Database Access Using Voice Input and Menu-Based Natural Language Understanding

Ian Menzies

A Major Technical Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

September 1988

ISBN 0-315-44839-3

ABSTRACT

Database Access Using Voice Input and Menu-Based Natural Language Understanding

Ian Menzies

This report reviews the problems facing the development of natural language interfaces to database systems. In particular it reviews three natural language interfaces to database systems that have been implemented. Speech recognition systems are also discussed, and various commercially available speech recognizers reviewed.

A natural language interface to a database system using voice input has been developed. This system is based on a menu driven natural language interface, and integrates several different software and hardware components that are available for personal computers. The system uses a VoiceScribe speech recognizer, managed by programs written in Microsoft C and Arity Prolog, in order to provide an easy to use, affordable, and easily adaptable natural language interface to a small relational database environment.

ACKNOWEDGEMENTS

I would like to thank the following people, without whom the completion of this report would not have been possible: Dr. T. Radhakrishnan, for his unwavering faith and support; my parents, Marge & George Menzies, for all their prayers and assistance; Mariam, for showing me it could be done; Sonia, for her understanding and encouragment; Mame and Malone, for always being there. I would also like to gratefully acknowledge the financial support provided for this work by the Canadian Workplace Automation Research Centre (CCRIT).

Tabel of Contents

Table of Contents (Continued)

Chapter I

Natural Language Interfaces

1.0 Introduction

The proliferation of personal computers and the advances in communication technologies over the last decade have placed vast amounts of information within the reach of an ever growing segment of the population. In spite of the technological advances, the need to master one or several database query languages, or to hire trained computer technicians, has placed much of this information beyond the practical reach of many potential computer users. While the development of many so-called "user-friendly" software packages has alleviated the problem somewhat, these packages are usually quite limited in their scope and capabilities and often do not respond to the on-going needs of their users. If computers are to become truely usable by the casual user, the user must be able to use a language that resembles the natural human language. The computer must be able to understand and act upon commands given in natural language. The development of workable natural language understanding systesms, usually as frontends or interfaces to database systems, is an ongoing challenge to computer professionals in the areas of database management and artificial intelligence. This chapter will look at the variety of problems that must be overcome in order to develop natural language understanding. We review some selected natural langauage database query systems particularly those which have been implemented.

1.1 Natural Language Understanding

Natural language understanding (NLU) by computers in the context of databases is a process of parsing, interpreting and responding appropriatly to a query made in a natural language such as English. Parsing is a process in which the input sentence is checked in order to determine whether or not it is syntactically correct according to the grammatical rules of the language, and if it is, to assign some sort of structure to it, usually a parse tree. To parse a sentence it is necessary to use a grammar that describes the structure of legal sentences in a particular language. Figure 1.1 gives a grammar for a small subset of English, breaking down a legal sentence into its grammatical components, such as nouns, verbs, and determiners. Figure 1.2 shows the parse tree generated for the sentence "The man eats the apple", which is a legal sentence defined by the grammar. Since this grammar uses syntax only to define a legal sentence, it would also accept sentences like "The apple eats the man", and "The apple sings". Clearly, this is not a very good grammar of English.

```
SENTENCE -> NOUN_PHRASE, VERB_PHRASE.
NOUN_PHRASE -> DETERMINER, NOUN.
VERB_PHRASE -> VERB | VERB, NOUN_PHRASE.
DETERMINER -> the.
NOUN -> apple | man.
VERB -> eats | sings.
```

Figure 1.1 A Simple Grammar for a Subset of English

Natural languages are, unfortunatly, very complex and not easily describable by a limited set of grammatical rules. In any NL there are a myriad of ways to convey one thought or concept. For example, "Do you have the time?", "Could you tell me the time?", "Could I see your watch?", and "What time is it?" all express the same desire to know the current time, but exhibit four different syntactic stuctures. This is an example of a many-to-one mapping between syntax and meaning that is a common trait of natural languages. One-to-many mappings are also possible in NL, as in the sentence "John saw the boy in the park with a telescope". Did John see with the telescope, was the telescope with the boy, or was it the park that had the telescope? Another example is "They are flying planes", which can have four seperate interpretations. Sentences such as this are often referred to as being syntactically ambiguous, since they can be parsed correctly in more than one way, and sytactic correctness alone is not sufficient in order to interpret them.

The complex syntactic rules of natural languages pose a major problem to the development of natural language understanding (NLU). The semantics of natural language present another problem. Semantics refer to the meaning of words, and in natural languages the meaning of a particular word may change from one context to another. The question "How many Canadians live in Montreal?", for example, may have different meanings, depending on whether one is requesting census information, discussing political affiliations, or refering to a hockey team. Sentences such as these are vague, or semantically ambiguous, and can be difficult to interpret eventhough they are quite clear from a syntactic point of view.

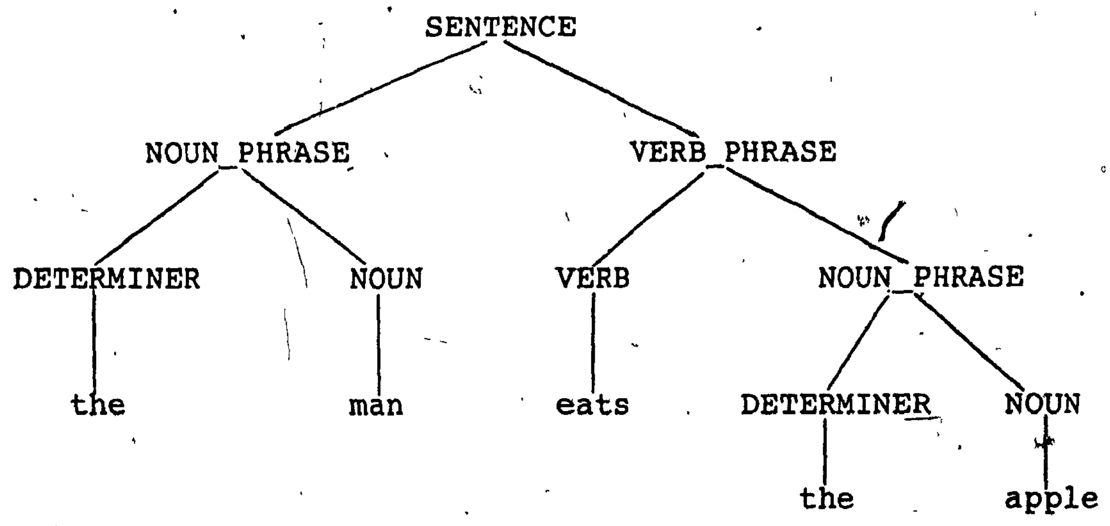


Figure 1.2 Parse tree for "The man eats the apple"

Fortunatly, most NLU systems used in the database context do not require a complete description of the syntax and semantics of an entire natural language. They deal with a limited subset of the language covering the particular domain of information contained in a database system. For a given database system it is possible to define a grammar that will include a class of query or sentence structures sufficent for addressing the application at hand. If the language is well designed we can hope to allow a fairly wide range of straight forward, naturally formulated queries in correct English. Limiting the subset of NL used in the system limits, but does not eliminate, the potential problems presented by the different forms of ambiguity. If the size of the language being used is sufficently restricted, however, many potential sources of ambiguity can be forseen, and mechanisims developed to handle them.

In designing a NLU, two areas of knowledge are normally used: world or global knowledge and domain-specific knowledge. World knowledge refers to the knowledge of the natural language, its syntax and meaning, and how people interpret the language. Such knowledge is used continually by people, usually unconsiously, to disambiguate sentences. It draws upon our understanding and perception of the world in which we live. For example, most people would have no difficulty understanding "The man drove down the street in the car". Few would consider that it might be the street that is in the car, even though this is a perfectly valid syntactic interpretation. People are not likley to confuse the meanings of "baseball diamond", "diamond ring", and "the ace of diamonds", eventhough "diamond" is present in each of the examples. World knowledge is learned by people from childhood and relies upon a lot of purely human experience. Apart from the basic syntax of the language, world knowledge is hard to build into a NLU system. Most NLU systems therefore rely heavily on domain-specific knowledge to interpret queries and handle ambiguity. Domain-specific knowledge is drawn from the words and phrases that are commonly used in referring to a database system and the data contained in it. The names of the databases and the fields in them can be built into the grammar of a NLU system, resulting in a language for the application that is tailor made for the specific database and its users. A NLU interface to a university database need not concern itself with the knowledge necessary for a medical database, and in the context of a baseball database, the word "diamond" need not be ambiguous.

1.2 Implemented NL Interfaces to DataBases

Since the mid 1970's, many attempts have been made to develop natural language interfaces to database systems. While it is not possible here to review many of these systems, we will review three that are of particular interest. The first one to be reviewed was one of the earliest implemented systems, and uses an approach based primarily on domain-specific knowledge. The second system to be discussed, developed in the mid 1980's, relies more on world knowledge. These two systems represent two different, but traditional, approaches to NLU. The third system represents a rather novel and non-traditional approach to the problems of natural language understading.

1.3 The LADDER/LIFER System

One natural language interface to a database that was developed in the mid-1970's was the LADDER (Language Access to Distributed Data with Error Recovery) system, developed at SRI International for use as a managment aid to U.S. Navy [1]. LADDER allows a user to query the naval database in English, and applies all of the lexical and syntactic information necessary to provide the answer. The database used in the LADDER system was made up of about 14 files with over 100 data fields loacated at various remote sites, but from the point of view of the user it was simply a general information database. LADDER freed the user from the necessity of understanding the underlying structure of the database system, and from the need to learn and use a structured database query language.

LADDER was built using three major components, the first of which was INLAND (Informal Natural Language Access to Navy Data). INLAND accepted the user queries in a restricted subset of English and produced one or more queries to the overall naval database. INLAND's queries were then sent to LADDER's second component, IDA (Intelligent Data Access), which broke them down into a sequence of queries against individual fields located on various machines distributed at various sites. This sequence of queries were then shipped to the third component, FAM (file access manager), which located the various files, queried them, ~~and passed~~ the response back through the system to the user's sites.

LADDER's natural language component, INLAND, was developed within the framework of a natural language processing package called LIFER (Language Interface Facility with Ellipsis and Recursion). LIFER supplied basic parsing procedures and allowed the system developer to create the interfaces in an interactive manner. It also contained certain user-oriented features, such as spelling correction, the processing of incomplete inputs (ellipsis), and mechanisims which allowed the user to paraphrase his/her queries. The interactive procedures for developing the language specification made use of semantic grammars, which grouped words together according to their semantic roles, rather than according to their syntactic roles. Words such as NAUTILUS and DISPLACEMENT were not grouped into a single <NOUN> category, but rather into <SHIP-NAME> and <ATTRIBUTE> categories, respectivly. Specific sentence structures were then built, giving grammar rules such as "What is the <ATTRIBUTE> of <SHIP>", rather than <NOUN-PHRASE>, <VERB-PHRASE>. For each such pattern an expression was supplied by the language designer for computing the interpretations of instances of the pattern, resulting in production rules such as that in Figure 1.3. LIFER maintained lexicons of individual words and fixed phrases that were associated with each of the metasymbols present in the production rules, such as <PRESENT> and <ATTRIBUTE>. The lexicon entries for the metasymbol <PRESENT> consisted of words such as PRINT, LIST, SHOW ME, and WHAT ARE, while <ATTRIBUTE> would have associated with it CLASS, FUEL, LENGTH, etc. Once all of the production rules had been specified by the language designers, they were built into a transition tree, a simplification of an augmented transition network. If the

LIFER grammar was defined by the four productions in Figure 1.4, where LTG stands for LIFER TOP GRAMMAR, and the e1, e2, e3 and e4 are the expressions associated with each of the patterns, the transition tree built from it would be that shown in Figure 1.5. When a query was presented by the user, LIFER's parser started at the beginning of the transition tree, attempting to move towards the response expressions on the right, working top-down, left-to-right. Literal words could be matched only by themselves, while a metasymbol such as <ATTRIBUTE> could be matched with one of the words associated with it in the lexicon.

