw man" design. This was a representation of the skeleton of the dialogue with many details of the dialogue omitted. Future revisions for the dialogue were discussed aloud. The dialogue designer decided that after the dialogue constructions were started, more of the details could be filled in later.

Initial Problems Using the UIMS

At the start, some problems with the UIMS arose. The first was that the dialogue designer had designed longer than 50 character prompts to the end-user. This constraint

was designed based on readings that speech output should be used when the message was short [MICH82].

Another problem was in the understanding of the system vocabulary and its uses, for example the term "Command." It is strongly suggested that this term be changed to a more appropriate term to better accommodate its function. The term "token" seems like a more appropriate choice but might not be appropriate for someone not familiar with computer jargon. This problem continued until the dialogue designer understood the purpose of the Command and adjusted to its function.

One main reason for the confusion in the vocabulary and functions of this system by the dialogue designer was his background. The dialogue designer had a programming background which initially interfered with the terminology of the UIMS. It is recommended that an attempt be made to portray each function in the dialogue in a more consistent manner. An example would be having the Commands displayed in a format similar to the #IF statement. This would appear: "IF [command] = [response]." This format might offer more of an explanation of the function of the Commands.

Dialogue Constraints

There was some initial concern on the part of the dialogue designer about the design of the total interface to the end-user. A human factors engineer normally would handle

both the dialogue and the other aspects of the end-user interface. Since this experiment only concerned the creation of a dialogue using the UIMS, the dialogue designer was instructed to assume that those aspects were already handled and to concentrate on the dialogue alone.

Initial Prompts Problem

After the name of the dialogue was entered, the dialogue designer was prompted to use an already created dialogue for assistance. On the screen, the prompts read: "Enter the name of the Dialogue whose Commands you wish to use." When this prompt was read aloud, the dialogue designer skipped over the part about the Commands. The dialogue designer then proceeded to take the Commands from a previously created dialogue and copy them into the new dialogue. The original plan was that he only take the applications that have been defined in another dialogue so that he would not have to redefine the application. It is recommended that the prompt be shortened, reworded, or have the key words for its functions highlighted in a way to make the request more obvious to the dialogue designer. This problem might have also originated from a problem in the understanding of the vocabulary used in this UIMS. This error did not have a detrimental affect on the experiment.

Feature Omission

The first feature recommended to be added to the UIMS

was a list of all possible choices whenever a choice had to be made. This was previously described as one of the features in the UIMS. Had a complete help system been implemented this would not have been a problem. It was also suggested that if a list is presented there should be a way to choose from the list and not have to type in a response by copying it from the list. Unfortunately the software constraints of DBASE III did not allow this implementation. A list was provided for location and Command selection. This proved very helpful to the dialogue designer and he expressed that this availability should be made for all choices.

Future Prompts

It was obvious from the use of the UIMS by the dialogue designer that some of the prompts presented by the UIMS were not as self-evident as they were designed to be. It is suggested for future modifications of this system that a dialogue designer design the prompts and screens for this system. It is also recommended that a format skeleton be presented so that formatting problems be reduced. This would indicate the format of the data that is expected from the dialogue designer. The implementation of a full error checking system would have given syntax warnings as the entry was being made.

Progress by the Dialogue Designer

After the dialogue designer comprehended the features of the system, the dialogue creation went smoothly. The creation of the dialogue followed the suggested order for dialogue creation in the UIMS description. The concept of prompts were understood and were the first part of the dialogue that was easily implemented.

Command-Actions

At one point in the dialogue, the problem concerning comprehension of the command-action relationship arose. This again is partly due to the terminology used in the system. It was explained that one function of the actions was to transfer the dialogue from one location to another. If there had been a graphical representation of the dialogue as the dialogue designer was building it, he/she would have seen the missing connection between the locations and created that link. In this implementation the only way to find this out was to run the checking diagnostics on the dialogue. Once the idea of the command comparison to action relationship was explained, there were no futher problems with the dialogue. When the dialogue was checked by the dialogue diagnostic package, there were no errors of missing links between locations.

Menus

When a menu was presented that had several items on it,

the dialogue designer had a hard time reviewing all of the items. Some of the separations of categories on the menus did help in this situation. It might be further recommended that the key words for each menu selection stand out so that they can be scanned easily. The splitting of topics on the menus did appear to be of some assistance.

Error Correction

One feature suggested by the dialogue designer was that when presented with an error or warning message by the system, there should be a way of editing the original entry. This feature was overlooked when designing the system. All entries, when tested true for an error, were erased from the screen so that they could be typed again correctly. It would be better if there was a mechanism to correct the field where the error had occurred. This would have prevented the extra keystrokes needed by the dialogue designer.

Highlighted entry fields

Any field on the screen that was used to enter data was highlighted so that the dialogue designer would know where the information would be placed. On several screens it was noted after the experiment, that the highlighted areas were not removed after an entry in that area had been made. These were the screens that allowed a list of entries to be made. It is recommended that the data entry area be

highlighted only. The highlight should be removed once a satisfactory entry had been made and a new area appears below it.

