Interface Design for an Audio Based Information Retrieval System

by

James Robert Johnson

Project report submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Information Systems

in

Computer Science

APPROVED:

Edward A. Fox, Chairman

Beverly A. Williges

Robert C. Williges

July 15, 1992

Blacksburg, Virginia

Interface Design for an Audio Based Information Retrieval System

by

James Robert Johnson

Edward A. Fox, Chairman

Computer Science

(ABSTRACT)

This project involves a telephone-based information retrieval system. Users interact with the computer by pressing buttons on a telephone keypad and listening to the computer respond by way of a speech synthesizer.

The purpose of this project is to redesign and revise an existing information retrieval system. The goals of this project include simplifying the job of the menu designer and providing a way so experience can aid users to perform a given task faster than previously possible. Key objectives of this project are adding a keyword command interface to the existing menu driven interface and developing a flexible menu representation scheme for the menu designers.

The original system is part of a National Science Foundation funded project entitled "Integrated Research Paradigm For Information Technology". The system works by speaking lists of keywords using a synthesizer and allowing the user to choose the most appropriate keyword from each list. After selecting a keyword the user is given another list of more specific keywords. By repeating this process the user should eventually reach some desired information. Whenever a user selects a particular keyword the system will present him with a message pertaining to that keyword.

Tasks accomplished include implementing a menu driven system together with a keyword command system and allowing use of a simplified menu representation scheme. Testing might evaluate the effectiveness of these changes. One comparison involves single keying versus double keying versus menu driven user interfaces. Another comparison concerns the effects of command

completion and various methods of handling user errors. Keying issues, such as the association of function to the keys, and the effects of feedback messages could also be studied.

Acknowledgements

I would like to thank Edward A. Fox for being the Chairman of my committee. The time he spent on my behalf, often at unconvent hours is appreciated.

I am also grateful to Beverly A. Williges and Robert C. Williges for being on my committee as well as providing the framework for the project. Despite some personal difficulties, they managed to find the time to help me complete the project.

Table of Contents

List of Illustrations

1 Introduction

This project is part of a larger research effort to determine the human factors design parameters for a telephone based information system.[1]

> "The National Science Foundation project is an integrated research paradigm for information technology. Its purpose is to use sequential experimental design to determine the human factors design parameters for a telephone based information system.
>
> The telephone information system is a data retrieval system using the telephone keypad as input and synthesized speech as output. The system works by speaking lists of keywords and allowing the user to select appropriate ones. Selecting a keyword will give the user another list of more specific keywords. By selecting more specific keywords, the user reaches the information target. When the user has reached the information target, making a selection will present a message pertaining to the keyword."[2]

This project deals strictly with systems in which a telephone keypad is the only input device. Voice recognition provides what may be the best alternative to a telephone keypad. Voice recognition could also be used to supplement (as opposed to replace) the telephone keypad. Many systems have trouble recognizing continuous speech, but most work well when the input vocabulary is constrained. Some of the most common uses of this type of system are in the banking industry, voice mail, telephone answering machines, and telephone call re-direction applications.

---

[1] The larger project was funded by the National Science Foundation.

[2] This quote is from the general information section of the HCLSOFT computer program on VTHCL (a computer in the Human Computer Laboratory at Virginia Tech). The program was written by Cal Selig and the text was composed by Jay Merkle (who has served several roles in this project).

1.1 Original System

The original system (illustrated in Figure 1 on page 3), with software referred to as IF3.PAS, was written by Cal Selig and Jay Merkle. It is actually the third version of a system which is continuously being revised as the National Science Foundation Project precedes. It is strictly menu driven, which makes it easy for novice users since the command set is very small. However, experienced users seem to find it too slow since there are no shortcuts for navigating through the database.

Creating or modifying menu databases in the original system was difficult because many of the implementation details were not hidden from the menu designer. See Figure 2 on page 15 and "Appendix F. IF3.PAS Database" for samples of databases in the original format.

The original system ran on a VAX 11/750 with a speech synthesizer attached. This synthesizer communicates with a user's telephone by way of a standard telephone line.

1.1.1 The NSF Project

The NSF Project involves a study of methodologies for designing information systems.[3] The three areas of research are:

1. selection of the critical independent and dependent variables,

2. describing the interrelationships among the variables manipulated, and

[3] Information about the NSF Project was adapted from the proposal submitted to the NSF.

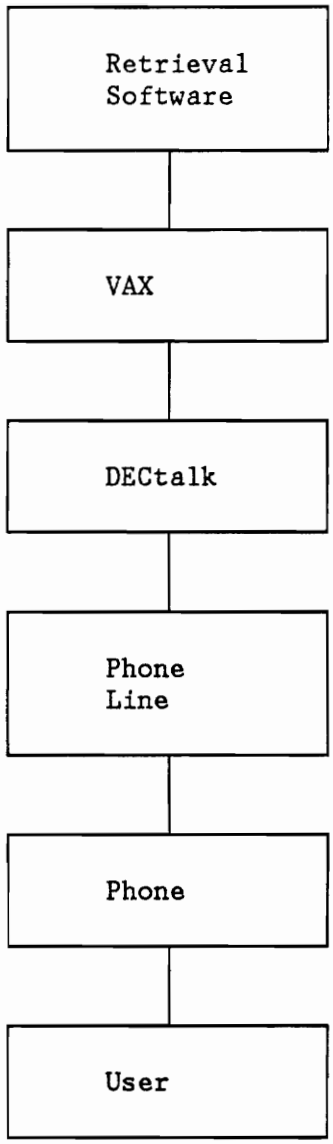


Figure 1. Overview of system.

3. finding the performance optimum.

The results of this project will include "the development of a systematic methodology for experimentation with complex systems" and "the collection of data to determine the factors of prime importance in optimizing telephone inquiry systems employing synthesized speech."

There are three general approaches used when researching information systems. They involve: direct observations of an operational system, analytical modeling, and experimental design methods. The third method is used in the NSF Project.

The major concerns when using experimental design are the expense of data collection and producing results which can be applied to the general case. Two methodologies will be used. They are: "computer simulation to generate data based on models selected to represent various states in the real world" and collecting data from actual experiments.

"In order to collect actual performance data a problem domain must be specified." The "design of a non-conventional communication interface, synthesized speech," is being used for the research.

There are two main research goals. The first is the "development of a systematic, multifactor research strategy for experimentation and empirical model building with complex information systems". The second goal is building "an empirically developed representation of the primary factors contributing to optimize the design of speech displays for telephone inquiry systems."

The project will investigate "several independent variables and dependent variables simultaneously." Pretesting of the various factors will be used to define the practical range of each factor. Testing several variables simultaneously will allow the interaction between the variables to be measured. "Research is needed to investigate the best procedure for specifying multivariate/multifactor relationships."

1.1.1.1 People Involved With the NSF Project

The principal investigators for the National Science Foundation Project (NSF) are Beverly Williges and Robert Williges. Calvin Selig is the System Manager and support programmer. Jay Merkle, Douglas Beaudet, and David Herlong are graduate students who are experimenting with the existing system. Robert Fainter, a computer scientist, assists in the development of the software.

1.1.2 Uses of Production System

A production version of this system would have numerous uses. For instance it could be used for dial-in information lines such as those supported by the **Communication Network Service** department at Virginia Tech.

Other uses include public service systems providing information on local events, medical information lines at the Montgomery County Hospital, and information services on Virginia Tech.

If modified to accept unconstrained input (supplied through double keying) the system could be used for data entry such as is required for people to register for classes.[4]

1.1.3 Hardware

Both the original and the revised systems were run on a VAX 11/750 known as VTHCL. This CPU was running VMS Version 4.7 and had a single DECtalk speech synthesizer (Version 2.0)

---

[4] The modified system is able to recognize constrained input using either single or double keying.

attached to it. A standard push button telephone was connected to the DECtalk unit by way of a normal telephone line.

1.1.4 Software

VAX Pascal is used to write both the original and the revised systems. Both systems use a few predefined DECtalk routines. The revised system uses some IF3.PAS routines but is mostly new code.

1.2 Goals of the Project

A quick demo of the original system is enough to point out several problems with it. First the system is (subjectively) very slow, and second there are no obvious short cuts for experienced users. Shneiderman states that "menu selection is very unappealing, and command language strategies, in spite of the greater memory demands on the users, become more attractive" when "response time is long and the display rate is slow" (Shneiderman, 1987). The slow response time is probably due mainly to the DECtalk unit since the load on the VAX's processor is negligible and very little I/O is required of the system. It is the author's belief that the response time of the original system cannot be improved significantly without upgrading to a faster speech synthesizer. The slow transfer speed is due to the inherently sequential nature of audio. One can speed up the speech rate of the speech synthesizer (within limits constrained by comprehension), but the process is still sequential. These factors point to the desirability of a command driven user interface.

The purpose of this project was to redesign the user interface of an existing information retrieval system to achieve the following goals:

- Simplify the process of constructing menus for the menu designer by reducing the amount of implementation specific information required. Formal testing could be performed to verify the results of the new menu representation scheme.
- Retain the original menu driven system. The revised system must be a superset of the original system.
- Provide a way for experienced people to use the system faster than the original menu system was capable of permitting. The keyword command facility is intended to provide this speed increase. Formal testing could be performed to verify the magnitude of the speed increase.
- Allow users to gradually and painlessly progress from the menu driven system to the command line system. The keyword command system is intended to allow the users to use the keyword command facility in parts of the database they are familiar with and the menu driven system in unfamiliar sections of the database. Formal testing of the effects of the keyword command facility is planned.