The variety and complexity of queries that can be built into a NLU system by LIFER are limited only by the time and effort put into the design, and by the time and space restrictions of the computer being used. In the case of LADDER, the types of queries allowed were quite broad, permitting queries of the type "List the current position and heading of the US Navy ships in the Mediterranean every 4 hours" and "What US ships faster than the Gridley are in Norfolk". LIFER's special features, such as spelling correction, the use of ellipsis, and allowing the user to redefine or paraphrase queries, made LADDER's natural language component both helpful and palatable from the user's point of view.

```
<PRESENT> the <ATTRIBUTE> of <SHIP> |
           (IDA (APPEND <SHIP> <ATTRIBUTE>)).
```

Figure 1.3 LIFER Production Rule

```
LTG -> <PRESENT> the <ATTRIBUTE> of <SHIP> | e1
LTG -> <PRESENT> <SHIP's> <ATTRIBUTE> | e2
LTG -> How many <SHIP> are there | e3
LTG -> How many <SHIP> are there with <PROPERTY> | e4
```

Figure 1.4 LIFER Sample Grammar

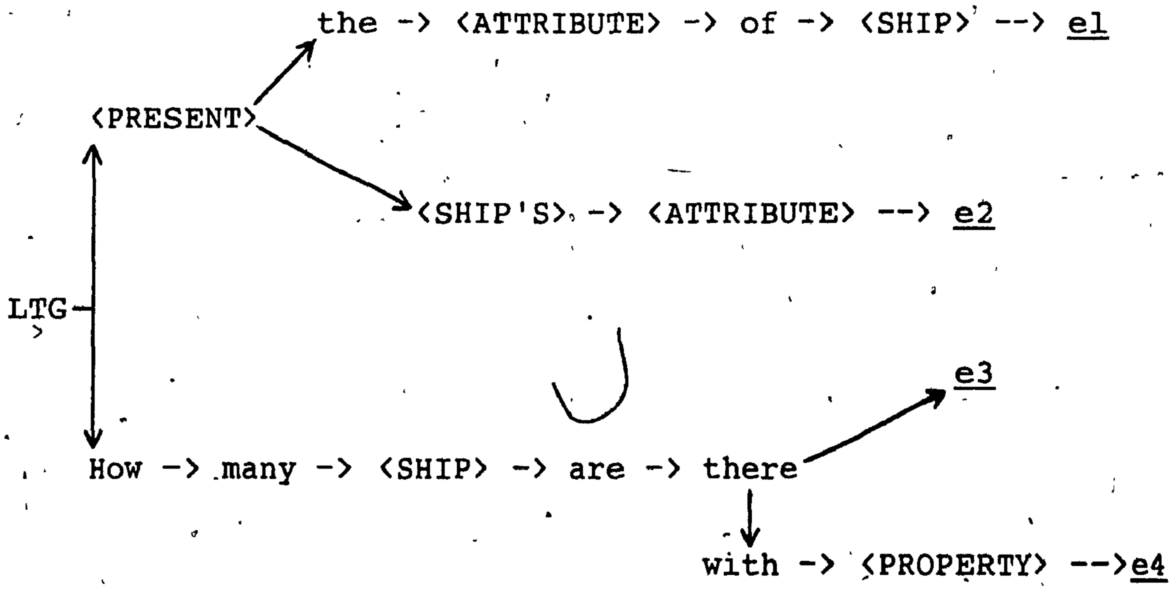

the -> <ATTRIBUTE> -> of -> <SHIP> --> e1
<PRESENT>
<SHIP'S> -> <ATTRIBUTE> --> e2
LTG
e3
How -> many -> <SHIP> -> are -> there
with -> <PROPERTY> -->e4

Figure 1.5 LIFER Transition Tree

While LADDER was quite robust in its handling of NL queries, it did suffer from several limitations. LIFER provided no general mechanism for dealing with the omission of words at arbitrary points in a sentence, so would treat "What Lafayette and Washington class subs are there within 500 mile of Gibralter" as "What Lafayette class subs are there", and "What Washington class subs are there". In order for LIFER to properly interpret the query, it would have had to have been entered as "What Lafayette class subs are there within 500 miles of Gibralter and what Washington class subs are there within 500 miles of Gibralter". Eventhough LIFER's grammar can, in theory, have any type of sentence structure built into it, in practice only a limited number can be formulated, often resulting in irregular coverage that can be irritating and confusing to the user. For example, the system might accept "The Kennedy is owned by whom" and "Who commands the Kennedy", but not "The Kennedy is commanded by whom". The use of ellipsis also suffered from irregularity. LADDER's elliptical processor was based on syntactic analogies, so would easily handle "How many cruisers are there?" followed by "cruisers within 500 miles of the Knox?". Since the second phrase is a noun phrase that is analogous to the noun phrase "cruisers" in the first query, ellipsis will work. However, if the second phrase had been "within 500 miles of the Knox", ellipsis would have failed, since this is a modifier with no analogous part in the first query.

LIFER provided no mechanisms for dealing with either semantic or syntactic ambiguity. Rather, the system was based on the assumption the the users had a very good idea of what was in the underlying database, knew how differnt bits of information were related, and would avoid using long and complicated sentence constructions, since most of the users disliked typing at a keyboard. Apparently these assumptions were fairly well founded, and ambiguous sentences were rare occurences; when they did occur, LIFER accepted the first legal parse of the sentence as being the only interpretation of the query.

Many of the shortcomings of the LADDER/LIFER system can be attributed to the fact that the design of its linguistic component was based solely on domain-specific knowledge. Although a good deal of domain-specific knowledge is needed to design any NLU system, it is not in itself sufficent to deal with troublesome linguistic phenomena such as syntactic and semantic ambiguity. In order to handle such phenomena it is necessary to have recourse to a certain amount of more general world knowledge. An over reliance on domain-specific knowledge also results in a NLU system that is bound to one particular database domain, and cannot be easily moved to another domain of knowledge without a substantial recoding effort. Such recoding can be costly and time consuming, and reduces the cost effectivness of domain-specific systems.

1.4 The COOP System

While the LADDER/LIFER system demonstrates the domain-specific approach to NLU, another system, COOP, was designed with more of an emphasis on world knowledge [2,3]. The main premise behind the development of COOP was that language-driven inferences should be distinguished from domain-driven inferences in the designing of a NLU system. Most NLU systems base the "understanding" of a query on inferences drawn from the underlying domain of the database - these are domain-driven inferences. Language-driven inferences, on the other hand, are based on the knowledge of the language itself, and upon the language related conventions that people use when communicating with each other. A story, dialogue, or question, is a description, and the description itself contains certain useful properties that are not associated with the domain being described. Words such as "former", "latter", and "respectivly" make use of the linear nature of the language to convey their meaning, and from a statement such as "John didn't know that the exam was yesterday" one can infer that the exam was yesterday, and that John may have missed it.

When dealing wih questions, language-driven inferences can allow one to address the presumptions that are inherent in the question, and to respond appropriatly. For instance, the question "Did John pass Comp210?" carries with it the information that the questioner believes there is a Comp210 and that John took the course. Giving a direct answer of either "yes" or "no" is only appropriate in the case where there is a Comp210 which John took. If, however, the question is based on one or more false presumptions, a direct answer of "no" would not be appropriate, since it would only serve to reinforce the questioner's misconceptions. In such a case, an indirect response such as "there is no Comp210" or "John didn't take Comp210" is more appropriate and helpful, since it corrects the questioner's false beliefs.

The COOP NL query system was designed with two hypotheses in mind. The first of these was that language-driven inferences are sufficent to run procedures which detect the need for a cooperative indirect response, and select an appropriat one. The second hypothesis was that the domain-specific knowledge needed to handle a significant class of NL queries already exists in a standard way in most database systems and need only be augmented with a suitably encoded lexicon. The mechanisims used to produce cooperative indirect responses are therefore domain transparent and can be transfered to new database domains with relativly little effort.

COOP computed the need for corrective indirect responses in the following manner. The user's query in natural language was transformed into an intermediate representation, called the Meta Query Language (MQL). The MQL was a graph structure where the nodes represent the sets given by the user, and the edges represent binary relations defined on those sets. The nodes and edges are based on the lexical and syntactic structure of the user's query, and are therefore domain independent. For example, the query "Which students got F's in engineering courses" would be representd by MQL as shown in Figure 1.6. Each of the subgraphs in the MQL representation corresponds to a presumption the user has made concerning the domain of the database. Should the actual query to the database return a null set, each of the user's presumptions could be checked against the database for non-emptiness, and if a presumtion was found to be incorrect, an appropriate corrective response would be generated. It is noteworthy that the computation of the user's presumptions was totally language-driven, and that access to domain-specific knowledge was needed only to select an appropriate response.

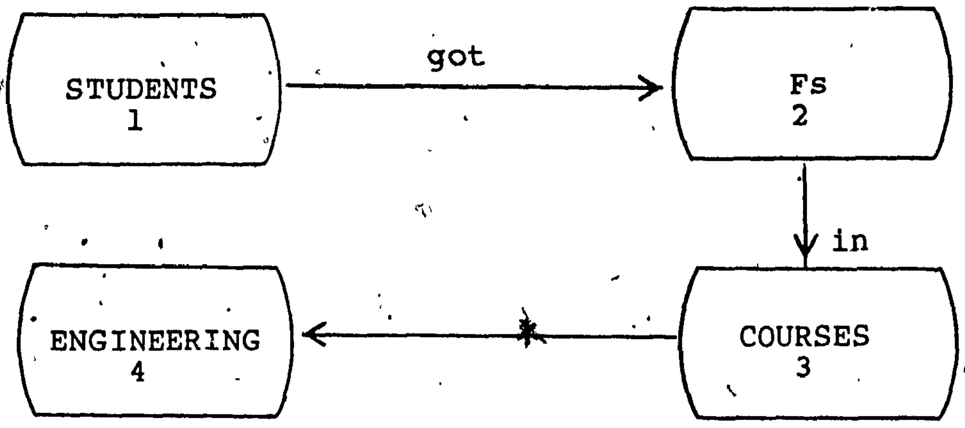


Figure 1.6 Meta Query Language Representation of "Which students got Fs in engineering courses?"

COOP was designed to be more than just cooperative - it was also meant to be a portable NL query system. While the procedures which determine the need for corrective responses are completly language-driven, domain-specific knowledge is required for parsing, interpretation and translation of NL queries into database queries. Since it is the need for domain-specific knowledge that limits the portability of NLU systems, COOP's sources of such knowledge were limited to sources that were implicit in the database system, well defined in scope, and did not require a large coding effort. These sources were the database schema, the database contents, and the lexicon, the first two of which are already present in any database application, and therefore don't require a major coding effort. The coding of the lexicon alone required a substantial effort. The lexicon consisted of three types of entries: general, stuctural, and volatile. General entries were verbs such as "to be" and prepositons like "in" and "from", which are largely domain independent and required little more than fine-tuning to apply to a new domain. Volatile entries refer to the actual database contents, and were not explicitly coded in COOP. The majority of coding was therefore for the structural entries, which were generally nouns that referred to specific data fields, and verbs that were used in the particular domain.