Errors for major revisions

There should be a full error checking system implemented to the UIMS. It was not implemented due to time and system constraints.

More information displayed

It was noticed that some important information was omitted from the supplementary information part of the screen. The information related to the dialogue location proved helpful when displayed. When a location was not defined as a prompt or an application it was not indicated. The dialogue designer, not being familiar with the system, took no notice of this dilemma. He assumed the location was defined as an application call and proceeded to omit the step of defining it as such. Had this information been displayed this oversight might not have occurred.

Global changes

Certain features were omitted from the design due to the time limitations in implementing the system. One of these omissions was the ability to globally change the names of locations or commands. Without this feature the dialogue designer had to either accept the names that he previously

defined or manually redefine all of the names in the system. It is obvious for a good user interface design that this feature should be implemented.

Dialogue designers impressions of the system

After the dialogue was built and tested the dialogue designer commented on the system. It was the general opinion of the dialogue designer that this UIMS had potential. He would prefer the system if it required less input being done through a keyboard.

The dialogue desginer felt that tools such as the UIMS were of valuable service in developing dialogues. He felt that there should be a way of avoiding the complexities of programming languages for the implementation of dialogue designs yet provide capabilities for the creation of a dialogue. He also felt that it is extremely difficult to implement a generic UIMS that could handle all situations.

The ability to bring in pre-defined libraries of commands, applications, and structures of the dialogues was appreciated. The dialogue designer felt that this was a valuable feature that could be useful when designing many dialogues. It was also remarked that the idea of having command comparisons helped his job as a dialogue designer by not having to design them into the dialogue.

As a general overview of the system he was very pleased in being able to see the entire dialogue in its completion at the end of the experiment. He also enjoyed seeing the

the dialogue in action when he was running the test of the dialogue he designed. He felt that the UIMS was a beneficial tool to dialogue designers.

Chapter VIII - Conclusion

Changes for future implementations

There are some changes to the system that should be implemented to improve the system. One change that should be made is to the vocabulary used in the system. The words used in the system were designed to cover certain concepts in the system. No testing was done to decide the actual terms that should be used. Consequently, there was some ambiguity related to these words from the dialogue designer. The term that proved most confusing to the dialogue designer was the term "Command." It is recommended that a vocabulary be established by a group of dialogue designers to cover all concepts used in this UIMS.

If further development is done, it might be advised to add another field to the application reference data base. This field would be a format field, which would specify in what format the application would like to see the data. This could be as general as the difference between numeric and character or more specific to include the number of decimal places and right or left justification. This field was not necessary for the experiment since the application interface was not implemented.

Many limitations were placed upon the implementation of this UIMS because of the hardware and software used. Concepts like #TIME functions were impossible to implement due to the software restrictions. It is also extremely difficult to link the data base program to any other

external programs with the available software. It is, therefore, recommended that future developmental systems have support features that can handle the implementation of the entire design. This would include full graphics capabilities and supporting software capable of interfacing into the UIMS developmental system. Additional hardware should be added to the system to support pointing devices for the graphical interface to the UIMS. The system hardware should also include enough memory and storage to hold the UIMS, all of its supporting software, and the dialogue with a minimum of delay for access time.

It is also advisable to implement all of the design and coding changes that have been previously discussed. These changes could have a great affect on the useability and success of this system.

Impressions on this topic

Due to the increasing literature that has appeared on UIMS since its inception, it is the belief of this author that research and development will continue in this area. There is a growing concern in the human factors area in speech technology, as indicated by the literature. Although UIMSs will not resolve all of the problems that are involved in the current technology, it will greatly reduce the frustrations that are expressed by the end-users.

More system designers have been taking an interest in the ease-of-use of the UIMS system. With an increasing

interaction between system designers and human factors specialist, there will be a production of better quality systems as perceived by the end-user.

References

[ALLE79] Allen, J., An Overview of LISP, Byte, 4, (8), August 1979, pp. 10-16,118-122.

[BETT87] Betts, B., Burlingame, D., Fischer, G., Foley, J., Green, M., Kasik, D., Kerr, S.T., Olsen, D., Thomas, J., Goals and Objectives for User Interface Software, Computer Graphics, 21, (2), April 1987, pp. 73-78.

[BROW84] Brown, P.J., Error Messages: The Neglected Area of the Man/Machine Interface?, Communications of the ACM, 26, (4), April 1984, pp. 246-249.

[BULK86] Bulkey, D., Freedom of Speech, PC Products, 3, (3), March 1986, pp. 53-66.

[CAPE21] Capek, K., R.U.R, Originally published in English by Oxford University Press 1923.

[CATE83] Cater, J.P, Electronically Speaking: Computer Speech Generation, Published by Howard W. Sams & Co., Inc, 1983.