1.3 Overview of Report

The remainder of this report is divided into four major sections. First is the literature review. The second part covers the design of the project. The third part details some of the implementation decisions the author has made. The fourth part lists some suggestions for future testing and enhancement.

A glossary is provided at the end of the report, which defines some of the key terms used. Also included in the appendixes are sample data files and illustrations of the major data structures.

2 Literature Review

The literature overview has been broken down into four subsections which cover human computer interaction, algorithms and data structures, reference materials, and testing methodologies.

2.1 Human Computer Interaction

The author found Ben Shneiderman's recent book Designing the User Interface: Strategies for Effective Human-Computer Interaction (Shneiderman, 1987) helpful as a reference. It provided guidelines for many of the design decisions made during this project.

Since I have experience in designing and developing small and medium size applications in a business environment, I was able to draw upon my 'real world' experience to answer most of the design questions which arose during this project.

In Shneiderman's article Human Factors of Interactive Software (Shneiderman, 1983) he states that it is important to have a "graceful evolution" between the mode used by novice users and the mode used by experienced users. He believes that "methods for smoothing the transition from novice to intermittent knowledgeable to frequent expert could be studied." I have attempted to achieve a smooth transition as the user becomes more experienced with the telephone based information system.

As pointed out in Planning for Data Communications (Bingham and Davies, 1977), human errors remain the single most important cause of system failures. Due to the highly constrained input, handling unexpected keys was not a major problem. If the user ever pressed a key that was not valid in the current context, a message was spoken to the user. Should the user get stuck and fail to press any keys for an extended period of time, the system would hang up the telephone (unless the user entered pause first). A more graceful solution to this problem is desirable. Another potential problem would be if a user maliciously tried to tie up the system so as to prevent other users from being able to use the system. No work was done in this area.

There are two ways to use the keyword command facility. In the first method an experienced user enters a keyword to quickly move to a known place in the database.[5] The second use occurs when a person uses the keyword command facility as a query processor, hunting for information which may or may not exist in the database. A recent article by Furnas et al. suggests that in order to achieve an 80 percent hit rate when the keyword command facility is being used for searching, in most cases four to twenty synonyms will be required (Furnas, Landauer, Gomez, and Dumais, 1987).

2.2 Algorithms and Data Structures

Since the author believes that good selection of algorithms and data structures is necessary for proper system design, he used them where appropriate. The most important and unusual ones are discussed below.

---

[5] This method of operation is the one which the keyword command facility was originally designed to support.

Information on tries can be found in the book Fundamentals of Data Structures in Pascal (Horowitz and Sahni, 1983) by Ellis Horowitz and Sartaj Sahni. A trie is used for the auxiliary tree employed in this project to hold keywords and synonyms. A trie was chosen for the auxiliary tree since traversing the trie and performing command completion can be done in optimal time (constant time for each keystroke of input). There are two major types of tries -- compact and non-compact. Compact tries typically store the pointers to the children of a node on a dynamically allocated linked list. Non-compact tries store the children pointers in arrays. There would be some advantages to using a compact trie in production systems since compact tries would decrease memory requirements and allow larger databases to be used. However, in a research environment where memory is abundant and CPU time is limited, the reduced complexity and faster speed of non-compact tries make them a clear choice.

All of the algorithms used in this project were adapted by the author to emphasis run-time speed over all other parameters. This was done because the program operated in a CPU constrained environment. The time required to perform most transactions is constant with respect to the size of the menu or database.

If necessary the system can be slowed down by means of appropriately inserted delays (obviously if this program was too slow, increasing its speed would be much more difficult).

Lisa Fast, writing in Dialing a Name: Alphabetic Entry Through a Telephone Keypad, mentions an interesting method of double keying (an explanation of this method follows) (Fast and Ballantine, 1988). The application she was testing involved entering names via a telephone keypad. This method, named "Press Per Letter", uses the number of times a given key is pressed to determine which of the letters associated with that key the user wants. For example, pressing the '2' key once would mean 'A', pressing the '2' key twice would mean 'B', pressing the '2' key three times would mean 'C', and pressing the '2' key four times would mean 'A', and so on. This method works best when the user can receive some visual feedback of her input. This relates to the author's work because it uses a similar method to achieve a similar result.

2.3 Reference Materials

Specific information about the speech synthesizer used can be found in DECtalk DTC01 Programmer's Reference Manual (DECa, 1983) and DECtalk DTC01 Owner's Manual (DECb, 1983). These manuals were useful for determining the syntax of the calls to the DECtalk unit.

The source code of the original system was available and served as a model for the system I developed. Jay Merkle was the author of the original system.

2.4 System Testing

The article in chapter 11.3 of Handbook of Human Factors, (Williges, Williges, and Elkerton, 1987) by Robert Williges, Beverly Williges, and Jay Elkerton provided guidelines on formal testing procedures. This chapter also suggested some of the parameters for future testing.

The experiments that could be conducted using this system can be evaluated in one of three methods according to Evaluation of Dialog Systems by Reinhard Oppermann et al. (Oppermann et al., 1989). These methods are 1) Question and Answer, 2) Experimental Evaluation, and 3) Guideline-Oriented Evaluation.

The Question and Answer method can involve either written or oral questions. It can be conducted after the test, or it can involve a running commentary during the testing, when the subject is "thinking aloud". A possible variation of the latter method is "video confrontation" in which the user is filmed at work and "subsequently discussing and interpreting the session, using the video recording."

Experimental Evaluation methods include a wide variety of techniques. Some of these are:

1. Para-experimental studies - in which "the observer monitors the user, using a screen in parallel, assessing the user's behavior and possibly controlling the action by system interaction".

2. Classical experiments - involve observation of the user to a more or less "total" degree and more or less concealed from the user.

3. Benchmark tests - involve "comparing the way different systems perform certain standardized tasks. This type of experiment is concerned mainly with measuring objective data such as time, errors, learnability, and functionality. The results are not so much absolute numbers as they are merely a ranking in relation to other systems."

Guideline-Oriented Evaluation methods are conducted by having the system examined by an expert who uses checklists or a set of guidelines to evaluate the system instead of based on performing a test. This type of method can be used for either development or evaluation of a system.

The measurable human factors issues according to Ben Shneiderman are (Shneiderman, 1983):

- time to learn
- speed of performance
- rate of errors
- subjective satisfaction
- retention over time

This project provides a suitable testbed for measuring the above mentioned human factors issues.

3 Design

This section describes the rationale and details of the design of this system. Information about how the system was actually developed and enhanced can be found in Chapter 4.

3.1 Evolution of the External Representation of the Menu

Both the original and the revised system store the menu (shown under "Appendix G. New Representation of IF3.PAS Database") in an external file, to facilitate change and repurposing of the system. Doing this allows the domain of the system to be changed by simply loading a different file. For the author's work to be useful it must have the ability to handle different menu designs. Other people involved with the NSF Project are planning to study the differences in user friendliness between deep and broad menus and other design parameters of user interfaces.[6] Therefore if this program could not adapt to different menu designs it would not be useful in these future studies.

3.1.1 The Original Representation

In the original scheme, for each node in the menu there is one line in the input file which stores the address of its parent node, the address of its right sibling, the address of its child node, and either

---

[6] The study involves more than 100 different parameters.

a keyword or a data file name. (Addresses in this context are Pascal array indexes.) Note that a keyword and a file name cannot be stored in the same node, because of an implementation restriction for which the reasoning is unknown to the writer. All addresses are array indexes of the nodes and must be calculated manually. This was usually done by using a large piece of paper to graph a representation of the logical structure of the menu. The maximum number of nodes must be calculated at compile time and cannot be adjusted automatically according to the size of the input file. Figure 2 on page 15 shows the original database file for the menu shown under "Appendix B. The Main Tree for the Sample Menu" on page 50.

This representation, while simplifying the programmer's job, adds a significant burden to anyone who wants to change the menu design.

3.1.2 The First Menu Redesign

The author developed a menu representation language which frees the menu designer from having to calculate the addresses of parents, siblings, and children. Instead these are calculated by the program when the input file is read in. Storage for the menu representation is allocated dynamically instead of at compile time. Thus no practical limit on the number of nodes in the menu exists.

Figure 3 on page 16 shows the sample database in the style of the first redesign. Notice that the menu designer does not have to calculate array indexes or the number of nodes in the menu. Both of these calculations are done automatically by the system. The ".df" tag is used to denote a data file. The ".h0" tag is used to provide an identifying name for the menu database.[7] The ".h1" through ".h9" tags are used to denote the level of the menu that is being described. The routines handling the tags, data file names, and keywords are not case sensitive.

[7] The author felt that the ".h0" name for this keyword was the most consistent with the other tags.

Database Records

```
Parent
   Sibling
        Child
            Keyword or data file name
-----------------------------------
  0    2    4 movies
  0    3    5 special
  0   -1    6 sports
  1    0    0 movies.inf
  2    0    0 special.inf
  3    7    9 next
  3    8   10 level
  3   -6   11 novies
  6    0    0 atnext.inf
  7    0    0 atlevel.inf
  8    0    0 novies.inf
```

Figure 2. A database in the original format.