Unlike the LADDER/LIFER system, COOP included mechanisms for resolving the different types of ambiguity that pop up in NL queries. Syntactic ambiguity most often arises when a modifying clause has more than one potential noun head, as in "John saw the boy in the park with the telescope". Coop employed three domain transparent heuristics for ranking the various potential head nouns for a particular modifier. The first was based on the distance back in the query to the head, with the head nearest to the modifying clause receiving the highest rank. The second heuristic was based on the predictive values of various words in the modifying clause. A verb such as "sponsors", for example, makes a strong prediction that its subject is a sponsor, while a verb like "is" has little predictive value. The third heuristic was to measure the distance between the modifier and its potential heads in the database schema. Since semantically related terms tend to be near each other in the schema, this simple heuristic was often very useful for disambiguation. For example, in the question "Which professors teach courses in Digital Design that are in the CompSci Department?" the modifier, "that are in the CompSci Department", has two possible heads, "professors" and "courses". If the database schema showed that professors were organized into departments, while courses were organized by areas-of-interest, "professors" would be chosen as the appropriate head. Having applied the three heuristic measures, COOP calculated the scores of the potential heads, and chose the one with the highest score.

Queries such as "who advises students in 641" are semantically ambiguous, since it is not clear who "who" refers to, and the meaning of "641" is not precise. When COOP came across such a query, it develops a set of condidate meanings for the various ambiguous words, and then applied two heuristics borrowed from the syntactic disambiguation process to constrain the possible meanings of the word. The first heuristic was to use the predictive value of the words in the query. The word "advises" makes a strong prediction that its subject is an advisor, while "in" might predict a course, department, or room. The second heruistic then examined the schema. In a universtiy schema, it is likely that "students" would be closer to "courses" that to "departments" or "rooms", so COOP would infer that "641" was probably a course number.

LADDER and COOP are the products of two different approaches to NLU query systems, developed with different goals in mind, and for different types of users. LADDER was designed for use on a particular database, to be used by people who were aware of the contents of the database and the types of questions it could answer. With these considerations in mind, the domain-specific approach was well suited to LADDER, and the absence of mechanisims for resolving ambiguity was not a major drawback. It was not designed with portability as a major consideration, and would, as a result, be difficult to port to another database domain. COOP, on the other hand, was designed for a less experienced user, and to be easily portable to any domain of knowledge, so had to minimize the need for domain-specific data, and concentrate on the use of language-driven inferencing. COOP did not make the assumption that its users would be aware of the precise contents of the database; nor did it assume that its users would ask only clear, unambiguous queries. Because of this, COOP had to provide cooperative responses, and have mechanisims for dealing with ambiguity.

While both LADDER and COOP worked quite well within the frameworks of their respective design, a study of the users of NLU query systems found that there were several problems inherent to most systems of this type. One of the most basic problems is that many users cannot type or spell very well, resulting in a lot of misspellings in the input queries. This problem can be accentuated by the fact that many users are not comfortable with computers, and have difficulty articulating their queries. The next major problem was with the scope of the natural language. Users often asked questions that the system could not understand, but could have understood if formulated in another way. Due to this, users often retreat to very simple query structures, and do not learn or use the system to its full linguistic potential. Another problem related to the conceptual coverage of the system was with users asking for information that was not contained in the database system.

There are several traditional approaches to overcome the problems inherent in NLU query systems. One is to give all users sufficient training and instruction in the linguistic and conceptual coverage of the system so that they do not ask questions that cannot be answered. Unfortunatly, this would make systems unavailable to the truly casual user, and even trained users might not remember all of their training from one session to the next. Another possible solution is to expand the linguistic coverage to cover all possible queries. Even if this were possible, it would be very time consuming, result in systems that were highly domain dependent, and not solve the problem of exceeding the conceptual coverage of the system. A third solution is to engage the user in some sort of clarifying dialogue when problems arise, as COOP did for user misconceptions. This, however, only works if all possible problems can be forseen, and mechanisms developed to deal with them.

1.5 The NLMenu System

LADDER and COOP used a variety of methods in an effort to overcome some of the above mentioned problems. LADDER assumed experienced users, and employed spelling correction for those who couldn't type or spell, while COOP handled the problem of conceptual coverage with cooperative responses, and used several heuristics for handling unclear queries. A third system, NLMenu, overcame the problems of NLU by employing a completly different approach [4,5,6]. NLMenu was designed to incorporate all of the advantages of natural language, while eliminating the need for training, making the linguistic and conceptual coverage of the system highly apparent to the user, and providing a 0% failure rate.

NLMenu works in the following manner. The user is presented with a screen containing a series of menus, each of which contains several words or phrases representing the various components of a natural language query. The user may choose any word or phrase from an active menu, using either the keyboard arrow keys or a pointing device. As the user picks words, the partial input query is parsed, and based on this, the next set of menus are activated, presenting the user only with those options that make sense under the current context. If the next sensible entry is a database value, NLMenu presents one of its "expert" menus, containing specific database values from which the user may choose. In this way the user can formulate a complete query in natural language that the system is guaranteed to understand.

The beauty of the NLMenu concept is its clarity and simplicity. The user can see what types of queries can be formulated and the domain of information contained in the system just from looking at the screen. Since the user does not have to type, the problem of misspelling does not exist. Since the user is restricted to choose from the active menus, only queries that can be understood can be constructed, eliminating all sources of ambiguity. By displaying particular database values through the "expert" menus, the need to maintain a lexicon of database descriptors is removed, and the contents of the system is always up to date.

The structure of NLMenu also makes it easily portable from one database domain to another. Starting with a small core grammar and lexicon, all that is needed to build an interface is a description of the names of the relations in the database, their attributes, and the characteristics of the attributes. Based on this information, an interface can be generated automatically, or the user can generate a new interface by choosing from a menu the set of relations to be covered. Since NLMenu only provides one way to phrase any given query, the task of building an interface is not open-ended, and can be done by the end user.

One of the main drawbacks of NLMenu is the size limitiation imposed by the screen. A large database system, with many relations, attributes and values would be difficult to mold into the menu framework. A standard 24 line screen could not hold more than fifteen or so lines of text if there is to be enough room for the menu frames on the screen. A high resolution screen might allow smaller letters and more text, but would be hard on the eyes and would present the user with a confusing number of options to choose from. Scrollable menus, or overlapping menus, might provide a partial solution, but would reduce the clarity and simplicity of the screen menus. Still, for a small number of relations, not more than twenty, the NLMenu approach to NLU seems to provide a relativly robust and expressive way for casual users to access database information, and one which is particularly well suited for use on the omnipresent personal computer.

Chapter II
Speech Recognition

2.0 Introduction

The goal of natural language interfaces to database systems is to allow casual users to query a database in a fashion with which they are familiar. Most traditional NL interfaces, however, rely on keyboard input of user queries. The advent of relativly low cost speech recognition hardware may now make it possible for casual users to query databases in the most natural form possible, namely using spoken natural language. This chapter reviews the major problems involved in speech recognition, the different types of speech recognizers, and looks briefly at some of the commercially available systems.

2.1 Problems of Speech Recognition

Just as NLU faces several inherent problems, there are problems encountered by all types of speech recognition systems [7,10]. First of all, no two people talk in exactly the same fashion, and no two people sound alike. A practical speech recognition system must be able to distinguish between speaker dependent variables, such as accent or pitch, and phonetic information, in order to understand more than one person. Speech recognition must also deal with the problem of phonetic ambiguity, since sounds do not map one-to-one onto phonemic variables or words. Humans deal with this by drawing on their knowledge of the language and context, but such knowledge is difficult to build into a computer. A third problem is that the speech patterns of individuals vary from one day to the next, and can be effected by factors such as fatigue, and psychological or emotional stress [12]. People do not always speak clearly, often reducing short words to monosyloabic grunts and running syllables and words together. The duration of a spoken word may vary over time, as may the accents placed on different syllables, and a speech recognition system must be able to deal with these variations in speech. Finally, a speech recognition system must be able to distinguish a speaker's voice from background noise and interference. While speech in a quiet, controlled environment poses fewer problems in this respect, a practical work environment may produce a lot of background noise, such as closing doors, ringing telephones, and people talking.

2.2 Speech Recognition Systems

Speech recognition systems can be divided into four catagories, in order of increasing difficulty. The least complex form of recognition is isolated-word recognition. Here the unit of recognition is the word, with words separated by clear pauses in speech. Pauses between words simplify the detection of the start and endpoints of each word. While this form of speech is not natural for people, isolated words are generally pronounced more carefully than words in continuous speech, and are therefore easier to recognize. In commercial systems isolated-word recognition is the most common form of recognition, and will be discussed in greater detail later on in this chapter.

The second type of recognition is word-spotting, or the detection of occurrences of a particular word in continuous speech. Each word to be recognized is represented by a template or model and the recognizer attemps to match these templates with the incoming speech stream. Unlike isolated-word recognition, there are no distinct pauses between the words in the speech stream, so the recognition process must be independent of starting and end points of utterances. The process must therefore treat each sample of incoming speech as a potential starting point for a word, and attempt to match the successive speech signals with the known word templates.

The third type of recognition is continuous-speech recognition. Because it is not practical to recognize an entire phrase as a unit, the process for continuous-speech recognition must be able to break or segment the speech stream into smaller parts, and to recognize word boundaries. Usually the unit of recognition is either the word or the phonemes that make up words. Word-based continuous-speech recognition can be built upon the techniques used for word spotting, but such an approach is only practical for tasks with very small vocabularies, such as digit recognition. Phoneme based recognizers, on the other hand, attempt to recognize individual phonemes, and then to identify words as collections of phonemes. The word matching process is usually based on phonetic rules, vocabulary and syntax rules that specify the legal sequence of phonemes and words. These rules are often in the form of grammars, much like those used for parsing sentences in NLU systems.

The fourth, and most ambitious form of speech recognition is speech understanding. The goal of such systems is to be able to understand continuous speech using the same processes that humans do. Such systems must be able to ignore noise and irrelevant speech, understand context, resolve ambiguities, and handle ungrammatical or incomplete sentences. Ideally, a speech recognition system should also be able to understand any person. (Some of the techniques used for achieving speaker-independance will be discussed later in this chapter.) In speech understanding it is more important to understand the meaning of speech than to recognize individual words. As with continuous-speech recognition, speech understanding systems use phonetic identification and word matching. Rather than relying only on syntactic rules, however, the word-matching process must also use lexical, semantic and world knowledge in order to "understand" the speech [11]. The problems of speech understanding systems are not only those of continuous-speech recognition, but also those of knowledge representation and artificial intelligence.

2.3 Isolated-Word Recognition Techniques

As mentioned above, isolated-word recognition is the most common and least complex of the four types of recognizers. As a result, many of the processes used for recognition in the other three recognizers are built upon techniques developed for isolated-word recognition. The first step in developing any speech recognition system is setting up a library of word patterns or templates for the words to be recognized. This is done by having a user repeat a given word distinctly several times, and thereby train the system. The features of the spoken word are then used to build a template for that word. Which features are extracted from the word in order to build the template may vary from system to system, but typical features are:

1) Amplitude versus time,
2) Zero-crossing rate,
3) High-frequency versus low-frequency energy,
4) Fine spectral details.