[CHAP72] Chapanis, A., Ochsman, R.B., Parrish, R.N., and Weeks, G.D., Studies in Interactive Communication: I. The Effects of Four Communication Modes on the Behavior of Teams During Cooperative Problem-Solving, Human Factors, 1972, pp. 487-509.

[CHAP77] Chapanis, A., Parrish, R.N., Ochsman, R.B., and Weeks, G.D., Studies in Interactive Communication: II. The Effects of Four Communication Modes on the Linguistic Performance of Teams during Cooperative Problem Solving, Human Factors, 1977, pp. 101-126.

[CHAP65] Chapanis, A., Man-Machine Engineering, Brooks/Cole Publishing Company, pp. 64-92.

[COHE85] Cohen, A., and Cuperman, V.M., Tools and Techniques for System Intergration of Speech Technology, Proceedings from Speech Tech '85, 1985, pp. 336-340.

[DECT85] DECtalk A Guide to Voice, Published by Digital Electronic Corporation, 1985.

[DREY86] Dreyfus, H., and Dreyfus, S., Why Computers May Never Think Like People, Technology Review, January 1986, pp. 42-61.

[ENDE83] Enderle, G., Report on the Interface of the UIMS to the Application, User Interface Management Systems, Proceedings of the Workshop on User Interface Management Systems held in Seeheim, FRG, November 1-3, 1983, pp. 21-29.

[FINK86] Fink, D.F., and Holen, R., Speech I/O Increases Industrial Quality Standards, Speech Technology, 3, (2), March/April 1986, pp. 70-72.

[FREE75] Freeman, P., Software Systems Principles A Survey, Science Research Associates, Inc., pp. 480-481.

[GARD85] Gardner, D.J., DeFruiter, D., Keith, M., and Kline, M., Automated Speech Recognition as a Function of Formal Speech Training and Passage of Time Between Template Training and Testing, Proceedings of the Human Factors Society, 1985, pp. 937-941.

[GOLD85] Goldhor, R.S., A Large-Capacity Highly-Programmable Speech Recognition System, Kurtzweil Applied Intelligence Inc., Proceedings from Speech Tech '85, 1985 pp. 123-126.

[GORE80] Gore, M., and Stubbe, J., Elements of Systems Analysis for Business Data Processing, Wm. C. Brown Company Publishers, pp. 136-482.

[GREE86] Green, M., A Survey of Three Dialogue Models, ACM Transaction on Graphics, 5, (3), July 1986, pp. 244-275.

[GULI84] Gulian, E., Fallside, F., and Hinds, P., Can Deaf Children Interact with Computers? Evidence from Speech Acquisition Training, Interact '84, 1, pp. 164-168.

[HILL86] Hill, R.J., Interaction - The Sassafras UIMS, ACM Trasactions on Graphics, 5, (3), July 1986, pp. 179-210.

[HIX86] Hix, D., Harston, H.R., An Interactive Environment for Dialogue Development: Its Design, Use, and Evaluation, CHI '86 Proceedings, April 1986, pp. 228-234.

[JACO84] Jacob, R.J.K., Using Formal Specifications in the Design of a Human-Computer Interface, Communications of the ACM, 26, (4), April 1984, pp. 259-264.

[JOOS86] Joost, M.G., James, F.G., and Moody, T., Ergonomics Research in Speech I/O, Speech Technology, Mar/Apr 1986, 3, (2), pp. 42-47.

[KING84] King, R.W., Cope, N., and Omotayo, O.R., Videotex for the Blind: Design and Evaluation of Braille and Synthetic Speech Terminals, Interact '84, 1, pp. 158-163.

[KURT85] Kurtzweil VoiceSystem 3000 User's Manual, June 1985.

[LANT87] Lantz, K.A., Tanner, P.P., Binding, C., Huang, K., Dwelly, A., Reference Models, Window System, and Concurrency, Computer Graphics, 21, (2), April 1987, pp. 87-97.

[MANG86] Mangione, P.A., "SSI's Phonetic Engine," Speech Technology, Volume 3, Number 2, March/April 1986, pp. 84-86

[MART87] Martin, G.L., The Utility of Speech Input in User-Computer Interfaces, MCC Technical Report Number HI-021-87, December 1987.

[MICH82] Michaelis, P.R., and Wiggins, R.H., A Human Factors Engineer's Introduction to Speech Synthesizers, Directions in Human/Computer Interaction, ed. A. Bradley, B. Schneiderman, Norwood N.J., Ablex Publishing Corporation, pp. 149-178.

[NORM84] Norman, D.A., Design Rules Based on Analyses of Human Error, Communications for the ACM, 26, (4), 1984, pp. 254-258.

[NUSB83] Nusbaum, H.C., Schwab, E.C., and Pisoni, D.B., Perceptual Evaluation of Sythetic Speech: Some Constraints on the Use of Voice Response Systems, Proceedings of the 3rd Voice Data Entry Systems Applications Conference, Synnyvale, California,: Lockheed, 1983.