The ending tags, ".eh0" etc., were chosen to be consistent with SCRIPT/VS. The number of the ".h0" etc. tags increases by one for each level further down the menu.

The syntax of a data file is shown in Figure 4 on page 17.

Note that a given keyword and/or data file name can appear in more than one place in the menu if the menu designer wished to do so.

The reason ".df" and the data file name are on the same line is that there is only a single data file name for each ".df". This does not hold true for the ".h1" etc. tags, which is why they are on different lines. It is necessary to use ".eh0" etc. tags (or a similar method) to indicate the ending of a menu.

The author is quite familiar with SCRIPT/VS, GML, and SGML. The influences of some of these may be apparent in this design.

```
.h0
sample menu
.h1
movies
.df movies.inf
special
.df special.inf
sports
.h2
next
.df atnext.inf
level
.df atlevel.inf
novies
.df novies.inf
.eh2
.eh1
.eh0
```

Figure 3. A database in the first redesign format.

3.1.3 The Second Menu Scheme

While the above menu representation scheme frees the user from having to calculate pointer values, it does not provide a representation of the menu whereby the structure is obvious when viewing the data file. The author designed a second new menu representation which retained the pointerless representation of the first menu representation and adds a structural representation of the menu. The logical structure of a menu is much more apparent when viewing a menu in the new representation than when viewing the same menu in either of the first two menu representation systems. Testing to determine the magnitude of the improvement of the second redesign menu over the original design menu has been suggested. Figure 5 on page 18 shows a sample of the first menu using the second representation style.

A valid menu representation has the following traits.

- The database identifying name is stored flush left on the first line.

```
<menu file> ::= <h0 tag> <database name> <h1 part> <eh0>
<h0 tag> ::= .h0 <cr>
<h1 part> ::= <h1> <data> <eh1>
<database name> ::= character string <cr>
<h1> ::= .h1 <cr>
<eh1> ::= .eh1 <cr>
<eh0> ::= .eh0 <cr> <eof>
<eof> ::= the end of file character
<data> ::= {<leaf node> | <internal node>} [<data>]
<leaf node> ::= <keyword> <cr> .df <data file name> <cr>
<internal node> ::= <keyword> <cr> <heading> <data> <ehead>
<keyword> ::= the menu choice for that node
<data file name> ::= any valid file name
<heading> ::= {.h2 | .h3 | .h4 ... } <cr>
[Note that the number must be one greater than the current
level number.]
<ehead> ::= {.eh2 | .eh3 | .eh4 ... } <cr>
[Note that the number must be one less than the current
level number.]
<cr> ::= the end of line character(s)
```

Figure 4. Syntax of first redesign for the data file.

- The last line is also flush left and serves as a comment (text is ignored).

- The level of the node in the menu times constant 'num_blanks_per_level' equals the number of blanks which begin the line.

- The keyword follows.

- Synonyms, if any, follow the keyword. Synonyms are separated from each other and the keywords by a special symbol which is currently "|". Synonyms are not in the example since they were not allowed in the original representation (Jay Merkle's work).

Synonyms can be entered while in the keyword command mode[8] instead of a keyword. When the system recognizes a synonym the system maps the synonym to its corresponding node. Next the

[8] The user is in this mode when she enters the first keystroke which is not valid in the menu mode.

```
Notice that spaces can follow the '(' if desired.

sample menu
  movies ( movies.inf
  special (special.inf
  sports
    next (atnext.inf
    level (atlevel.inf
    novies (novies.inf
end
```

Figure 5. A database in the second redesign format.

system issues a message to the user informing him that the word he entered was a synonym and telling him which keyword the synonym has been mapped to.

If a line represents a leaf node, the keyword and any synonyms present will be followed by a left parenthesis and a data file name for the node. Lines which do not include a left parenthesis and a data file name are used to represent interior nodes of the menu. After the data file name (if it exists) the menu designer can add a right parenthesis and an integer representing the node's priority. This priority field will be discussed in greater depth under "4.4.3 The Priority Field" on page 33. The syntax of a data file is shown in Figure 6 on page 20.

By varying the value of the constant 'num_blanks_per_level' the system can adapt itself to input files which represent a change in level by any fixed number of blanks. Currently the system requires exactly two blanks for each level.

At one time the designer considered relaxing the fixed number of blanks per level restriction and allowing the user to indicate an increase in menu level by increasing the number of blanks by one or more. While this approach allows representation of the transition to lower levels of the menu, it is cumbersome when representing the change from a lower level on the menu to a higher level on the menu since it is possible and likely that a change of several menu levels could occur at once. Another problem is that without a consistent number of spaces before a given menu level it is

difficult for the menu designer to tell if two non-adjacent nodes are at the same level. For these reasons the author decided upon a 'fixed number of blanks per level' rule. The use of tabs was discounted because the author's experience has led him to believe that tabs should not be used in input file since they tend to cause more problems then they solve. This is because different programming languages, editors, and file systems store, process, and interpret tabs differently.

3.2 The Help System

When a user of the original system reached the end of a menu (at any point in the tree of menus) without selecting one of the menu items the system would deliver the help message and wrap around to the beginning of the same menu.[9] If the user failed to make a choice during the second consecutive pass down a menu the system would hang up the phone. While this behavior may seem strange there is a perfectly logical explanation for it. The speech synthesizer currently in use is unable to detect when the user hangs up the phone. Since the designer does not want the program to repeat a menu indefinitely after a user hangs up, the system simply hangs up after two passes. For a production system a useful precaution might be to increase the number of passes before the system hangs up.

The original system assumes the last (right most) node in each menu is marked by setting its right sibling pointer to the negative of the value of the first node in that menu. The original system detects the negative pointer as it traverses the menu and delivers the help message. The menu designer must manually insert these negative pointers.

In the new system a help node is actually linked in during the program's initialization time (when the menu representation is being built in memory). This requires slightly more memory but

---

[9] Please see the third paragraph of this section for a discussion of how the modified help system works.

```
The syntax of a data file is:

<menu file> ::= <database name> <menu rows> "end" <cr>
<database name> ::= character string <cr>
<menu rows> ::= <menu row> <cr> [<menu rows>]
<cr> ::= the end of line characters
<menu row> ::= <blanks> <keyword> [<synonyms>]
               ["(" <data file name>] [")" <priority>]
<synonyms> ::= "|" <keyword> [<synonyms>]
<keyword> ::= the menu choice for that node or a synonym
<data file name> ::= any valid file name
<priority> ::= priority number
```

Figure 6. Syntax of second redesign for data files.

simplifies the run-time traversal of the menu. In the modified system the menu designer does not have to be concerned with the help nodes since the system will insert them automatically at the end of each menu level. The help information, which can easily be changed, is shown under "Appendix H. The Test Data" section "Help.inf" on page 60.

In the introductory message, both the original and the revised system automatically ask the user if she wants instructions on how to use the system. The instructions for the single keying system (one of the two user interfaces the author designed) are shown under "Appendix H. The Test Data" section "Instruct.inf" on page 59.

3.3 The Menu Driven System

The menu driven part of the system retains all the important features of the original system (i.e., IF3.PAS). The implementer has relabeled some of the keys (see "4.1 Layout of Keypad" on page 26) but all of the original functions remain. Some minor improvements to the menu driven system

have been made such as adding information messages when 'backup' or 'restart' are invoked and changing the text of the help messages.[10]

3.3.1 The Main Tree Data Structure

The main data structure of the system is called the 'main tree'. This structure, a generalized linked list, represents the menu.[11] It is basically a superset of the information stored in the input files of the original system. "Appendix B. The Main Tree for the Sample Menu" shows a graphical illustration of the main tree for the sample database shown in Figure 2 on page 15, Figure 3 on page 16, and Figure 5 on page 18.

Each node of the main tree stores the information shown in Figure 7 on page 22.

3.4 The Keyword Command Mode

The original menu driven system worked well for small menu systems but forced the user to spend significant amounts of time to reach the leaf nodes in larger menus. Its slow speed became very annoying and frustrating to frequent users. The main reason for this slow speed is that audio, by its very nature, is a sequential medium. This means that each element of a menu must be presented sequentially, followed by a pause so that the user can respond if she chooses to do so. Contrast this with video display terminals which can show the user a complete screen full of information virtually instantaneously.

---

[10] These feedback messages can be switched on and off so that their effect can be accurately measured.

[11] A generalized list is a finite sequence of elements where each element is either an atom or a list (Horowitz and Sahni, 1983).

Field	*Description*
data	the keyword
priority	the priority of the node. See "4.4.3 The Priority Field" on page 33 for more information.
depth	the depth of the node in the menu. The first level menu has a depth of one and the second has a depth of two, etc.
parent	a pointer to the parent menu. This pointer is set to nil for nodes on the first level menu.
leftmost_sibling	a pointer to the first node in a menu. For the leftmost node, this pointer points to itself.
sibling	a pointer to the right sibling of a node. For the rightmost node of a menu this pointer points back to the first node of the same menu.
child	a pointer to more specific menus of the menu representation. It is set to nil for leaf nodes.
aux_link	a pointer to the next node in the auxiliary tree. It is set to nil for the last node in a linked list.
file_name	a VMS file name (found if and only if the child pointer is nil)

Figure 7. Information stored in main tree data structure.