Using such features, patterns are constructed based either on some sort of feature-by-feature segmentation of the word, or on time functions which span the whole word. The latter approach is the most common for isolated-word recognition.

After the word templates have been established, they are used for recognizing incoming speech. For isolated-word recognition the pauses between spoken words simplify the process, but do not remove all ambiguity. A word said during recognition is rarely said in exactly the same fashion as during training. Generally, the utterances will not be of the same duration, nor will the spacing of phonetic events be consistent. Furthermore, many speakers will end a word by trailing off the intensity of the word, or with a short breath noise that can result in the misidentification of the word. It is therefore important for the recognition process to be able to accuratly identify the endpoints of the word. Normally amplitude is used to identify endpoints, with the start of the word being the point where the energy of the word exceeds some threshold value, and the end being the point where the energy drops below the threshold. The endpoint recognition process must also be able to filter out random noises that may exceed the energy threshold and be able to handle breath noises that may obscure the precise endpoint of a word.

Once an utterance has been detected, the recognition process must attempt to match it with one of the stored word templates. Due to the differences between words spoken during training and those spoken for recognition, exact matches are not likely. In fact, the unknown utterance may appear as different from the correct template as it does from those that are incorrect. This discrepancy between the unknown utterance and the templates is usually handled by some sort of time normalization process. Time normalization was originally simply to stretch or compress the unknown uniformly to make it the same lenght as the template. The accuracy of this process depended on accurate endpoint detection, something that is not easy to guarantee. A process known as "time warping" is now frequently used for nomalization. Time warping is a process whereby the time axis of the unknown utterance is distorted, or warped, in a nonuniform way in order to aligne its features with those of the template. Figure 2.1, taken from Parsons [7], gives and example of time warping. The two contours to be matched, A and B, are shown along the axes, and the wavy diagonal line, XY, shows the mapping between them. If the mapping fuction passes through point (i,j) the ith sample of contour A is aligned with the jth sample of B. If the matching was only a uniform expansion or compression, the line shown would be straight. Time warping is exceptionally powerful and has greatly improved the accuracy of recognition systems.

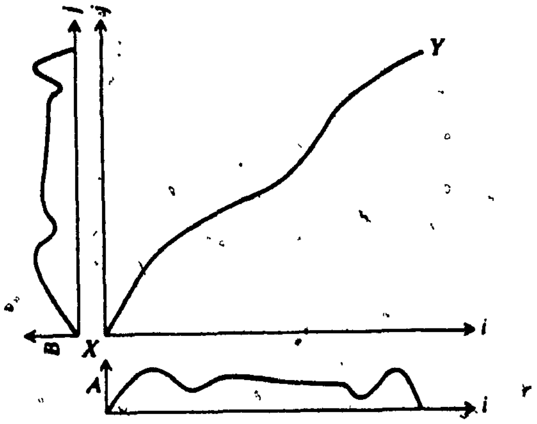


Figure 2.1 Time Warping

Speech recognition of any kind works best with a single speaker. Several stratagies have been developed, however, to allow for speaker-independent recognition. One stratagy is to select features for the word template that are stable between speakers. Since such features must represent relativly broad phonetic categorizations, such as vowels and consonants, this approach is only practical for very small vocabularies. A second approach is to maintain multiple templates for each word, one per word for each speaker. For this approach to be practical, the templates must be grouped in such a way as to avoid a prohibitivley large pattern library. A third approach is to average the speech patterns obtained during training to give one general template for all speakers. Irrespective of the approach taken for training speaker-independence, some sort of formant frequency normalization must be carried out during the recognition process, since different speakers have different formant frequencies for the same vowels [13]. This type of normalization is often done using simple linear scaling that normalizes the speaker's formant frequencies with those of the templates, or with a non-linear warping process similar to time-warping.

While much work has been done in the development of speech recognition technology, the state of the art is not yet at the point where speech provides a truly practical interface between a computer and a casual user. Isolated-word systems are the most common and reliable, but isolated-word speech would not be considered natural by the casual-user. Speaker-dependent systems can provide high levels of recognition, especially in a controlled environment. Speaker independence would be necessary for the casual user. However, speaker-independence, except for very small vocabularies, is difficult to achieve. The problems of environmental noise must also be considered with a casual user in a realistic working environment. Despite these problems, many speech recognition systems have been made commercially available in recent years. In the next section, wil will take a brief look at recognition systems now available on the market.

2.4 Commercially Available Systems

There are over 30 speech recognition systems now available on the market, ranaging in price from a few hundred dollars to close to $100,000CDN [8]. These products can be grouped into three categories: a) single chip or chip sets, b) single circuit-boards, and c) complete systems (incorporating computers, displays, and remote control boxes). Of these three categories, single circuit-boards are becoming the most popular. There has also been a growing trend to develop and market speech recognition software for use with recognition hardware.

To accurately compare the relative performance of various recognizers, one would have to consider various features, such as speaker dependency, noise handling, the form of speech permitted, cost, and error rate. Unfortunately, very little information is available on recognizer performance. Furthermore, data on the internal features of the recognizers, which would be useful for evaluation, is generally not provided, in order to protect proprietary rights. There does seem to be, however, a fairly close correlation between cost and error rate, with higher cost associated with lower error rates.

While it is difficult to accurately characterize the performance of a particulat recognizer, it is possible to say what recognizers in general cannot do. Current devices cannot easily handle continuous speech, speaker independance, large vocabularies, high noise levels, or environmental conditions such as vibration, speaker stress, fatigue, or emotion. Device manufacturers rarely provide clear guidelines for the design of a good vocabulary, or describe applications for which their device is particulary not suited. In general only a small fraction of what is known about speech production, accoustics, human hearing and perception, and linguistics is incorporated in any existing speech recognition device.

2.5 Desirable Characteristics of a Speech Recognizer

What are the desirable qualities that a recognizer should posses? In particular we will look at qualities that would aid in the design of database query systems using isolated-word recognizers. One of the most necessary qualities that a recognizer should possess is that it is easily integratable. Being integratable has two aspects.

First, a recognizer must be easily integratable from a hardware point of view. A circuit board that can be fit into the expansion slot of a personal computer is an example of this, while one requiring more complex connections is not. Given the popularity of personal computers, it is also important that a recognizer be compatible with existing PC technologies if it is to survive in the marketplace. A recognizer should also be integratable at a reasonable price, while offering a high recognition rate. As a rule of thumb, a recognizer should not cost more than the PC itself.

The second aspect of integratability deals with software. Given the popularity of software packages such as LOTUS-123 and Dbase III, it is important that a recognizer be able to work with such packages, either directly or indirectly. In order to be able to do this, the device driver for the recognizer must be small enough that, once it has been loaded, sufficent memory remains for other applications. Apart from software packages, a recognizer should also be integratable with some of the more well known programming languages, such as PASCAL, BASIC or C-Language, and should permit direct access to the recognition processes. This type of integratability allows for the development of fairly specialized recognition systems that would be beyond the scope of packages such as Dbase or LOTUS.

Another desirable quality of a recognizer is that it should be fairly easy to learn and use. This does not mean that a complete novice should be able to master the system in a few hours. It does mean, however, that someone should be able to get a good feeling for the recognizer's capabilities in a day or so without being an expert in speech technology. In order to possess this quality, a recognizer should be accompanied by a clear and well documented user's guide, easy to use demonstrations, and good development tools and utilities. The ease of use, however, should not be restrictive. It should be possible for a qualified programmer to develop his/her own routines for use during recognition, and not to have to rely soley on the existing development tools. This can be made possible only through a high level of integratability with programming languages, and a good development manual for advanced users.

A further aspect of ease of use is ease of training. A recognizer should come with a training utility that permits the end user to train the vocabulary without necessarily understanding the training process. Since a user's voice may change over time, the word templates created by the training process should be adjustable, so that the templates can be updated from time-to-time. Advanced users should have the option of developing their own training routines to meet their own particular needs. The quality of homemade training routines depends, to a certain degree, on the quality of the programmer, and on the ease with which the recognizer can be manipulated by programming languages.

Yet another characteristic that a recognizer should have is flexibility. A good recognizer should be usable for any number of different applications, in any given language, and in variable work environments. It should be able to adjust the recognition parameters to take into account different levels of background noise. A good recognizer must be flexible enough to permit different types of microphones, or input from a telephone or tape recorder. The recognizer should work equally well with small and fairly large vocabularies, and for very small vocabularies, should allow for use by multiple speakers. Finally, a recognizer should be able to make use of expanded memory, beyond the 640K limitation of DOS, when expanded memory is available.

Chapter III

System Technologies and Integration

3.0 Introduction

In the last two chapters we looked at the problems facing natural language understanding and the various types of speech recognition technologies. While unrestricted natural language or speech understanding is beyond the scope of current technologies, it is possible to use limited NLU and speech recognition to provide more natural interfaces for the casual user. The next chapter will look at a small system that combines speech recognition with the NLMenu approach to natural language interfaces. In this chapter we will discuss the technologies that were used in the system, why they were chosen, and how they were integrated.

3.1 The Computer

The primary goal of this project was to develop a simple, and yet non-trivial, natural language interface to a database using speech recognition. Since the project was undertaken with the casual user in mind, it was important from the outset to use technologies and systems with which the user would be familiar, and to develop the system at a reasonable cost. The best choice for the computer system was clearly the ubiquitous personal computer, since they-have become relativly inexpensive, and are probably the most familiar computer in current use. Furthermore, there is a large variety of software programming languages and commercial data management systems available for the PC, and a number of relativly inexpensive speech recognition systems for the PC are available. The computer actually used in the development of the system was initially a PC-XT clone with 640k. The system was later moved to a PC-AT in order to improve the system's overall speed and allow for the future growth and development of the system.

3.2 The Natural Language Understanding System

Development of a system for the casual user using speech recognition also placed restrictions on the type of NLU interface to be used. As mentioned earlier, users of traditional NLU systems often exceed the the conceptual or linguistic coverage of the system. When voice is added, another problem arises, that of exceeding the phonetic coverage of the system, or saying something that the reccognizer cannot recognize. The NLMenu approach to NLU seemed to be the most amenable for user with speech recognition. Just as NLMenu removed the problems of linguistic and conceptual over-reach, it seemed to offer the best way of eliminating the possibility of phonetic overreach. Since NLMenu leads the user through the construction of a query by highlighting the next set of possible words, the user would not be expected to say a word that was not included in the recognizer's pattern library. Equally important was the fact that the highlighting of the next possible words in NLMenu takes just long enough to force the user to pause between words. Since isolated-word recognition is being used, the pause beteween the words is very important for accurate recognition. The small size of the NLMenu system also makes it practical for use on a PC.

3.3 The Speech Recognition System

Cost restrictions elimininated the possiblilty of using a continuous speech recognition system, so isolated-speech recognition was chosen. Several reasonably priced isolated word recognizers are available on the market for use with PC's. The recognizer used for the system was the Voicescribe 1000 speech board. The VoiceScribe recognizer meets most of the requirements of a desirable recognizer outlined in the previous chapter. First of all, it meets the cost requirements, selling for less than CDN $1700. The VoiceScribe system also meets the ease of installation requirement. The recognizer's hardware is contained in a single circuit board, and can be installed in a PC by simply inserting the board into one of the PC's expansion slots. Installation of the recognizers's software can be done automatically by running an installation program, or manually in a few minutes by someone who is familiar with a PC.