[OLSE87] Olsen, D.R., Kasik, D., Rhyne, J., Thomas, J., ACM SIGGRAPH Workshop on Software Tools for User Interface Management, Computer Graphics, 21, (2), April 1987, pp. 71-72.

[PAUL86] Paul, D.B., Lippmann, R.P., Chen, Y., and Weinstein, C.J., Robust HMM-Based Techniques for Recognition of Speech Produced Under Stress and in Noise, Proceedings from Speech Tech '86, 1986, pp. 241-249.

[PECK84] Peckham, J.B., Automatic Speech Recognition - A Solution in Search of a Problem?, Behaviour and Information Technology, 1984, 3, (2), pp. 145-152.

[PISO85] Pisoni, D.B, Greene, B.G., and Nusbaum, H.C., Some Human Factors Issues in the Perception of Synthetic Speech, Proceedings from Speech Tech '85, 1985, pp. 57-61.

[RICH84] Richards, M.A., and K.M. Underwood, How Should People and Computers Speak to Each Other?, Interact '84, 1, pp. 33-36.

[RUBI84] Rubinstien, R., and Hersh, H., The Human Factor: Designing Computer Systems for People, Digital Press, Maynard Massachusetts, 1984, pp. 82-89.

[SALT83] Salton, G., and McGill, M.J., Introduction to Modern Information Retrieval, Published by McGraw-Hill, Inc., 1983.

[SCHO85] Schoen, J., When You Talk, Your PC Listens, PC Magazine, March 5, 1985, pp. 122-132.

[SCHU85] Schurick, J.M., Williges, B.H., and Maynard, J.F., User Feedback Requirements with Automatic Speech Recognition, Ergonomics, 1985, 28, (11), pp. 1543-1555.

[SHNE86] Shneiderman, B., Seven Plus or Minus Two, Central Issues in Human-Computer Interaction, CHI '86 Proceedings, 1986, pp. 343-349.

[SMIT82] Smith, D.C., Irby, C., Kimball, R., Verplank, B., and Harslem, E., Designing the Star User Interface, BYTE 7, (4), April 1982, pp. 242-282.

[TANN85] Tanner, P., and Buxton, W., Some Issues in Future User Interface Management Systems(UIMS) Development, in G. Pfaff(ed), User Interface Management System, publisher Berlin: Springer-Verlag, 1985, pp. 67-69.

[TREM76] Tremblay, J., and Sorenson, P.G., An Introduction to Data Structures with Applications, Published by McGraw-Hill, Inc., 1976, pp. 72-73.

[VANP85] Van Peursem, R.S., Do's and Don'ts of Interactive Voice Dialog Design, The Official Proceedings of Speech Tech '85, 1985, pp. 48-56.

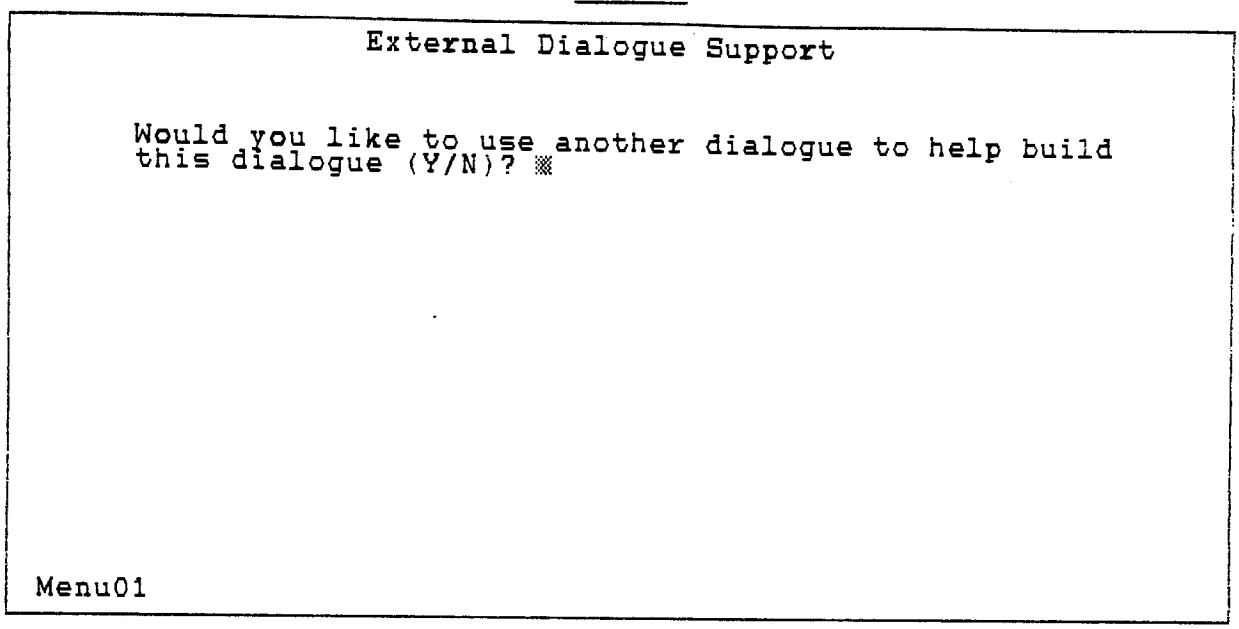
[VN5185] Scott Announces Continuous Recognition Terminal, Voice News, 5, (1), p. 1, January, 1985.