A major part of this project involved designing a way for frequent users to navigate through large menus without spending an inordinate amount of time. One solution to this problem is a keyword command interface.

3.4.1 Command Completion

Command completion is especially important in this application because of the slow nature of the user's input. The author's informal tests have shown that telephone keypad input is far slower than using a standard typewriter keyboard. On small menus command completion can cut the number of keystrokes required by a factor of two to three. On large menus the reduction in the number of keystrokes required should be lower than the reduction when using small menus; nevertheless, it will still be worthwhile.

3.5 The Interface Between the Menu and Command Modes

In order to achieve the goal of making this system as usable as possible the author decided that the transition between the menu driven mode and the keyword command driven mode should be as smooth as possible. Having different modes with difficult transitions between these modes makes programs much harder for inexperienced users to learn. The implementation of this is described in "4.6.1 The Interface Between the Menu and Command Modes" on page 36.

3.6 Handling Q, Z, and Space

There are many possible mappings between the alphabet and the telephone keypad. Since 24 of the 26 letters in the Roman alphabet are already associated with certain keys, it makes sense to utilize this mapping. The most difficult part is handling the remaining two letters.

When using a telephone keypad to enter the letters of the alphabet an interesting problem arises: "How should the user enter the characters Q, Z, and space?" The standard telephone keypad has three letters associated with the keys two through nine. The characters Q, Z, and space are not associated with any keys.

There are two general types of solutions to this problem:

- Associate all twenty seven characters with keys on the telephone keypad. Assign more than one character to some or all of the keys.

- Require more than one keystroke for some or all of the characters.

Since this application is required to recognize only a constrained set of words the author chose to associate four characters with each of three number keys. This approach makes it more difficult for the system to recognize the commands the user enters (in the keyword command mode) since three of the keys are now associated with four characters each instead of three. However, this approach eases the burden for the user when working with small and medium size databases (which was the goal of this whole project) since she does not have to use double keying (a style of interaction requiring two key-presses for each letter entered). During testing I provided a sheet of paper containing a graphical representation of the keyboard. The author's intentions are for this aid, shown in Figure 8 on page 25, or a smaller version of it, to be provided to users of the production system.

There is one major potential problem with the approach that is used in this system -- if the system is expanded sometime in the future it will be hard to incorporate unconstrained input.[12] One method of implementing unconstrained input is double keying, which is the process of having the user press two keys for each letter or selection she wants to enter. On conventional telephones double keying often involves using the '*', '#', and sometimes the '0' key since those three are distinguished by position. "Appendix D. Key Assignments for Double Keying" on page 52 shows one possible method of associating keystrokes with letters. A blank has been included so that the user can enter phrases such as 'tennis court' in addition to single words.

When some of the characters have four letters associated with them[13] the implementation of double keying is complicated since there are more than two or three possible second keystrokes. One solution, which still retains fours letters for some of the keys, is to define a fourth key to supplement the *, 0, and # keys. The number one key would be a likely choice since it has no letters associated with it.

---

[12] Unconstrained input refers to the situation when the system is required to accept and process an infinite number of valid input strings (i.e. not limited to a finite number of keywords). An example of this is a system which allows people to enter their names.

[13] As shown in "Appendix D. Key Assignments for Double Keying" on page 52, the seven, eight, and nine keys each have four associated characters.

1 select repeat	2 ABC	3 DEF
4 GHI	5 JKL	6 MNO
7 PQRS	8 TUVb	9 WXYZ
* pause	0 restart	# backup

Figure 8. A sample keyboard template.

The author expanded the program so that by changing the value of a few constants a switch can be made from single to doubling keying. See source file "command.pas" for documentation on making this change. The *, 0, #, and 1 keys are presently used for the second keystroke when double keying is in effect. (The order of the two keystrokes could be reversed if desired.)

Figure 8 shows the key definitions that are in effect when double keying is in use. Notice that the key definitions are still the same as those used when single keying is in effect.

4 Implementation

This section describes how the enhanced system was implemented, building upon the design discussed in Chapter 3.

4.1 Layout of Keypad

The users of the original system were concerned with only four keys on the telephone keypad. These were:

- # -- select - make a choice from a menu
- * -- backup - go up one level on the menu hierarchy
- 0 -- restart - go up to the top of the menu hierarchy
- 8 -- pause - pause, waiting until the user presses pause again

The improved system makes use of all twelve keys. The keys have the following meanings:

- 1 -- select - make a choice from a menu; also serves as the repeat key
- # -- backup - go up one level on the menu hierarchy
- 0 -- restart - go up to the top of the menu hierarchy
- * -- pause - pause, waiting until the user presses pause again
- 2..9 -- letters - for entering command names and keywords

The eight keys between two and nine are each associated with three letters of the alphabet. Three exceptions are the seven, eight, and nine keys which each have four characters associated with them. While the characters 'Q', ' ', and 'Z' are not represented on the standard telephone keypad, the system acts as if they are marked on the seven, eight, and nine keys respectively. The author designed this scheme because it was the most intuitive to him. As far as the author knows, no one else uses this scheme.

4.2 Data Structures

Since this is an interactive system, response time is very important. Efficient data structures must be used to insure that the program can respond quickly enough.[14] The three major data structures are the auxiliary tree, the main tree, and a set of linked lists connecting the auxiliary tree to the main tree. Due to the algorithms and data structures used in this system, the author believes that computer processing is not the limiting factor in overall system performance.

4.2.1 The Auxiliary Tree Data Structure

The keywords and synonyms are stored in a trie called the Auxiliary Tree. Each node of the trie has eight children numbered two through nine (when single keying is being used).[15] As the input file is read the keywords and synonyms are inserted into the trie. After all of the input file has been read a command completion algorithm is applied to the trie. This routine effectively performs the

---

[14] Faster processors have rarely been able to solve performance problems that are caused by bad program design.

[15] When double keying is being used each node has 27 children numbered zero through 26. This has not caused any problems, regarding memory limits, with any of the menus the author tried. In some circumstances using a sparse trie could reduce the amount of memory used.

command completion at initialization time -- instead of at run-time as one might suspect. Each leaf node of the trie stores the full command name, a pointer to the head of a linked list (see the next section), and the number of nodes in that linked list. "Appendix C. The Auxiliary Tree for the Sample Menu" on page 51 shows what the auxiliary tree for the sample menu looks like before the command completion operation is performed. Only the links marked with a '*' on the drawing and their associated nodes remain after command completion.

Each node of the auxiliary tree stores the following information:

Field	*Description*
full_name	a keyword or synonym.
main_tree	a pointer to a node of the main tree. This is actually the head of one of the second set of linked lists which are implemented on the main tree nodes. See the next subsection for more information on the second set of linked lists.
num_occurrences	the number of nodes in the linked list pointed to by the above field.
child	an array of eight[16] pointers to the child nodes of this node. The absence of a particular child node is indicated by a nil pointer.
path	an array of eight[16] integers. It stores the number of keywords or synonyms contained in each subtree of this node.[17]

See the source file "init.def" for more information on this data structure.

---

[16] 27 when double keying is being used

[17] It has been noted that this array could be moved down one level in the auxiliary tree and replaced with a single scalar. This method would save a few bytes of memory; this could be a desirable change in a memory constrained environment.

4.2.2 Links Between the Auxiliary Tree and the Main Tree

The third major data structure in the system connects the other two major data structures (which are the auxiliary tree and the main tree) together. It is used to move from the auxiliary tree to the main tree. It consists of a set of singly linked lists. The leaf nodes of the auxiliary tree point to the heads of these linked lists. The nodes of these linked lists are the same as the nodes used in the main tree. Two completely separate sets[18] of links are defined on a single set of nodes.[19] See the source file "init.def" for more information on this data structure.

Each linked list is only one node long except in those cases where there is more than one node in the main tree with the same keyword (or synonym). In these cases the linked list connects all the main tree nodes which have the same keyword. For example, if the keyword 'theater' is in the database five times (as in "Appendix G. New Representation of IF3.PAS Database" on page 57) then there are five nodes on the linked list for 'theater'.

Please see "Appendix B. The Main Tree for the Sample Menu" on page 50 and "Appendix C. The Auxiliary Tree for the Sample Menu" on page 51 for illustrations of these data structures for the sample menu.

4.3 Command Completion

At program initialization time the complete input file is read and the auxiliary tree is created. As soon as this tree is created (and before the telephone can be answered) the tree is compressed to

---

[18] See "3.1.1 The Original Representation" on page 13 and "3.1.3 The Second Menu Scheme" on page 16 for information about both of these data structures.

[19] Doing it this way greatly simplifies keeping the two data structures synchronized.

remove nodes which are not needed because of command completion. See "Appendix C. The Auxiliary Tree for the Sample Menu" on page 51 for an illustration showing an auxiliary tree before and after command completion is performed.

The algorithm for this compression follows. When the algorithm begins, the variable 'current' points to the root of the auxiliary tree.

```
procedure condense_tree (current)

count the number of children of the current node

set flag to false
if the number of children of the current node is one then begin
   if there is only one leaf node below the current node then
      erase all of the subtree except the root(20)
   else
      set flag to true
   end
else
   set flag to true
end of first if

if flag is true then begin
   for each child of the current node which exists do
     if there is only one keyword which uses that path(21) then
        erase all of the subtree except the first node(20)
     else
        call condense_tree (current->child)(22)
   end
end of third if

end procedure condense_tree
```

---

[20] "Erase" in this context means both a logical and physical deletion of the nodes in question.