VoiceScribe is also quite easy to learn to use. The system comes with a well documented training utility, called DragonLab, that allows a user to go through the recognition process in a step-by-step manner, loading the language files, training the words, and then recognizing them. Throughout the DragonLab procedures, the user is given feedback concerning amplitude levels, confidence factors, and other variables pertinant to the recognition process. While DragonLab does not teach the user everything there is to know about developing a speech recognition system, it does demonstrate the various steps and concepts that the user must consider when developing a recognition system.

VoiceScribe uses a language compiler, called VOCL, to transform a language description into a structure similar to the ATN. In order to develop a system, the user first designs the language to be used, describing the language in a grammar similar to those more commonly used for continuous-speech recognition. This grammar is then transformed into a network of productions and states. As the user speaks, the network moves from one state to the next, restricting the choice of possible words at each state. This formalism improves the recognition rate by restricting the words that can be said at any given state, and parallels quite closely the highlighting of words in NLMenu.

Another VoiceScribe development utility is DragonKey. Once activated, DragonKey can run in background while another application is being used. The utility also provides pop-up menus that allow the user to train and recognize words from within another application, or to see which words can be recognized at any given point in the application. DragonKey comes with several ready-to use language descriptions for use with DOS, DBASE, or LOTUS. When used with DragonKey, these language files allow the user to speak certain DOS or Dbase commands rather than typing them. This utility can be very useful for applying voice in a limited fashion to the more popular commercial data management packages. DragonKey, however, is not very flexible, and occupies almost 200K of the limited memory available to a PC.

While DragonLab and DragonKey provide the user with an easy way to learn and apply voice recognition, they are limited in what they can do and not easily modifiable. VoiceScribe, however, also has a library of low-level voice board functions, known as the Speech Driver Interface (SDI), that can be accessed by either MicroSoft C or Lattice C. These functions allow a software designer to develop customized speech recognition packages by managing the speech driver directly from within a program written in C. The SDI functions allow a designer to manipulate the recognition process as necessary, build specialized training routines, or modify the system's paramaters in order to improve performance. For example, VoiceScribe's default level of confidence for recognition is set at 50%. For very small vocabularies of dissimilar words a confidence level of 40% might be more appropriate, while 60% or more might be better for larger vocabularies with similar words. While the confidence level can be set as high as 100%, such a level would never in practice be used, since it would result in a very low recognition rate. Using the SDI functions, confidence levels for recoginition can be set at levels most appropriate for a particular application, or modified during the course of an application to optimize recognition. Access to the speech board's recognition paramaters also makes it possible for an application to monitor the recognition process, and provide helpful feedback to the user as necessary. This ability to control the speech driver's recognition process greatly increases the number of possible applications to which VoiceScribe can be applied.

VoiceScribe possesses a few other desirable qualities. The speech board can be easily adjusted to accept input from a variety of different sources, including different microphones, telephones, headsets, and tapes. While designed primarily for use with one user at a time, the word models are adaptable, so do allow for the possability of a certain level of user independence. When word models are fixed in form, they can be built from only one person's voice patterns, and cannot be modified once they have been created. Adaptable models, on the other hand, can be built originally using one person's voice, and then modified to include other peoples' patterns. The adaptation of the word template averages the different voice patterns, creating one overall model for the word. For small vocabularies, adaptive models can be used to develop speaker independent recognition systems. Adaptive models are also useful for single user systems. Since a user's speech patterns will change gradually over time, fixed word models may lead to a gradual deterioration in the recognition process. With adaptive models, periodic retraining of the words can be done in order to maintain an up-to-date model of the user's voice. Neither DragonLab nor DragonKey are designed for adaptive training, but adaptive training routines can be built using the SDI functions. The SDI functions also allow for a limited use of expanded memory if it is available.

3.4 Software Considerations

The choice of programming languages for the system was restricted to a certain degree by the hardware being used. VoiceScribe's development utilities, DragonLab and DragonKey, did not provide the low level of control needed for the system's implementation, while the Speech Driver Interface, SDI, did. In order to make use of SDI's functions, Microsoft C was chosen. "C" is a powerful language used frequently for system development. Unfortunatly, it does not possess a high level of screen management facilities, and is not easily integrated with the more popular database management systems used on PCs. Another programming language, Arity Prolog, was therefore chosen to provide the screen managment and database interface capabilities. Prolog is a predicate calculs based language that is very popular for use with natural language systems. Arity's implementation of Prolog was particularly useful because of its flexibility. Arity Prolog comes with both an interpreter and a compiler. The Arity system allows for functions written in other programming languages, such as C, Pascal, or Assembler, to be added to the interpreter. In this way, useful functions that do not already exist in Prolog, such as those for the speech driver interface, can be built into Prolog. Arity Prolog also possesses a fairly high level of screen management capabilities, and interfaces with several data managment packages.

One of the more interesting things about Prolog is that there is no physical or conceptual separation between a Prolog program and a Prolog database. A program in Prolog consists of a collection of clauses and predicates. A database in Prolog consists of a collection of clauses, which can be accessed and manipulated directly by the Prolog program. The clauses in a Prolog database closely resemble the structure of a relational database. Because of this similarity, it is possible to view a Prolog database as an extension of the relational model, and to develop interfaces between Prolog and relational systems. Arity Prolog makes use of this similarity to provide interfaces to both SQL and DbaseIII, two of the more popular relational systems.

3.5 System Integration

The system being developed consisted of a PC-AT, the VoiceScribe recognizer, Microsoft C, and Arity Prolog. The various elements of the system were integrated in the following fashion. In order to allow for the direct manipulation of the speech recognizer from within Prolog, low level speech driver functions were written in C, making use of VoiceScribe's SDI library. These functions were then compiled, and added to the Prolog interpreter, using Arity's

C interface. The screen management routines, the heart of the NLMenu system, were written in Prolog. The database used in the initial system prototype was a Prolog database, although the data contained in it could have been imported from either SQL or from DbaseIII.

A conceptual diagram of the system's main components is given in Figure 3.1. The system's major component is the system coordinator, or kernel. Written in Prolog, this module controls the voice board's movement through its ATN, using the C functions that were added to Prolog. As the user speaks, the kernel monitors the speech board's recognition process, accepting words that surpass a minimun confidence level, and displaying them on the NLMenu screen. If a word is not recognized, the kernel examines the information provided by the recognizer, and displays an appropriate message on the screen. As words are recognized, the kernel moves the voice board's ATN to its next state.

The system coordinator also looks after the highlighting of the NLMenu screen, syncronizing the menu and the voice board's ATN. When the NLMenu system signals the need for an "expert" menu, the system coordinator extracts the required data from the database, and presents it on the screen. When a complete query has been entered, the system coordinator converts the natural language query into a

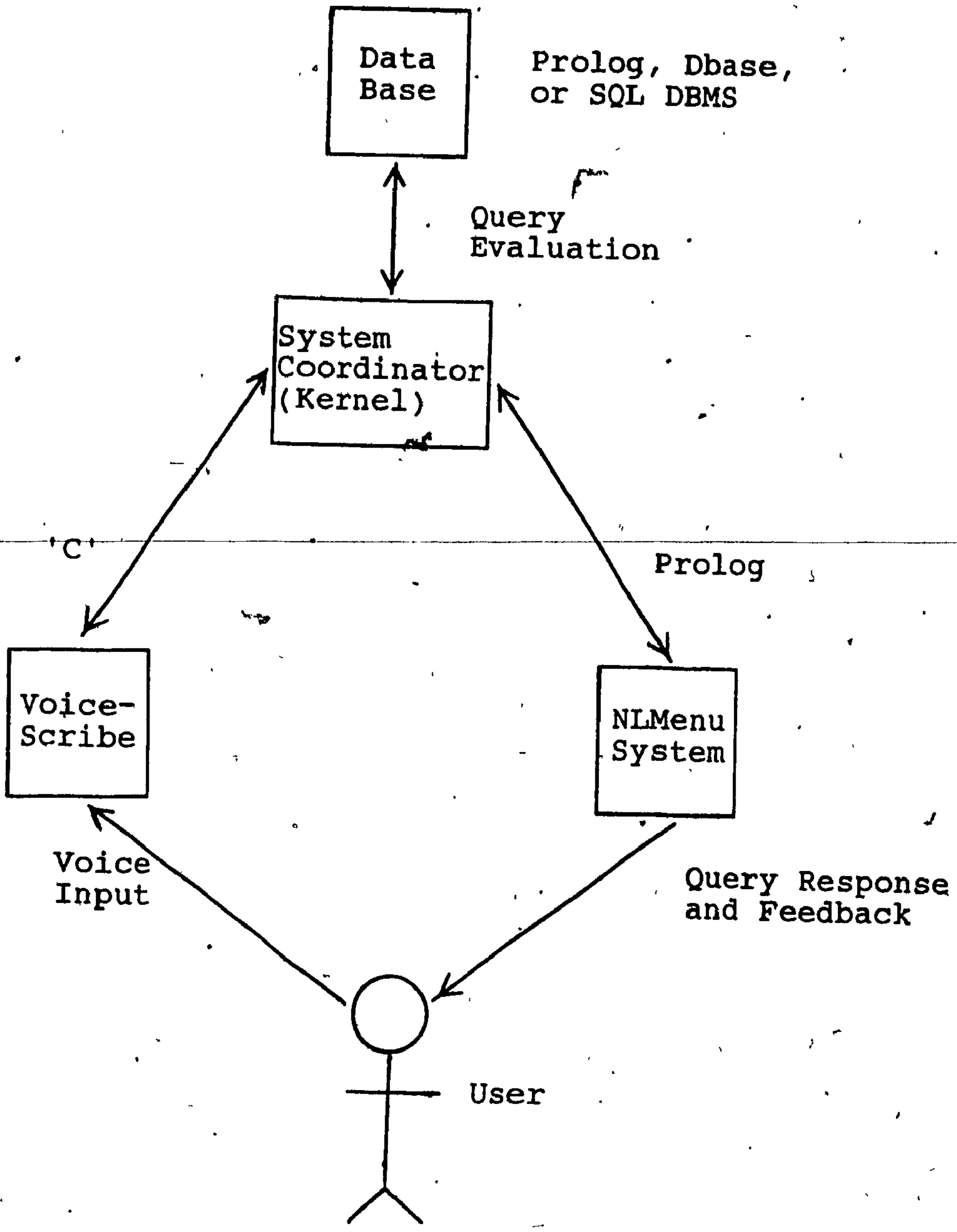

Data Base
Prolog, Dbase, or SQL DBMS
Query Evaluation
System Coordinator (Kernel)
'C'
Prolog
Voice-Scribe
NLMenu System
Voice Input
Query Response and Feedback
User

Figure 3.1. Conceptual Diagram of System

database query, and sends the query to the database. The response to the query is then presented to the user on the screen. If at any point during the formulation of the query the user decides to modify the query, s/he can simply say "BACKUP". When the kernel recognizes this word, it will move the ATN back to its previous state, move the highlighting of the NLMenu screen back to its corresponding prior state, and remove the last word from the displayed query.