[WIEG84] Wiegner, K.K., If Machines Could Hear, Pigs Could Fly, Forbes, December 31, 1984, pp. 118-119.

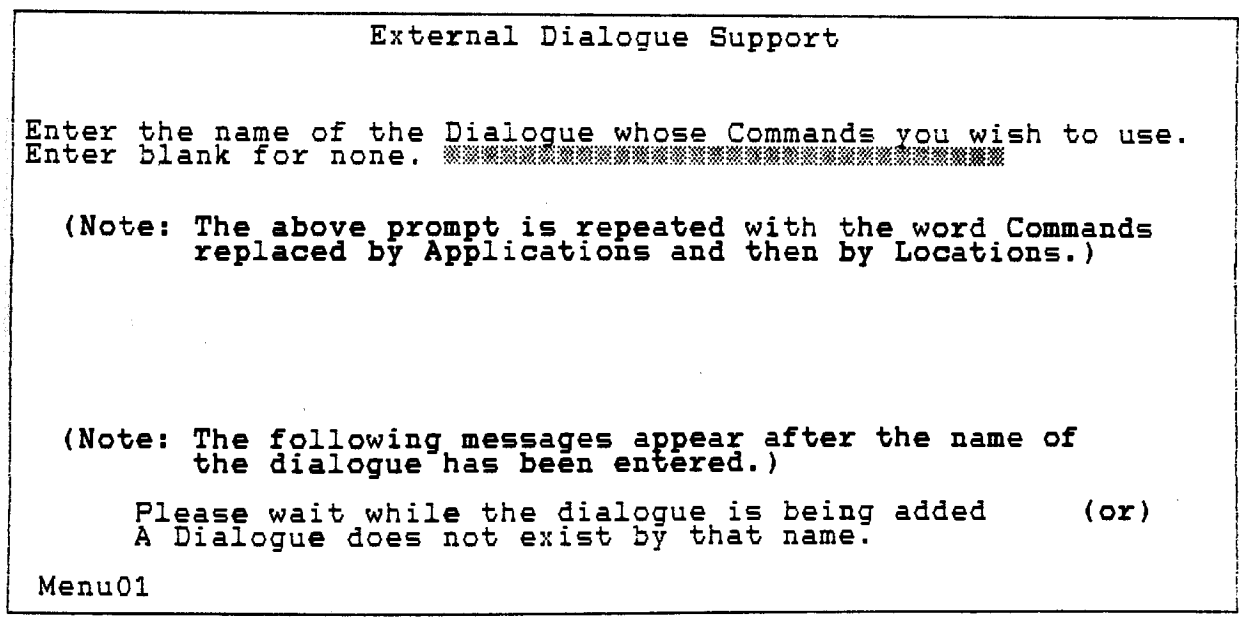
[WONG85] Wong, P. C. S., Rapid Prototyping of Voice Systems Using FLAIR, Proceedings from Speech Tech '85, 1985, pp. 324-326.

Appendix

Menu01



Menu01



Menu02

```
                    Dialogue Name


      Enter the name of the dialogue: ▒▒▒▒▒▒▒
      (The name may be up to 7 characters long)




      (Note: The following message appears after the name
             of the dialogue has been entered.  If it does
             exist then the message reads: does not exist
             Do you wish to create it (Y/N)? : ▒ )

       This Dialogue already exists
       Do you wish to modify it (Y/N)? : ▒
Menu02
```

Menu03

```
                    Location Display
Prompt or
Application     Location


     P          LOCATION NUMBER ONE
     A          CALL APPLICATION NUMBER THREE
                TEST NUMBER TIMES THROUGH LOOP
     P          TELL END-USER ENTER A COMMAND


     (Note: The list of location above have been inserted
            as an example.  If there are no Locations to
            Display then the message: "There are no
            Locations to display" appears.)

     (Note: If the screen is filled with Locations then
            The message: "Do you wish to see more (Y/N)?"
            appears.)

       Press <Return> to return to menu.

Menu03
```