[21] "Path" is the set of nodes and associated links which represent a keyword or synonym.

[22] It has been noted that iteration could have been used instead of recursion -- the author believes that the use of recursion produces a more maintainable system in this situation. It has also been noted that there are other methods of compressing the auxiliary tree -- the author has used the method which is most intuitive to him; others may prefer (and should feel free) to code differently.

4.4 Ambiguous Commands

Another type of problem encountered with the keyword command system is that the menu designer may want to include keywords or synonyms in the menu which are not unique (i.e., they appear more than once) or whose keystroke representation is not unique (e.g., movies and novies when single keying is used). The following sections describe how the system handles these problems.

The system begins by making a linked list of all nodes whose keystroke pattern matches the keystrokes the user entered.

4.4.1 Nodes With Different Names But the Same Keystrokes

The first type of ambiguity the system resolves is when two or more keywords have the same keystroke patterns.[23] An example of this is the two strings 'movies' and 'novies' both of which are entered with the six - six - eight - four - three - seven pattern of keystrokes. It is also possible that some of these ambiguous keywords will be found in more than one place in the menu system. When this occurs, the system first finds the correct keyword and then finds the correct node, as follows.

The system sorts the linked list in order based on the values in the priority fields (see "4.4.3 The Priority Field" on page 33) of the nodes. The system then says the keywords (in descending order based on their priority) allowing the user to choose the keyword he intended. (If the keyword the user intended is not on the list the user can press 'backup', 'restart', or no key at all. The system will then leave the keyword command mode and return to the menu mode.) The menu must be repeated several times before the system assumes that the user is not going to enter a choice. If the

[23] This type of ambiguity occurs only when single keying is used.

user does select a keyword, the system will modify the linked list of possible nodes so that it contains only nodes with that keyword. In many cases the modified linked list will then contain only one node and the system will have effectively resolved the ambiguity. If the linked list contains more than one node, however, the system follows the procedure described in the following section.

4.4.2 Several Nodes With the Same Name

When several nodes with the same name exist in the same menu the system attempts to determine which node the user really wants to visit. To do this the system first finds the deepest menu which contains all of the possible nodes in its subtrees. After the system finds this menu depth it displays parts of this menu to the user so that she can choose which keyword is the closest to the intended context.[24] Only the nodes of the menu which are the root of trees containing the possible nodes are presented to the user. These nodes are presented in order of their priority field. If the user does not rapidly choose a keyword from the list of keywords given, the menu will be presented several times. After several passes[25] or when the user presses 'backup' or 'restart' the system returns to the menu mode.

After the user chooses a keyword from the menu that has been presented, the system removes from the list of possible nodes all the nodes which are not in the tree whose root is the selected node. At least one node will be removed from the list of possible nodes by this action. At most all but one node will be removed from the list.

At this point the list is checked to see how many nodes it contains. If the list contains only one node, then the ambiguity has been resolved. If the list contains more than one node, this process

---

[24] For example two nodes could have the name 'cost', where one was associated with 'movies' and the other with 'cars'.

[25] The number of passes is determined by a constant defined in the program header. This is currently set to two.

described in the above two paragraphs will be repeated until only one node remains on the list. The process begins again by finding a node which is the lowest node in the menu system which contains all the possible nodes in its subtrees. Note that each time this process is repeated, the level of this common subtree node must be at least one level deeper.

Thus, in the worst case this process of resolving ambiguities will terminate in S steps, where S is the smaller of the following two numbers. The correctness of the algorithm is obvious if the above two paragraphs are considered.

- the number of levels in the system
- the number of nodes with the same name minus one

Since for deep menus this process could take a significant amount of time, it is recommended that when using deep menus the menu designer try to avoid using multiple nodes with the same keywords located in the lower levels of the menu.

4.4.3 The Priority Field

During the process of resolving ambiguous commands the system will need to present one or more menus to the user to obtain the additional information needed to resolve the command. In order to speed the interaction the nodes on these clarifying menus are presented in presumed order of the most likely to the least likely. This ordering priority is read from the input file.

There is not a strict ordering. Two or more nodes can have the same priority. All nodes which are not assigned a priority by the menu designer are given the same priority, whose value is determined by the Pascal constant default_priority. If the menu designer chooses not to include priorities in the menu file the system assigns the default priority to all the nodes, thereby effectively bypassing the priority mechanism.

4.5 System Related Issues

The system contains built in debugging facilities, and produces a log file. These will be helpful to anyone who needs to modify the system or chase down a bug.

4.5.1 Built in Debugging Facilities

Included with the system is a set of debugging routines, described briefly below.

Name	*Description*
dump_main_node	displays the contents of all the fields in every node of the main tree. Pointers may not be displayable with some compilers since the author has not written Pascal I/O procedures (he simply uses the ones provided with the compiler). No attempt was made to have the code dereference pointers since the author does not believe that dereferencing would be preferable to the raw data. (Dereferencing is best left to source level debuggers.)
dump_aux_node	displays the contents of all the fields in every node of the auxiliary tree. Pointers may not be displayable with some compilers since the author has not written Pascal I/O procedures (he simply uses the ones provided with the compiler). No attempt was made to have the code dereference pointers since the author does not believe that dereferencing would be preferable to the raw data. (Dereferencing is best left to source level debuggers.)
trace	displays a preorder trace of the main tree.
trace_aux	displays a preorder trace of the auxiliary tree.
display_full_name	displays all the keywords and synonyms stored in the auxiliary tree.

dump_linked_list displays selected fields of each node of a linked list. The fields to be displayed are selected by adding and deleting comments from this procedure. The output of this procedure will be easily understood by anyone who is familiar enough with the code to attempt to debug it. Interested readers are referred to the code if they desire more details.

4.5.2 The Log File

The system creates a log file which retains information such as the time the phone was answered and which parts of the menu representation were actually visited by the user. One could always add additional write statements to the program if the need for a special type of information arises. A sample of a log file is shown in "Appendix E. A Sample Log File" on page 53. (Currently most of the information in the log file relates to the testing of the program and not to the testing of the interface design -- it is not intended to be understood by anyone other than the programmer who is maintaining the code.) The logging cannot be turned off. Changing what information is logged must be done by the programmer who is maintaining the code (not by the person conducting tests).

4.6 User Interaction

Several important issues relating to user interaction arose during the implementation of the system. These included the interface between the menu and command modes, the user's response time, and handling the user's errors.

4.6.1 The Interface Between the Menu and Command Modes

Williges suggests that consistency throughout a system interface is very important (Williges, Williges, and Elkerton, 1987). Therefore one of the goals of this project was to have a smooth transition between the menu driven mode and the keyword command mode. In order to achieve this goal the implementer assigned the functions unique to the keyword command interface to different keys than those used for functions unique to the menu driven interface. The functions that are common to both modes share the same key assignments.

The function assignments that the programmer used allow the user to enter the keyword command mode simply by starting to type in a command -- no special keystrokes are required to switch into the keyword command mode.[26]

Leaving the command mode is accomplished in one of two ways:

- If the user does not press any keys after completing a command, she will be returned to the menu driven mode (the time that the system will wait is determined by a constant). Her position in the menu will be determined by the last command she entered. This will allow her to use the keyword command mode to navigate to a position deep in the menu hierarchy and then use the menu driven mode to search unfamiliar menus. After she leaves the command mode the revised system behaves in a manner similar to the original system.
- If the user wishes to return to the menu driven mode without completing his command he can press 'select' to return directly to the place in the menu where he was before he started typing in his uncompleted command. The user can press 'restart' or 'backup' if he wishes to return to the top of the menu system or the previous menu, respectively.

---

[26] There are actually two distinct modes; however, the transition between them is almost transparent to the user.

4.6.2 The User's Response Time

The system will wait a certain length of time after speaking so that the user has a chance to respond (if she wishes to do so). These delays and other parameters can be varied by changing the following constants:

respon_lat: the system response time

rate: the speech rate of the DECtalk unit

rings: the number of rings before the system answers the phone

max_cycles: the number of times a menu will be repeated before the system assumes the user has hung up

max_cycles_command: the number of times the 'please continue entering your command or press one to return to the menu mode' message will be repeated before the system assumes that the user has hung up (command mode only)

max_no_responses: controls the timing between the 'please continue entering your command or press one to return to the menu mode' messages (command mode only)

max_cycles_pause: the number of times the pause message will be repeated before the system decides the user has hung up

pause_time: controls the timing between the 'conversation on pause'[27] messages

[27] This is a message designed to reassure the user that the system is still working correctly.

max_cycles_display: the number of times the display menu will be repeated (used in the menu mode when the user's keystrokes were mapped to several nodes with different keywords)

Preliminary tests suggest that most of these values will vary widely depending on the application. For example, when the programmer was testing the implementation he found that allowing very short response times was the most effective way to test the system. When demonstrating the system to someone who had never used it and who was interested in knowing the methodology behind it very long response times were necessary. This was especially true for the pause command so that the system would not hang up too quickly in situations such as the user being distracted by a knock on the door before she could complete the interaction. For production use (which has been simulated by the author's test subjects) one would expect to use intermediate values for parameter settings.