Chapter IV

Implementation of NLMenu Interface Using Voice Input

4.0 Introduction

In the last chapter we looked at the different hardware and software technologies that were used in the development of a voice driven NLMenu based interface to a database. The reasons for chosing each of the various components were discussed, along with the approach taken to integrating them into one coherent system. In this chapter we will discuss the actual design and implementation of the system prototype. The considerations and techniques involved in the design process will be reviewed. The contents and structure of the different modules of the conceptual diagram (Figure 3.1) will be discussed, and examples of a typical interaction with the system given. Finally, we will consider the steps that would be required in order to adapt the current system to another database.

4.1 The Database Description

While the database contents is not crutial to the functioning of the system, the database structure is used heavily during system development, so knowing the domain used in the prototype will help clarify the examples and discussions that follow. The database domain used in the development of the system was a small subset of a university database domain. In particular, the database contains data concerning instructors, courses, and topics in a Computer Science Graduate Studies programme. The database consists of seven relations. The three primary relations are those containing the data for the instructors, courses, and topics, while the four remaining relations describe relationships existing in the domain. The structure and attributes of the seven relations are given in Figure 4.1. Conceptually, any relational database system could have been used for setting up the databases. For reasons of simplicity, however, they were actually implemented as sets of Prolog clauses. Examples of these Prolog database clauses are given in Figure 4.2.

4.2 Screen Design

The NLMenu screen is a set of boxes containing the different words and phrases that make up valid natural language queries over the database domain. For the system prototype that was developed, the screen menu is made up of six different boxes, representing commands, attributes, nouns, connectors, modifiers, and specific database values (the "expert" menus). All queries in the system end with a

Relation Name	Attributes	Comments
inst	instructor # instructor name phone # office #	Database containing the main information concerning instructors.
crs	course # course name hour	Database contining the course number, name, and starting hour.
topic	topic # topic name	Database contining topic number and name.
teach	course # instructor #	Database linking courses and instructors.
int	instructor # topic #	Database linking instructors and their research interests.
rel	course # topic #	Database linking courses and topics.
preq	course # prerequisite #	Database linking courses and their prerequisites.

Figure 4.1 Description of Databases

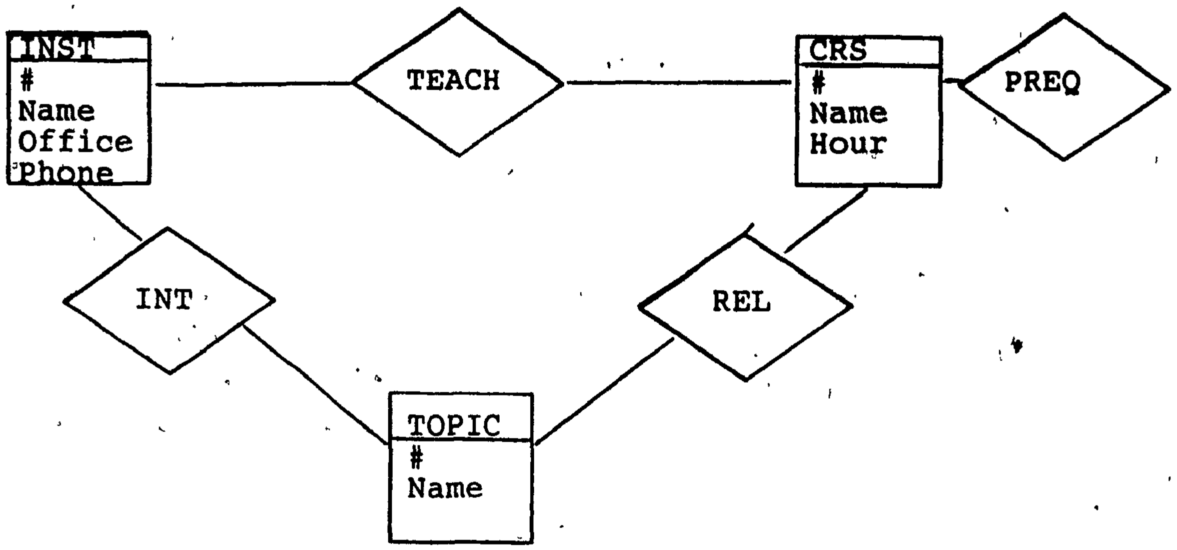


Figure 4.3 Entity-Relationship Model of Database

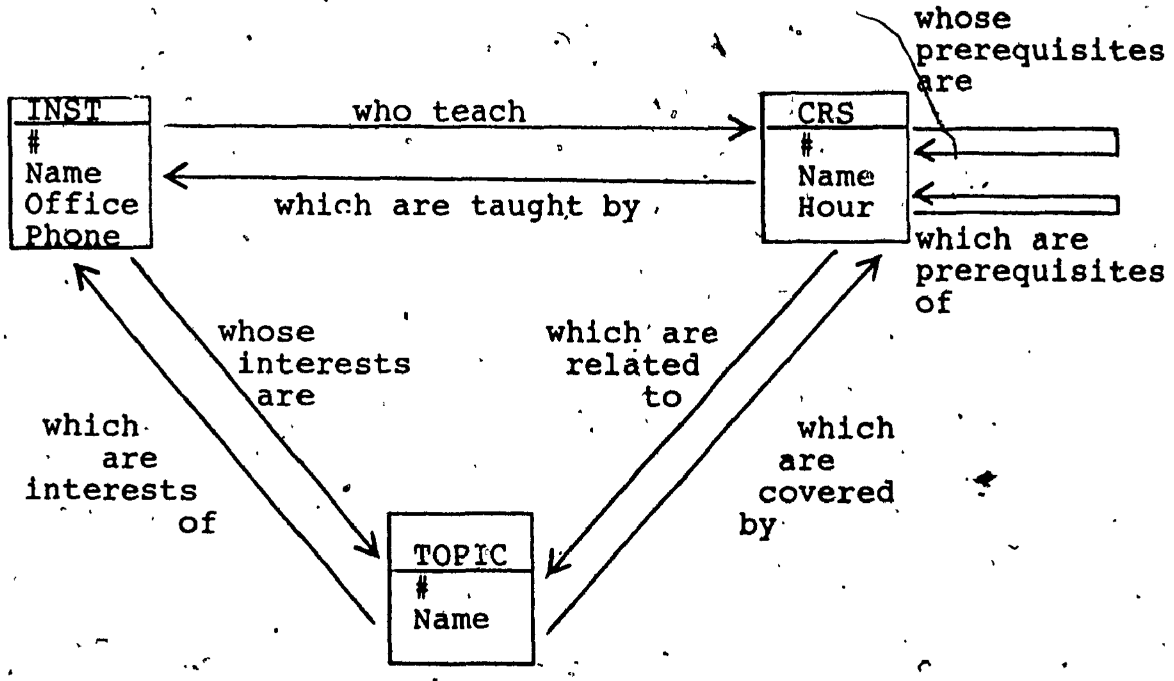


Figure 4.4 Annotated Entity-Relationship Diagram

```
inst(01,Alagar,848-1234,H961)
inst(02,Atwood,848-2345,H961)
inst(03,Boom,848-3456,H961)

crs(Comp627,Microprocessor System Arch,16:00)
crs(Comp675,Intro to Man-Machine Comm,18:00)
crs(Comp773,Seminar in Man-Machine Comm,20:30)

topic(01,Computer Systems)
topic(02,VLSI Architecture)
topic(03,Database and Information Systems)

teach(Comp627,18)   preq(Comp627,Comp525)
teach(Comp773,22)   preq(Comp773,Comp772)

rel(Comp627,01)     int(01,12)
rel(Comp675,15)     int(02,04)
```

Figure 4.2 Sample Prolog Database Clauses

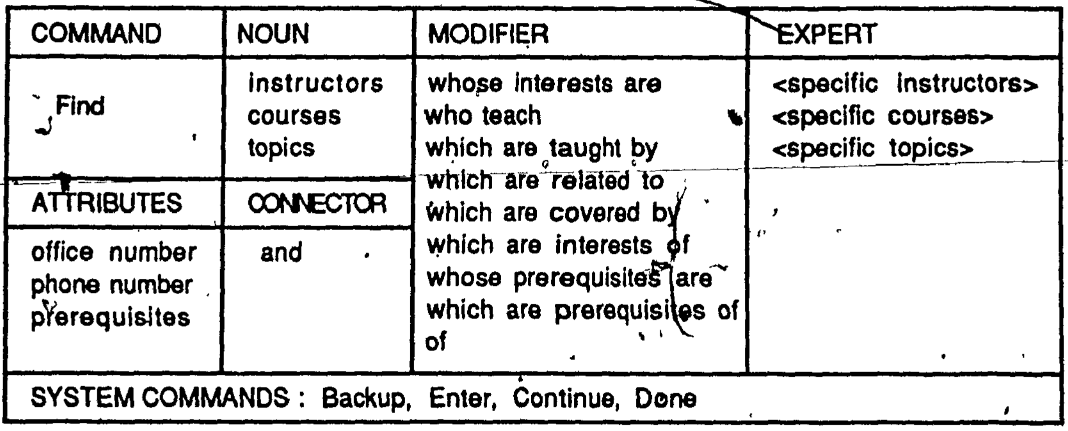
specific database value. The actual process of screen design can be decomposed into three steps:

1) define all valid queries to be used in the database;
2) break the queries into their constituant parts;
3) place these parts in the appropriate box on the screen.

These three steps can be done quite easily by viewing the database from the perspective of entity-relationship (ER) modeling. When viewed as an ER model, the three primary relations in the database used in this system become three entities, while the four remaining relations become relationships between the entities. An ER model for the database is given in Figure 4.3.

Once the ER model of the database is done, all of the valid queries can be enumerated by simply describing each of the relationships in natural language. For example, the relation <u>teach</u> represents a relationship between the entities <u>instructors</u> and <u>courses</u>. This relationship can be described as either "instuctors <u>who</u> <u>teach</u> courses", or "courses <u>which</u> <u>are</u> <u>taught</u> <u>by</u> instructors", with the relationship descriptors underlined. By continuing this process for all of the relationships in the ER model, relationships such as "courses which are related to topics" and "topics which are interests of instructors" can be described. Figure 4.4 gives the annotated ER diagram for the database. From this we can ennumerate queries such as "Find instuctors who teach <specific course>", or "Find courses which are related to <specific topic>".

After all of the relationships have been described in natural language, the various boxes of the NLMenu screen can be filled in. In general, the names of the entities are placed in the NOUN box, and the relationship descriptors go in the MODIFIERS box. Natural language synonyms of any non-key attributes of the entities are placed in the ATTRIBUTE box. The "expert" box represents specific instances of the entities, so the entity names are placed in this box. Commands such as "list", "print", or "find" are placed in the COMMAND box. Finally, connectors such as "and" are placed in the CONNECTOR box if more than one attribute is to be requested in a query. The NLMenu screen built from the ER diagram (Figure 4.4) is shown in Figure 4.5.



COMMAND	NOUN	MODIFIER	EXPERT
Find	instructors courses topics	whose interests are who teach which are taught by which are related to which are covered by which are interests of whose prerequisites are which are prerequisites of of	<specific instructors> <specific courses> <specific topics>
ATTRIBUTES	CONNECTOR		
office number phone number prerequisites	and		
SYSTEM COMMANDS : Backup, Enter, Continue, Done			

Figure 4.5 NLMenu Screen

4.3 Language Grammar Design

The NLMenu screen presents the user of the system with all of the words and phrases that can be used in constructing queries about the database in use. In order to allow the user to speak his/her query, a parallel grammar must be developed for use with the VoiceScribe recognizer.