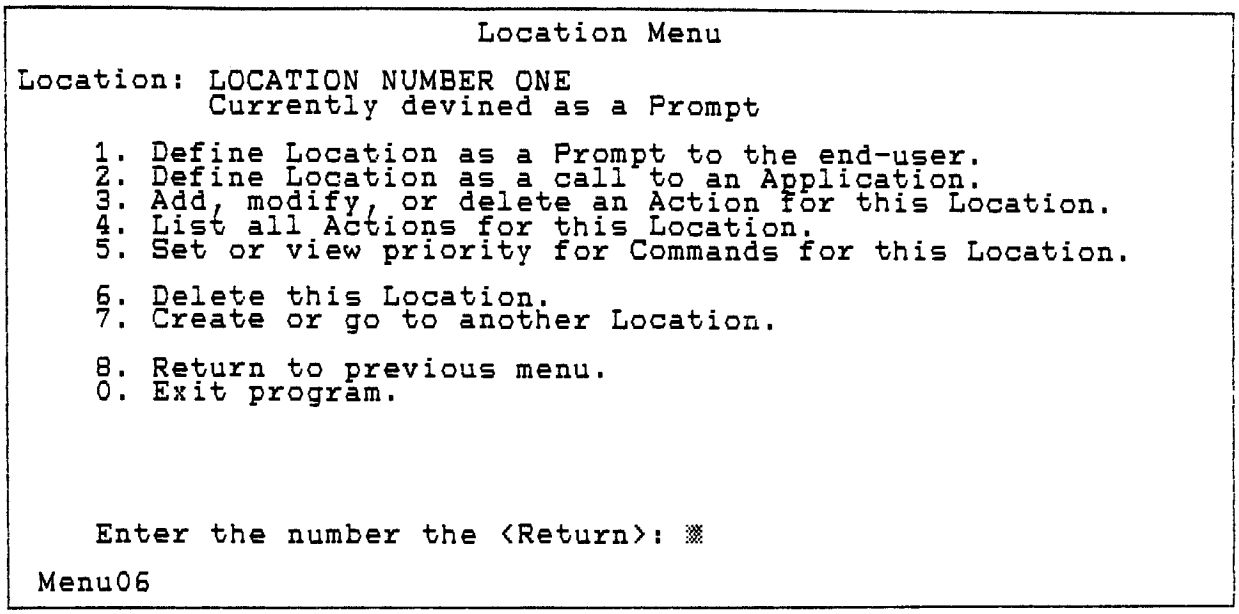
Menu04

```
                    Main Menu


                1. Add, modify, or delete a Location.
                2. List all Locations.

                3. Add, modify, or delete a Command.
                4. List all Commands.

                5. Check the Dialogue.
                6. Run the Dialogue.

                0. Exit Program.




      Enter the number then <Return>: ▒
Menu04
```

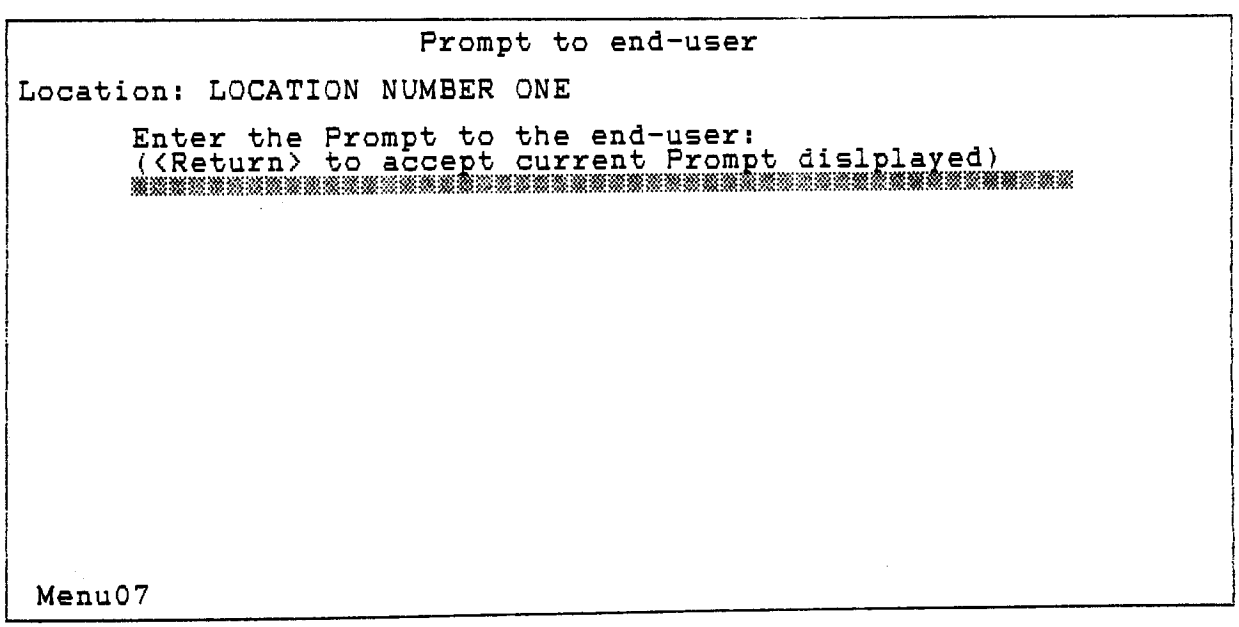
Menu05

```
                Location Label Name

      Enter the name for the Dialogue Location label:
      ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
      (Up to 50 characters may be used for the Location Label)


      (Note: If the Location label did not previously
             exist then it is created and the following
             message appears in the message area:
             "Generating new location - please wait."


      Press the "Do" key for list of Locations.


Menu05
```