The above relates to "User's" response time in that it discusses how different users found different response time settings to be advantageous for different applications.

4.6.3 Handling User Errors

The author's preliminary tests indicate that users are prone to make more mistakes while using the command line mode than they are when using the menu driven mode. The three most common mistakes appear to be pressing the wrong key, spelling a word wrong, and trying to enter a word which is not a keyword. When the system is in the keyword command mode it traverses the auxiliary tree as the user enters the command one 'letter' at a time. Immediately after the user enters a 'letter' which is not part of a known keyword the system discards the 'letter' and informs the user that the keystroke was not used. The system then provides the user with a list of the valid keystrokes entered so far. The user can continue entering the command as if the last keystroke

never happened or she can return to the menu mode with a single keystroke (i.e., select, backup, or restart).

4.6.4 The Pause Command

When the user presses the pause key the system goes into the pause mode and waits for her to press the pause key again. The original system was not able to detect if the user hung up the telephone while the system was in the pause mode; the new system can simulate the ability to detect a hang up in the pause mode by placing limits on how long the system stays in that mode. See "4.6.2 The User's Response Time" on page 37 for information on 'pause_time' and 'max_cycles_pause' which can be varied to change this limit.

The system also provides a second fix regarding use of the pause command. The problem in this case is that when in pause mode in the original system, if the user pressed any other key besides the pause key, that key would be ignored and no information message would be given. The author added a message to the new system which reminds the user that she must press the pause key to exit the pause mode.[28]

4.7 Pilot Testing

As mentioned earlier the purpose of this project was to create a testbed that would be useful to the researchers working on the NSF project studying user interfaces. In an effort to verify that this system would be usable for its intended purpose, the author asked several people to try out the

[28] This message is issued immediately after the system finds the invalid keystroke.

system while he observed their reactions to the system. These people had varying degrees of computer experience. The author also created several menu representations -- one of which was a direct conversion of the main testing menu used in IF3.PAS. The IF3.PAS menu is shown in "Appendix F. IF3.PAS Database" on page 54 along with its new representation in "Appendix G. New Representation of IF3.PAS Database" on page 57. "Gym.inf" on page 61 is an example of the data files used in this system.

5 Future Work

Now that informal testing has been completed, more formal testing should be undertaken to prove that the system accomplishes the goals of this project. The first part of this section details parameters which the author believes would make good candidates for further study. The second part covers suggested enhancements to the system.

5.1 Future Testing

Among the areas that should be tested in the future are the keyword command facility, various keying issues, feedback issues, and user modeling issues.

5.1.1 Keyword Command Facility

The following is a list of tests to perform to explore settings of various parameters that relate to the keyword command facilities.

1. Single keying versus double keying versus menu use. Experimentation to determine how these interaction modes will effect the user is suggested. Useful metrics for comparisons include: how long it takes the user to find his information target, his satisfaction with the system, and his error rate.

2. Effects of command completion. Experimentation to determine what effect command completion has on user satisfaction, error rates, and speed is important.

3. How to best handle **valid** keystrokes in keyword command mode after an invalid keystroke -- should the valid keystrokes be discarded? Should the valid keystrokes be displayed?

4. Effects of priority and ordering of nodes. Experimentation to determine if this makes the system more useable is suggested. If significant improvements are not discovered, than this feature should be removed since it adds greatly to the complexity of the system (from the view of the programmer) and therefore makes parts of the system difficult to maintain and enhance.

5. Is the keyword command facility used for fast access to known information or as a method of searching? Experimentation to determine **why** people use the keyword command facility is desirable.

5.1.2 Keying Issues

The following is a list of issues relating to keying.

1. Assignment of keys for 'select', 'restart', 'backup', and 'pause'.

2. Assignment of keys for 'Q', 'Z', and space.

3. Echoing letters in double keying. Testing with and without echoing need to be done in order to determine the effects of echoing in this environment. Echoing, if it is done, would occur after each pair of keystrokes.

5.1.3 Feedback Issues

The following is a list of issues relating to feedback.

1. What to do when the user presses an invalid second key when using double keying.

2. Effect of more specific instructions for procedure Identical_keywords (the procedure which resolves ambiguous commands -- see "Appendix C. The Auxiliary Tree for the Sample Menu" on page 51) such as saying 'Press 1 for dune' or 'Press 1 for Gone with the wind' instead of 'Did you mean that as in dune' or 'gone with the wind'.[29]

3. Effect of saying the lower level menu keyword (for example 'Show times') before presenting the top level menu in cases where 'Show times' is in the database more than once.

4. Effect of feedback messages[30]:

   a. for 'backup', 'restart', and 'select'.

   b. when using keyword command mode to jump to the middle of the database (listing of the nodes traversed).

5. Length of feedback messages.

The above list relates to future work because it is a list of items which could be tested using the author's system.

---

[29] The author believes that using the same key for each of the menu choices is the correct approach here since it is consistent with the manner menus are presented and choices selected in the rest of the program. Furthermore, the same 'select' key should be used throughout the system.

[30] Feedback messages are messages that are spoken by the system immediately after the key is pressed.

5.1.4 User Modeling

The following is a list of issues that relate to the possibility of adapting to users.

1. Adjusting priority.

2. Adjusting the ordering of the menus.

Studying the above issues would tell developers whether or not it is useful and cost effective to add these complicated features to software.

5.1.5 Other Issues

The following is a list of other variables which do not relate to the keyword command facility.

1. Effect of system response time. Previous studies have shown that when the system response time is either too short or too long performance is degraded.

2. Effects of varying the speech rate on user satisfaction, error rate, and length of time needed to find the information target.

3. Best way to leave the pause mode. Do we require, or even allow, the user to leave the pause mode by hitting the pause key a second time?

4. Effects of a wallet guide (database map). Does it actually improve performance? If so, by how much?

5. Effects of changing the voice used in certain parts of the system. For example each time the user descends (or ascends) to a different level a different voice could be used.

Any and all of the above items could be tested using the author's system and such testing would provide useful information on design issues.

5.2 Suggested Enhancements

The system could be expanded in new directions by adding enhancements such as a Database Format Converter or a Phonetic Spelling feature.

5.2.1 Database Format Converter

It is possible to convert menu databases written in the new format to the original format. One possible reason for wanting to do this conversion would be if a person only wanted to adopt the new menu representation and not the new code. An outline for a program to do this conversion is given below.

1. Modify COMMAND.PAS by removing all the procedures except PROCESS and those procedures called by PROCESS.
2. Add an integer field to NODE record type called NODE_ADDRESS.
3. Add an integer variable to the main program called CURRENT_NODE_ADDRESS and initialize it to zero.

4. Modify PROCESS so that each time it creates a new node it increments CURRENT_NODE_ADDRESS and assigns the new value to the NODE_ADDRESS field of the new node.
5. After the input file has been read, traverse the main tree and write out every node.
6. As the output file is being written convert the Pascal pointer values into NODE_ADDRESS values.
7. Precede each line that is written with its NODE_ADDRESS.
8. After all the nodes have been written out, sort the output file by the NODE_ADDRESS field and then remove the NODE_ADDRESS field.
9. Handle the removing of help nodes and the creating of data file name nodes in any convenient manner. Details of this are best left to the individual programmer.

5.2.2 Phonetic Spelling

Testing revealed that the DECtalk mispronounces many of the words found in the test menus. DECtalk provides a method to correct these mispronunciations by spelling a word phonetically.

One additional way to expand this program is to modify it so that it allows both an English spelling and a phonetic spelling for each keyword or synonym. This could be done by allowing any word in the menu representation file to be followed by its phonetic spelling enclosed in brackets. For example, "Hoodoo Gurus" could be represented as "Hoodoo Gurus [Gooroos]".

6 Conclusions

There are many differences between a production system and a research tool. The production system needs to be efficient and reliable while the research tool needs to be flexible and contain metering facilities. The system the author built is a cross between these two types of systems.

The author has shown that a keyword command facility can be incorporated into a menu driven system. Formal testing is suggested in order to determine the advantages and disadvantages of the keyword command style of interaction.

Experimentation to determine if the new database representation is superior to the original database representation is also suggested. Testing of a variety of parameters and options might give further insight and aid in tuning the system for general use.

Bibliography

Bingham, John E., Garth W.P. Davies, Planning for Data Communications, 1977, John Wiley & Sons. Inc., New York, New York, pp. 147-148.

Bohl, Marilyn, Information Processing, 1984, Science Research Associates, Inc., Chicago, Illinois, pp. 106-107, 135-136, 500-501.

DEC, DECtalk DTC01 Owner's Manual, 1983b, Educational Services of Digital Equipment Corporation, Maynard, Massachusetts.

DEC, DECtalk DTC01 Programmer's Reference Manual, 1983a, Educational Services of Digital Equipment Corporation, Maynard, Massachusetts.

DEC, Programming in VAX PASCAL, 1985, Digital Equipment Corporation, Maynard, Massachusetts.

Fast, Lisa, Roy Ballantine, "Dialing a Name: Alphabetic Entry Through a Telephone Keypad", SIGCHI Bulletin, October 1988, ACM, New York, New York, p. 34.

Furnas, G.W., T.K. Landauer, L.M. Gomez, S.T. Dumais, "The Vocabulary Problem in Human-System Communication", Communications of the ACM, November 1987, Volume 30, Number 1, ACM, New York, New York, pp. 964-971.