The VoiceScribe documentation clearly explains the grammar rules that are used for writing languages for the recognizer, so these rules and techniques will not be repeated here. In brief, the grammar written for use with the NLMenu system allows the user to form only those queries that are permitted by the screen, and movement through the different states of the grammar parallels exactly the movement through the NLMenu screen. The complete VoiceScribe grammar used for the system prototype is presented in Appendix 1.

4.4 Language Training

Once the grammar for the NLMenu screen is written, it is compiled into a finite state machine similar to an ATN, giving a Language Description File (LDF) that is used by the recognizer. Before the system can be used all of the words in the language must be trained, and the user's word models stored in a vocabulary file. The training can be done using either of VoiceScribe's development utilities, DragonLab or DragonKey, or with a separate training routine that we had written in C for use with the system prototype.

4.5 System Implementation

In the previous section the design of the NLMenu screen and the VoiceScribe grammar were discussed, both based on the domain of the database being used with the system. In this section the implementation of the complete working system will be considered, and the management of the various system modules discussed.

4.6 The System Coordinator

The System Coordinator, or kernel, is the heart of the system. Written in Prolog, with added C functions for managing the recognizer, it is responsible for the initialization of the system, management of both the screen and the voice board, and calls to the database management system. Each of these kernel functions are outlined below.

4.7 System Initialization

The system initialization consists of several steps. The first is to prepare the voice board for recognition. This is done by creating a task for the voice board, allocating memory for use of the recognizer, and loading the language description and vocabulary files into the recognizer's memory space. Since this is a single user system, the user is asked for the name of his/her vocabulary file. If the file does not exist, or if all of the words have not yet been trained, the system initialization is terminated and the user told to train the words before continuing.

Once the recognizer has been successfully initialized, the kernel sets up the NLMenu screen. The information and routines necessary for drawing the screen are not contained within the system coordinator, so are loaded from a seperate Prolog file named BOX. This file contains the screen coordinates of the NLMenu boxes, character strings containing the words ~~and~~ phrases that appear on the screen, the screen coordinates of these words, and the Prolog predicates used for implementing pop-up windows. The system coordinator uses this information to present the initial menus on the screen, and then removes most of the code contained in BOX from the system's memory space.

The system coordinator then prepares the pop-up "expert" windows that are used during the formulation of queries. These windows present the user with the key attributes of all instances of the database entities, such as course number and name or instructor number and name. Rather than retrieving this information from the database management system whenever an expert menu is called for, the kernel loads the clauses for the three database entities into the Prolog work space, and stores the key attributes in Prolog database form. Once the key attributes have been recorded, the clauses for the three entities are also removed from the work space.

The kernel then loads the information needed for the highlighting of the active words during the formulation of queries. Since the movement of the screen highlighting must parallel exactly the movement through the recognizer's grammar, this information is stored in a series of Prolog clauses that associate the screen coordinates for the first letter of each word with the corresponding state number from the recognizer's ATN. Since there can be a fair number of states in the ATN, the screen coordinates for each word are recorded in a Prolog B-tree according to the state number, in order to allow for more rapid retrieval. Once the B-tree has been set up, the Prolog clauses containing the screen coordinates are unloaded.

4.8 Recognition

After the System Coordinator has successfully initialized the system, it is ready to begin the recognition of the user's query. Recognition starts at the initial state of the ATN, with the first letter of the first word, FIND, highlighted on the screen. The user then formulates the query word by word, following the highlighting of words on the screen, until a complete query has been entered. As the system prepares to recognize a word, the system coordinator takes the current state number from the ATN, and searches the B-tree for the screen coordinates of the words to highlight. After highlighting the appropriate words on the screen, the kernel sends a command to the recognizer to begin listening, and calls the recognize function. This function is sent the number of the state in the ATN to be recognized, and returns the word that was recognized, the next state on the ATN, and flags that indicate errors or the end of a query. After recognizing a word, the listening process is suspended. If the word that was recognized is not "BACKUP", the system coordinator writes the word to the screen. The B-Tree is then searched again for the coordinates of the words from the state just recognized, and the highlighting removed from the screen. If the word recognized is "BACKUP", the system coordinator erases the last recognized word from the screen, removes the highlighting, and retrieves the previous state number from a stack. This previous state now becomes the new state. The entire highlighting and recognition process is then repeated for the new state.

After each word has been recognized, the system coordinator looks ahead in the ATN to see if the next state requires an expert window. As mentioned before, the "expert" window presents the user with the key attributes of instances of the database entities. Because database entries can change quite frequently, the system is not designed to recognize the actual names of instructors or course titles. Instead, the user chooses the desired database value by saying the number associated with the name. In order to see if the next state is an "expert" state, the kernel simply examines the ATN to see if the next words to be recognized include numeric values. If the next state does include numbers, the coordinator calls the routines to pop-up the appropriate "expert" menu. While an expert menu is on the screen, the system coordinator looks ahead in the ATN for the end of the expert states. When the coordinator sees that the next state on the ATN is not an expert state, a routine is called to un-pop the expert window and restore the initial screen.

The recognition process continues, word by word, until the ATN reaches a final state. When a final state is encountered, the system coordinator reads the course, topic or instructor number from the screen, and replaces it with the corresponding course, topic or instructor name. At this point a complete query has been entered by the user. The user is then presented with three options: send the query to the DBMS, enter a new query, or terminate the session. If the user chooses to have the query evaluated, the system coordinator loads the Prolog file DBMS, which contains the database clauses, and the predicates used to evaluate the query. Since the query at this point is in natural language, it must be transformed into a Prolog query before the query can be evaluated. This process has two basic steps. The first is to issue a call to a speech board function that checks the list of completed ATN productions. This function returns to the DBMS the number of any modifier that was used in the query, and a flag indicating whether or not an attribute is present in the query. The next step in the query transformation process is to search the natural language query for actual attribute names, if any were used, and to extract the database key value from the end of the query. At this point the various components of the query are known to the DBMS, and are put together to form a Prolog query. This Prolog query is then evaluated, and the answers presented to the user on the screen. After the user's query has been answered, the system coordinator unloads the entire DBMS from the system's memory space. The user may then enter another query, or terminate the session. Examples of the utilization of the system prototype are given in Appendix 2.

4.9 System Performance and Limitations

In earlier chapters some of the problems encountered by traditional natural language interfaces were discussed. Among these problems were linguistic and conceptual overreach. When speech recognition is added to an NLU system a related problem, that of phonetic overreach, can also be encountered. One of the main ideas behind the development of NLMenu was that presenting the user with a clear picture of all possible query formulations would eliminate to a large degree the various problems of overreach. In order to see whether or not this hypothesis was true, a series of informal tests were carried out using ten subjects and the NLMenu prototype. The form and results of these tests are summarized below.

The first step in the testing was to develop a vocabulary file for each subject. This was done using the training utility, written in C, that was developed for use with the system prototype. The next step in the test was to ask each user to formulate queries using three seperate versions of the prototype. The first version offered the user no assistance in query formulation, presenting a completely blank screen. Before the subject attemped to ask queries using this version of the prototype, the query and database structures were explained, and examples of valid queries were given. The second version of the system gave the user a clearer idea of proper query formulation by presenting the initial NLMenu screen, but without any highlighting of words. The third version used was the whole NLMenu system, with screen, highlighting, and pop-up menus. As each subject attempted to ask a query, the confidence level for each utterance was recorded, and an average confidence level was calculated for each query. A confidence level of zero was given to any utterance that did not match an active word at any given state in the grammar.

The testing showed that the problem of overreach was most noticeable with the first version of the prototype. Common errors included the use of articles such as "the" before attributes, or the use of "professor" instead of "instructor". In general, formulation of valid queries without the screen menus was not easy, with average confidence levels for queries ranging from 40% to 69%, with an average for all subjects of about 58%. Overreach was also found during the testing of the second version of the system. Despite the presentation of the menus, without highlighting queries such as "find courses which are interests of..." were attempted, eventhough courses are not considered "interests" of instructors in the database domain. However, the confidence levels for queries were considerably higher when the menu was displayed, ranging from 62% to 82%, with an overall average of about 68%. When highlighting was added for testing of the final version the problem of overreach became negligable, and none of the ten subjects had difficulty forming queries. Confidence levels ranged from 80% to 95%, with an average for all subjects of 85%. While the tests carried out cannot be considered statistically rigorous, they clearly demonstrate that the use of the NLMenu approach, with highlighting of active words, does eliminate many of the problems of traditional natural language interfaces, even when voice is used as the method of query input. The averages for the ten subjects, and the overall averages are presented in Appendix 3.

While the NLMenu approach to natural language database access may indeed resolve some of the problems encountered by traditional natural language interfaces, it is not a panacea for all of the problems faced by NLU. NLMenu suffers from several inherent problems of its own that severely restrict its applicability to a major portion of existing database installations. One of the major limitations of the NLMenu approach is the restriction placed on the size of the interface by the screen display. Since the idea behind the NLMenu approach is to present the user with a a clear view of all possible query formulations, only a small database domain can be accessed using this type of interface. Apart from the restriction on domain size, the screen also restricts the number of tuples that can be reasonably represented in an expert window. The scrollable expert windows used in the system prototype can alleviate this problem somewhat for small numbers of tuples, but would not be practicle for presentation of a database containing hundreds or thousands of tuples.

The primary benefit of adding voice recognition to the NLMenu interface is that it provides "keyboard-free" input of queries, making the interface easier to use for many users. The use of voice also presents several problems, the foremost of which is the need for training, since only trained users can use the system. Training itself can be a problem. During the testing of the system, many of the subjects found the training process very long and dull, even though training of the entire vocabulary took less than five minutes to complete. The limitations imposed by training could perhaps be overcome by development of user-independent vocabularies for interfaces, but this would only be practical for small vocabularies, and could even further restrict the size of the database domain.

4.10 Adaptation to Another Database Domain

In Chapter 1 the issue of portability of natural language systems to new database domains was discussed. One system discussed, LIFER/LADDER, was not easily portable because its natural language component was domain dependent. The COOP system, on the other hand, was more easily portable due to the high level of domain independance of its natural language component. Like the LIFER/LADDER system, the natural language component of the NLMenu system prototype is very domain dependent. The NLMenu screen designed is primarily based on the database structure, and the corresponding grammar for speech recognition is based entirely on the screen contents. In the case of the NLMenu prototype, however, the domain dependance of the system does not mean that the system cannot be easily adapted to another database domain. Due to the small size and limited complexity of the system, adaptation to another domain can be carried out without too much difficulty. The steps necessary to adapt the NLMenu interface to a new database environment are discussed below, using the database and contents of the original system as a reference. These steps do not include any changes that will be required to the actual database evaluation process. In the original prototype the database and DBMS were both written in Prolog, and managed the translation and evaluation of the query when called by the System Coordinator. This approach was chosen because of its ease and simplictity, rather than for reasons of practicality. A more useful approach may have been to have the queries translated for use by a separate DBMS, such as SQL or DBASE, that would have been responsible for the actual query evaluation.

Adapting the NLMenu interface to a new database domain basically consists of redesigning the system's screen and VoiceScribe's grammar, and making minor modifications to the

System Coordinator and to a few of the recognizer functions. The redesign of the screen and grammar follows the same methodology as the initial screen and grammar design discussed previously. Assuming a relational database management system such as SQL or Dbase, or a Prolog database, the first step in redesigning the screen is to build an entity-relationship model of the database domain. An ER model for a database can be easily designed be viewing the major database relations as entities, and the relations that link them as relationships, as described above in Section 4.2. A complete description of ER modeling techniques can be found in Chen (9).