Menu06



Menu07



Menu08

```
                        Application
Location: LOCATION NUMBER ONE
      Enter application to be called: ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

          Available Applications:
                          APPLICATION NUMBER ONE
                          APPLICATION NUMBER TWO
                          APPLICATION NUMBER THREE


       (Note: The following messag appears if the
              application is not found.)

       This application does not exist
       Press <Return> to continue

 Menu08
```

Menu09

```
                Parameters for Application
Location: LOCATION NUMBER ONE
          Application: APPLICATION NUMBER ONE
          Number of Parameters: 3

          Enter the variables to be passed as parameters:

          1. $VARIABLE NUMBER ONE
          2. $VARIABLE NUMBER TWO
          3. ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

    (Note: The following message may appear:
           "Variables must start with a "$" "
            when appropriate.)



 Menu09
```

Menu10

```
                        Command Display
Command

#ANY
#NONE

COMMAND NUMBER ONE
COMMAND NUMBER TWO
COMMAND NUMBER THREE



        (Note: If there is more to be displayed then
               the following message is displayed:
               "Do you wish to see more (Y/N)? *")



         Press <Return> to return.
 Menu10
```

Menu11

```
                        Command Name


     Enter the name for the command:
     ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
     (Up to 30 characters may be used for the command)









        (Note: If the command was not previously defined
               the following message is displayed:)

        Generating new command - please wait
 Menu11
```

Menu12

```
                        Command Menu
Command: COMMAND NUMBER ONE

                    1. Modify current Command.
                    2. Delete current Command.

                    3. Return to previous menu.
                    0. Exit Program.

        Enter the number then <Return>: ▒
 Menu12
```

Menu13

```
                        Command Words
Command: COMMAND NUMBER ONE

          Enter the Words for this Command:
          (Enter blank line when done)

          1. WORD NUMBER ONE
          2. WORD NUMBER TWO
          3. ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

 Menu13
```

Menu14

Dialogue Information

The current starting dialogue location is:
LOCATION NUMBER ONE

(Note: If a change of starting location is requested then the location that is displayed is blanked out and an input field is put in its place.)