Horowitz, Ellis, Sartaj Sahni, Fundamentals of Data Structures in Pascal, 1983, Computer Science Press, Rockville, Maryland.

Oppermann, Reinhard, et al., GMD-Studien Nr. 169 Evaluation of Dialog Systems, 1989, Gesellschaft fur Mathematik und Datenverarbeitung mbH, Schlob Birlinghoven, pp. 8-11.

Shneiderman, Ben, Designing the User Interface: Strategies for Effective Human-Computer Interaction, 1987, Addison-Wesley Publishing Company, Reading, Massachusetts, p. 107.

Shneiderman, Ben, Lecture Notes in Computer Science Enduser Systems and Their Human Factors, Human Factors of Interactive Software, 1983, Springer-Verlag, Berlin Heidelberg, pp. 13-14, 21.

Williges, R.C., B.H. Williges, An Integrated Research Paradigm For Experimentation in Information Technology, proposal for grant, Virginia Tech, Blacksburg, Virginia, 1988.

Williges, R.C., B.H. Williges, J. Elkerton, "Software Interface Design", Handbook of Human Factors, 1987, In. G. Salvendy (Ed.), Wiley, New York, pp. 1416-1449.

Appendix A. Glossary of Terms Used

Terms describing menus:

1. Keyword - the word(s) which is spoken for the user. It is a label for the fact that is stored in the node.
2. Node - a single choice on a menu. It contains one and only one fact.
3. Menu - a list of nodes from which the user chooses one.
4. Database - a set of menus stored in a file.
5. Synonym - a substitute for a keyword in the Keyword Command System.
6. Sample Database - a small database for illustration purposes.
7. Menu Level or Depth - how deep a menu is in the database, i.e., the number of levels in the tree.

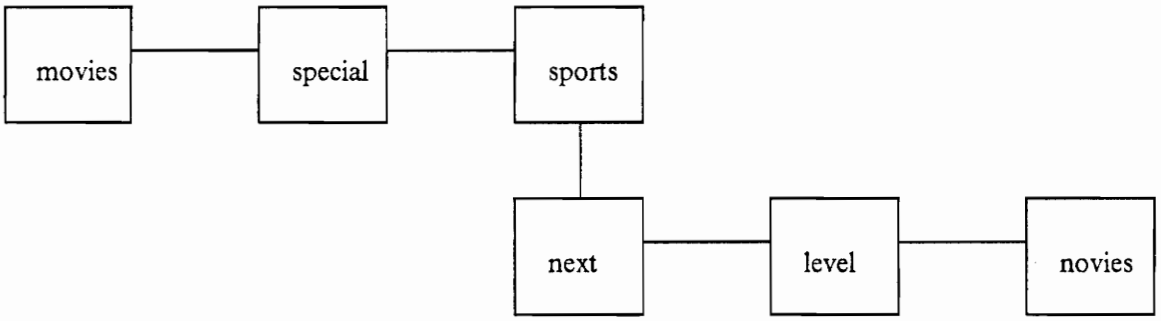
Types of interaction styles:

1. Menu Driven System - the user chooses a selection from a menu.
2. Keyword Command Driven System - the user enters a keyword by pressing keys on the telephone keypad.

Commands:

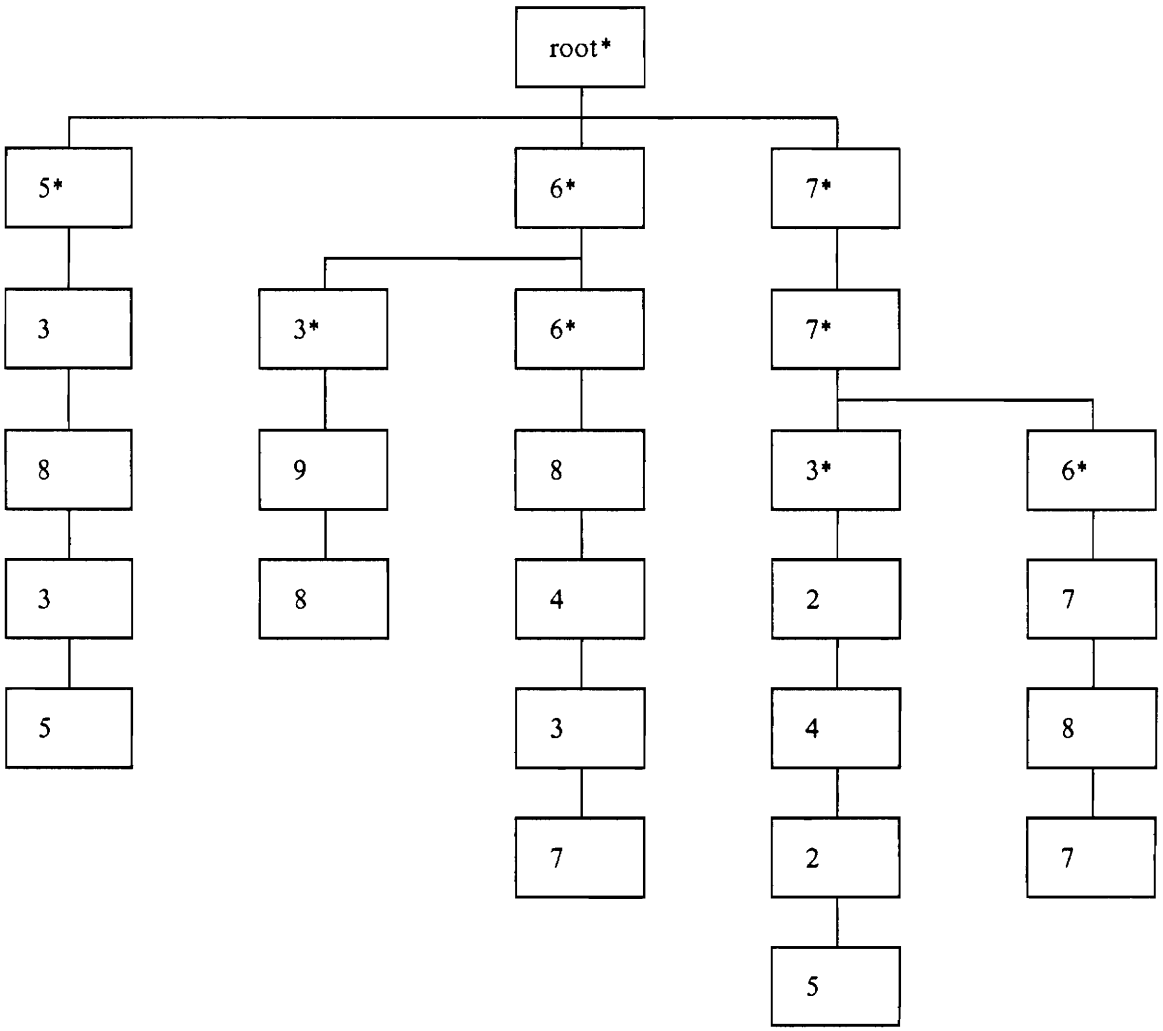
1. Select - choosing a node on a menu.
2. Backup - moving to the parent of the current node.
3. Restart - moving to the first node on the database.

Appendix B. The Main Tree for the Sample Menu



Appendix C. The Auxiliary Tree for the Sample Menu

Nodes marked with an '*' remain after tree compression.



Appendix D. Key Assignments for Double Keying

- 2,* - A
- 2,0 - B
- 2,# - C
- 3,* - D
- 3,0 - E
- 3,# - F
- 4,* - G
- 4,0 - H
- 4,# - I
- 5,* - J
- 5,0 - K
- 5,# - L
- 6,* - M
- 6,0 - N
- 6,# - O
- 7,* - P
- 7,0 - Q
- 7,# - R
- 7,1 - S
- 8,* - T
- 8,0 - U
- 8,# - V
- 8,1 - blank
- 9,* - W
- 9,0 - X
- 9,# - Y
- 9,1 - Z

Appendix E. A Sample Log File

This is a sample log file. It is intended to be understood by and helpful to the programmer maintaining the code.

```
Ready to answer the phone
Answer phone started at 23-FEB-1988 08:27:44.49
Phone answered at 23-FEB-1988 08:27:59.34
Flushing buffer
get response read
get response read
get response read 3
just waited  1.00000E+00 seconds.
Flushing buffer
Saying keyword "NOVIES"
get response read 7
Entering command_mode
get response read 4
Flushing buffer
Flushing buffer
get response read 0
leaving give_menu response 0
at top in abend response 0
final.data NOVIES
leaving command_mode
Flushing buffer
current.data NOVIES
speakfile  FALSE
Saying keyword "NOVIES"
get response read 6
Entering command_mode
get response read 6
get response read 8
get response read 4
at top of display_keywords
Flushing buffer
```