Do you wish to modify it (Y/N)? ▒

Menu14

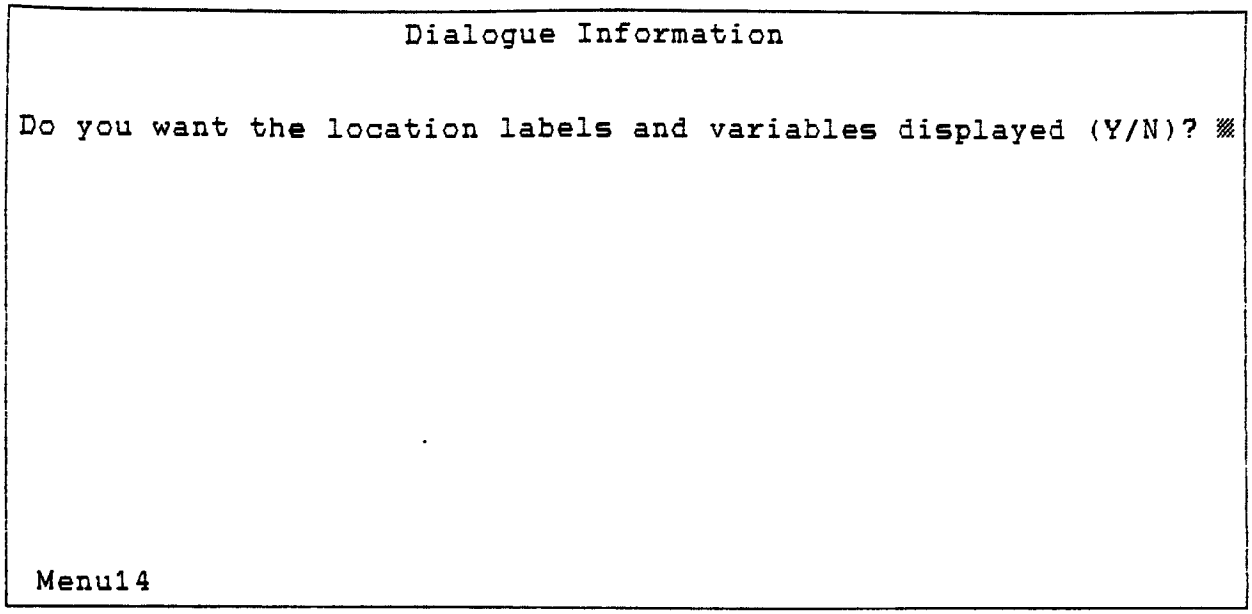
Menu14

Dialogue Information

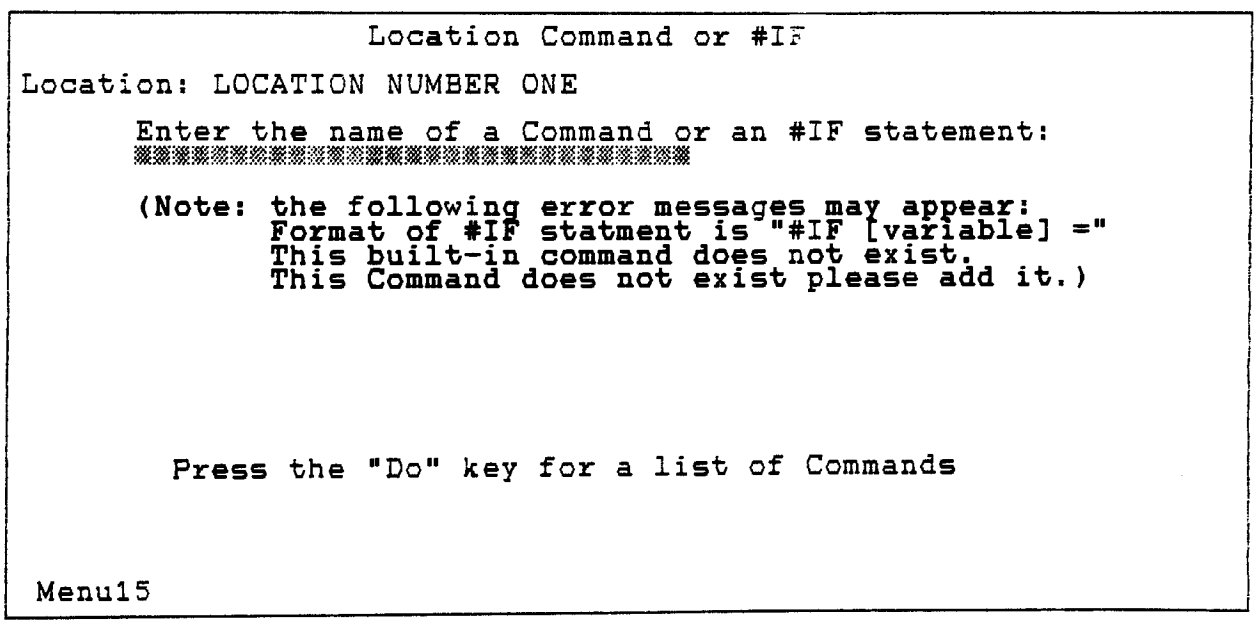
Do you wish to use previous values for variables(Y/N)? ▒

Menu14

Menu14



Menu15



Menu16

```
                        Action Listing

Location: LOCATION NUMBER ONE
Command                         Action

#IF $VARIABLE = 23              GOTO LOCATION NUMBER ONE
COMMAND NUMBER ONE              GOTO LOCATION NUMBER TWO
COMMAND NUMBER TWO              $VARIABLE = $VARIABLE + 1

    (Note: The above list has been insterted as an example.
           If there are no commands to be displayed then the
           message :"There are no Commands to display for this
           Location."  is displayed.)

    (Note: If there is more to be displayed the following
           message is replaced with: "Do you wish to see
           more (Y/N)? ".)

       Press <Return> to return to menu.

 Menu16
```

Menu17

```
                        Action Entry

Location: LOCATION NUMBER ONE
Command:  COMMAND NUMBER ONE
          Enter the Actions for this Command:
          (blank over an action to delete it)
          (Enter a blank line when done)

          GOTO LOCATION NUMBER TWO
          ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

          (Note: The Action above was inserted as an example.)

          (Note: The following messages may appear:
                 "Actions must be a "GOTO" or variable
                 assignment",  "Variable must be in the format
                 "[variable] = ".)




 Menu17
```

Menu18

```
                    Command Priority
Location: LOCATION NUMBER ONE

    Priority  Command

         1      #IF $VARIABLE = 23
         2      COMMAND NUMBER ONE
         3      COMMAND NUMBER TWO

        (Note: The above commands were inserted as an example.)

        (Note: The following messages may appear:
               "There are no Commands for this Location to
               display.",  "Do you wish to modify the
               order (Y/N)? # ", "Move cursor up and down and
               enter new Priority.", "Move cursor passed last
               value to enter order.")


 Menu18
```