Appendix F. IF3.PAS Database

This is the original IF3 database. Please see "3.1.1 The Original Representation" on page 13 for information about this menu.

```
 0   2   6 Movies
 0   3  11 Concerts
 0   4  14 Sports and sporting events
 0   5  16 Recreational facilities
 0  -1  22 Meetings and special events
 1   7  27 Dune
 1   8  29 Gone with the Wind
 1   9  31 Br'uh'zih'l
 1  10  33 North by Northwest
 1  -6  35 From Russia with Love
 2  12  37 Classical
 2  13  40 Jazz
 2 -11  42 Rock
 3  15  62 Participation sports
 3 -14  67 Sporting events
 4  17 110 Tech gym
 4  18 111 Fitness Connection
 4  19 112 Montgomery County Parks
 4  20 113 Blacksburg Parks
 4  21 114 Blacksburg Pool
 4 -16 115 Fitness trail
 5  23 116 Wild life Society
 5  24 117 Ski Club meeting
 5  25 118 Foot ball team yard sale
 5  26 119 Mental Health Banquet
 5 -22 120 Vet Med dog wash
 6  28  71 Review
 6 -27  47 Theatre
 7  30  75 Review
 7 -29  50 Theatre
 8  32  79 Review
 8 -31  53 Theatre
 9  34  83 Review
 9 -33  56 Theatre
10  36  87 Review
10 -35  59 Theatre
11  38  91 Audabon Quartet
11  39  92 PDQ Bok
11 -37  93 Roanoke Symphony
12  41  94 Windham Hill
12 -40  95 Joe Kennedy
13  43  96 Not Shakespeare
```

```
13  44  97 Hoodoo Gooroos
13  45  98 Flying Eyes
13  46  99 Huwey Lewis
13 -42 100 Sting
28  48  72 Location
28  49  73 Show Times
28 -47  74 Ticket Prices
30  51  76 Location
30  52  77 Show Times
30 -50  78 Ticket Prices
32  54  80 Location
32  55  81 Show Times
32 -53  82 Ticket Prices
34  57  84 Location
34  58  85 Show Times
34 -56  86 Ticket Prices
36  60  88 Location
36  61  89 Show Times
36 -59  90 Ticket Prices
14  63 101 Bowling
14  64 102 Base Ball
14  65 103 Soccer
14  66 104 Soft ball
14 -62 105 Foxridge Classic, 5 K run
15  68 106 Tech volley ball
15  69 107 Tech Base Ball
15  70 108 Tractor pull
15 -67 109 Martins ville 500
27   0   0 dune_m.inf
47   0   0 dune_tl.inf
48   0   0 dune_ts.inf
49   0   0 dune.tt.inf
29   0   0 gone_m.inf
50   0   0 gone_tl.inf
51   0   0 gone_ts.inf
52   0   0 gone_tt.inf
31   0   0 brazil_m.inf
53   0   0 brazil_tl.inf
54   0   0 brazil_ts.inf
55   0   0 brazil_tt.inf
33   0   0 north_m.inf
56   0   0 north_tl.inf
57   0   0 north_ts.inf
58   0   0 north_tt.inf
35   0   0 russia_m.inf
59   0   0 russia_tl.inf
60   0   0 russia_ts.inf
61   0   0 russia_tt.inf
37   0   0 audabon.inf
38   0   0 pdq.inf
```

```
 39   0   0 roanoke.inf
 40   0   0 windham.inf
 41   0   0 kennedy.inf
 42   0   0 not.inf
 43   0   0 hoodoo.inf
 44   0   0 flying.inf
 45   0   0 huey.inf
 46   0   0 sting.inf
 62   0   0 bowling.inf
 63   0   0 baseball.inf
 64   0   0 soccer.inf
 65   0   0 softball.inf
 66   0   0 fox.inf
 67   0   0 hokiev.inf
 68   0   0 hokieb.inf
 69   0   0 tractor.inf
 70   0   0 martin.inf
 16   0   0 gym.inf
 17   0   0 fit.inf
 18   0   0 mont.inf
 19   0   0 parkc.inf
 20   0   0 parkp.inf
 21   0   0 trail.inf
 22   0   0 wild.inf
 23   0   0 ski.inf
 24   0   0 yard.inf
 25   0   0 mental.inf
 26   0   0 vet.inf
  0   0 122 help
121   0   0 help.inf
```

Appendix G. New Representation of IF3.PAS Database

This is the new representation of the IF3 database. Please see "3.1.3 The Second Menu Scheme" on page 16 for information about this menu.

```
IF3.PAS
  Novies (novies.inf
  Movies
    Dune
      Review (dune_m.inf
      Theater
        Location (dune_tl.inf
        Show Times (dune_ts.inf)90
        Ticket Prices (dune_tt.inf
    Gone with the Wind
      Review (gone_m.inf
      Theater
        Location (gone_tl.inf
        Show Times (gone_ts.inf)98
        Ticket Prices (gone_tt.inf
    Brazil
      Review (brazil_m.inf
      Theater
        Location (brazil_tl.inf
        Show Times (brazil_ts.inf)5
        Ticket Prices (brazil_tt.inf
    North by Northwest
      Review (north_m.inf
      Theater
        Location (north_tl.inf
        Show Times (north_ts.inf
        Ticket Prices (north_tt.inf
    From Russia with Love
      Review (russia_m.inf
      Theater
        Location (russia_tl.inf
        Show Times (russia_ts.inf)55
        Ticket Prices (russia_tt.inf
  Concerts
    Classical
      Audabon Quartet (audabon.inf
      P D Q Bach (pdq.inf
      Roanoke Symphony (roanoke.inf
    Jazz
      Windham Hill (windham.inf
```

```
      Joe Kennedy (kennedy.inf
    Rock
      Not Shakespeare (not.inf
      Hoodoo gurus (hoodoo.inf
      Flying Eyes (flying.inf
      Huey Lewis (huey.inf
      Sting (sting.inf
  Sports
    Participation
      Bowling (bowling.inf
      BaseBall (baseball.inf
      Soccer (soccer.inf
      Softball (softball.inf
      Foxridge Classic five K run (fox.inf
    Spectator
      Tech volleyball (hokiev.inf
      Tech BaseBall (hokeib.inf
      Tractor pull (tractor.inf
      Martins ville five hundred (martin.inf
  Recreational
    Tech gym | gym (gym.inf
    Fitness Center (fit.inf
    Montgomery County Parks (mont.inf
    Blacksburg Parks (parkc.inf
    Blacksburg Pool | Pool (parkp.inf
    Fitness trail (trail.inf
  Special
    Wildlife Society Presentation (wild.inf
    Ski Club meeting (ski.inf
    Football team yard sale (yard.inf
    Mental Health Association Banquet (mental.inf
    Vet Med dog wash (vet.inf
end
```

Appendix H. The Test Data

"Instruct.inf" shows the instructions for using this system. "Help.inf" on page 60 shows the help file that is presented to the user, and "Gym.inf" on page 61 shows a sample data file.

Instruct.inf

Using this system, you can learn more about what is going on in the Virginia Tech area. Presently the system can tell you about movies, concerts, sports, recreational facilities, and special events. The system works by speaking lists of keywords, and allowing you to select them. By selecting more specific keywords, you can reach the information that interests you.

After each keyword is spoken the system will pause to permit you to make a selection. There are five basic commands you may use during the pause.

They are: select: press one to select the option you just heard.

pause: press * to place the system in pause. Press * a second time to return to the program.

restart: press zero to return to the top of the menu.

backup: press # to return to the beginning of the previous menu.

You can also spell words that you want information about. Press any number between two and nine to enter a letter.

Look at the template that is beside your phone. It associates three or four letters with each key between two and nine. For example, suppose you wanted to hear information about movies. Movies is spelled m o v i e s. Start by pressing six for m, then six for o, then eight for v, then four for i, then three for e, and finally seven for s. The system will stop you after you have entered enough digits for it to recognize that you want to hear information about movies.

Help.inf

The five basic commands are:

select: press one to select the option you just heard.

pause: press * to place the system in pause. Press * a second time to return to the program.

restart: press zero to return to the top of the menu.

backup: press # to return to the beginning of the previous menu.

You can also spell words that you want information about. Press any number between two and nine to enter a letter.

Look at the template that is beside your phone. It associates three or four letters with each key between two and nine. For example, suppose you wanted to hear information about movies. Movies is spelled m o v i e s. Start by pressing six for m, then six for o, then eight for v, then four for i, then three for e, and finally seven for s. The system will stop you after you have entered enough digits for it to recognize that you want to hear information about movies.

Gym.inf

Notice that some of the words below are intentionally misspelled. This was done to improve the output of the DECtalk unit. The text of this file follows below:

The Virginia Tech War Memorial Jim has several facilities including: Racket ball, Basket ball, Hand ball, and Squash courts, as well as an open use weight room, golf practice room, and a large swimming pool, complete with 8 lanes and a 50 foot platform dive. The War Memorial Jim is also the location of the Tech scuba and aerowbic clubs. The Tech Jim is located in the middle of the South side of the Drill Field. Hours are from 7 Eh M to 9 Pe M Monday through Friday, and 9 Eh M to 6 Pe M on Saturday and Sunday.

Vita

of

James Robert Johnson
July 15, 1992

Home Address

701 Hickory Hill
Nicholasville, KY 40356
(606) 885-1208

Personal

Born - May 2, 1964

Single

Work Experience

The Jockey Club October, 1991 - May, 1992
VM System Support.

IBM Corporation March, 1988 - October, 1991
VM System Support and OS/2 source level debuggers.

Education

MIS - 1992	Virginia Tech, Blacksburg, VA Report: Interface Design for an Audio Based Information Retrieval System Major: Information Systems
BS - 1986	University of Kentucky, Lexington, KY Computer Science