Using an ER diagram of the new database domain, the next step is to describe the different relationships in natural language, and formuate the natural language queries that will be presented on the screen. Examples of this query design process are also given in Section 4.2. It is helpful to actually add the various word that are to be used in formulating the queries to the ER diagram of the domain, as was done in Figure 4.4. It is important at this point to keep in mind that all of the queries in this system end with actual database values, and to formulate the NL queries accordingly. This restriction allows queries such as "Find students who are advised by <specific instructor>", but does not permit the equivalent "Find students who have <specific instructor> as their advisor". This restriction can be removed from the system in order to allow different query formulations, but the modifications needed for changes are beyond the scope of this report. The relationship descriptors, or modifiers, should also be kept as short and precise as possible, since the size of permissible queries is limited by the size of the screen menus. A query such as "Find parts which are used in the constuction of <specific object>" would not fit on the screen, whereas the equivalent "Find parts which are components of <specific object>" would not exceed the screen limitations. The screen size also limits the number of modifiers that can be presented on the screen to no more than about fifteen. If the database domain contains more than fifteen relationships, it may be necessary to design more than one NLMenu interface for the domain, or to not include all of the possible query formulations on the screen.

After the relationships in the database domain have been identified in natural language and the corresponding queries formulated, it is possible to put the screen together. If the new database domain does not contain more modifiers, nouns, or attributes than the initial system prototype, the size of the various boxes that make up the NLMenu screen will not have to be modified. On the other hand, if the new domain has a larger number of any particular language component than the prototype, it may be necessary to expand the screen's boxes. This can be done by changing the screen coordinates of the various boxes contained in the Prolog program file BOX.ARI. The words that are to appear on the screen are also contained in the file BOX, along with their screen coordinates. Adapting the various words and phrases that appear on the screen to reflect the new domain can be done by editing the character strings stored in the BOX program file.

The actual positioning of the words on the screen can be determined from their positions on the annotated ER diagram of the database domain. The entities presented in the diagram represent the nouns and expert values to be displayed on the screen, so the specific entity names should be placed in the NOUN and EXPERT boxes. The relationship descriptors from the diagram, such as "who are advised by" or "which are components of", are placed in the MODIFIER box of the screen. The non-key attributes of the entities are naturally placed in the ATTRIBUTE box of the screen. Since the key attributes of the entities are used for choosing a particular database record through the pop-up "expert" windows, they are not placed in the ATTRIBUTE box. All non-key attributes are not necessarily presented on the screen.

Any database attributes that are not of importance to the queries being designed should not be included on the screen, for the simple reason of efficiency. Any connectors that are to be used in the queries are placed in the screen's CONNECTOR box. Generally, the COMMAND and SYSTEM COMMANDS boxes need not be changed when adapting the screen to a new database domain.

Once the NLMenu screen has been redesigned to present the natural language queries for new domain, the corresponding VoiceScribe grammar for the queries must be designed. The grammar for the original screen is contained in the file NLMENU.LAN, which can be accessed by any standard text editing package. This grammar begins with a root production containing the word "find", followed by a series of non-terminal sub-productions representing the various attributes, nouns, and modifiers found on the screen. The end of the NLMENU.LAN file contains a series of clauses that associate each terminal symbol, such as "one", with its corresponding output string, "1" in this case. The output string is what will actually appear on the screen when the word is recognized by VoiceScribe. The grammar for the new screen can be written by simply modifying the various productions contained in the original LAN file. Before attempting this, however, it is advisable to read the sections of the VoiceScribe user's manual that deal with grammar writing using the VOCL compiler. It is important to remember that the new grammar must parallel exactly the queries presented on the NLMenu screen.

The written grammar for the new interface must then be compiled using the VOCL compiler. If there are any errors or ambiguities in the grammar, they will be displayed on the screen during the compilation of the grammar. Successful compilation of the grammar will result in the creation of two new files, with LDF and LST extensions respectively. The LDF file, or language description file, contains the ATN-like finite state representation of the grammar that will be used by VoiceScribe during recognition. The LST file gives a state-by-state listing of all transitions in the grammar. This rather lengthy listing will be necessary for the final system modifications. Before proceeding to these modifications, however, it is important to train and test the grammar. Training can be most easily done using the DragonLab utility. Following training, the grammar should be tested to make sure that all of the queries can be formulated in the proper fashion, and that no other formulations are permitted by the grammar. This testing can also be done using DragonLab.

As mentioned earlier, the System Coordinator uses information extracted from the recognizer's ATN to manage certain system functions, such as popping up the expert windows and controlling the screen highlighting. This information is primarily in the form of state numbers that represent particular locations in the ATN, and can be most easily found by viewing the LST file produced during the compilation of the VoiceScribe grammar file. Since the state numbers for the new grammar will not be the same as those for the original grammar, the last steps needed to adapt the system to the new database domain involve changing any references to the ATN state information in the System Coordinator and its associate functions. The screen highlighting functions depend heavily on state information, so will require major modifications. These modifications are easy to do, but may be time consuming and rather tedious. The highlighting information is stored in the file DB.ARI in a set of state clauses of the form:

```
state(1,4,4).
state(2,9,1).
state(2,10,1).
```

For example, the state 1 clause represents the word FIND, and highlights the letter at screen coordinates 4,4, while the state 2 clauses highlight the first letters of the various NOUNS and ATTRIBUTES. In order to adapt these clauses to the new screen and grammar, one must work through the state changes listed in the LST file while keeping track of the corresponding screen coordinates of the words active at each state. Once all of the new state/coordinate clauses have been listed, they can be used to replace the original clauses in DB.ARI. A quick viewing of the original state clauses will reveal three clauses with a state number of 999. These clauses do not correspond to an actual state number on the ATN. They are used simply to store the screen coordinates of the system commands ENTER, CONTINUE, and DONE. In the unlikely event that the new ATN actually contains more than nine hundred and ninety nine states, a different number will have to be used here.

Apart from the information necessary for highlighting the NLMenu screen, there are a few other references to the ATN used by the System Coordinator that require modification. The first of these can be found in the LOAD.ARI program, which makes up the body of the System Coordinator. The LOOP predicate found in this program file contains a recognizer call, RECOG, that is used for recognizing the system commands ENTER, CONTINUE, and DONE. Since these words are not part of the grammar's root production, the recognizer must be provided with the actual identification number of the production in which they are to be found. In the original system prototype, this identification number was 77. The new identification number for the system commands can be found in the LST listing of the grammar at the very end of the section entitled SYSTEM INFORMATION, and should replace the id number found in the original RECOG function call.

Production id numbers are also used in determining when the "expert" pop-up menus should be displayed on the screen and removed. As each word is recognized by the System Coordinator, the LOOKAHEAD1 predicate is called to see if the next production is an expert production. While there is an expert window on the screen, LOOKAHEAD2 is called to check for the end of the expert production. Both of these predicates make use of a recognizer function LISTSYMS to examine the next symbols on the ATN. LISTSYMS is written in C, and can be found in the ICPRO.C program file. LISTSYMS checks the list of id numbers for next symbols to be recognized. If the list contains the id for the word COMP, a value of 1 is returned to the LOOKAHEAD1, signaling the need to pop up the course expert window. A value of 2 is returned for instructors, and 3 for topics. If the list of symbol id numbers does not contain the id's for any numeric symbols, the expert production is finished, and LISTSYMS returns a value of 4 to LOOKAHEAD2. To make the NLMenu interface function for the new database domain, the id numbers that LISTSYMS looks for will have to be modified within the C function to reflect the new grammar. Changes to the values returned by LISTSYMS and to the Prolog code that interprets the values, may also be necessary, depending on the number of entities in the new domain.

The only remaining modifications to be made are domain dependent, so cannot be outlined precisely. As discussed earlier, the clauses containing the database entities are loaded into Prolog's internal database during the system initialization process for use in the pop-up windows. The actual loading of the clauses is done by the LOAD_CRS, LOAD_INST, and LOAD_TOPIC predicates, which are contained in the BOX.ARI program file. These predicates will have to be modified, replaced, or expanded, as necessary, to accomadate the entities present in the new database domain. Along similar lines, the CRS_UP, INST_UP, and TOPIC_UP predicates in LOAD.ARI, which are called by LOOKAHEAD1 to initialize the pop-up windows, will have to be modified to fit the new domain. Finally, the SWITCH_INST and SWITCH_TOP predicates, which replace the entity numbers with the entity names on the screen, will require modification. These predicates can also be found in LOAD.ARI.

After all of the necessary modifications have been made to the system, the adapted interface should be ready to accept natural language queries that can be then sent off to whatever DBMS is being used with the system for evaluation. While the actual adaptation process may seem rather long and arduous, in fact it can be done in relativly little time. In one trial adaptation, the original system prototype was modified not only to work with another database domain, but to also accept input in another language, French. Due to the change in natural language, the NLMenu screen and VoiceScribe grammar had to be severly modified to reflect a new grammatical structure. The entire adaptation process, carried out by someone with a good understanding of Prolog, but little knowledge of either VoiceScribe or C, took less than three days to complete.

Chapter 5

Conclusions

In this report we have looked at some of the general problems facing the development of natural language interfaces to database systems, and reviewed several aspects of currently available speech recognition technology. Neither natural language interfaces nor speech recognition systems have reached the level of development where they can provide easy and unrestricted access to databases for the truely casual user. It is, however, possible to couple natural language interfaces and speech recognition systems in order to provide a fairly easy method of data access for the relativly casual user.

The development of a natural language interface using voice has been discussed. The NLMenu/VS system was built with the goal of showing that existing NL interfaces could be integrated with affordable speech recognizers to provide a query system that was both easy to learn and to use. The prototype discussed in this paper is very limited in its scope, but was not designed for large database systems, and can be quite easily adapted for use with different database domains and various database management systems. The system prototype has demonstrated the potential power of isolated word recognition in conjuction with menu based natural language interfaces. Such a system is very practical for use with many of the small database management systems available for use with personal computers.

References

1. Hendrix, G.G. "Developing a Natural Language Interface to Complex Data". ACM Transactions on Database Systems, Vol. 3, No. 2, June 1978, pp. 105 - 147.

2. Kaplan, S.J. "Designing a Portable Natural Language Database Query System". ACM Transactions on Database Systems, Vol. 9, No. 1, March 1984, pp. 1 - 19.

3. Kaplan, S.J. "Appropriate Responses to Inappropriate Questions". Elememts of Discourse Understanding, Cambridge Press, 1981, pp. 127 - 144.

4 Tennant, H.R. "Menu-Based Natural Language Understanding". Proceedings of 21st Meeting of the Association for Computer Linguistics, 1983, pp. 151 - 158.

5. Thompson, C.W. "Building Usable Menu-Based Natural Language Interfaces to Databases". Proceedings of 9th International Conference on Very Large Databases, 1983, pp. 43 - 55.

6. Tennant, H.R. "Menu-Based Natural Language Understanding". Proceedings of AFIPS Conference, 1984, pp. 631 - 635.

7. Parsons, T. "Voice and Speech Processing", McGraw Hill, 1987.

8. Radahakrishnan, T. "Voice Inquiry Systems", Proceedings of 1988 Computer Society of India Conference, Madras, India, MacMillan Publishing.

9. Chen, P. "Entity-Relationship Approach to System Analysis and Design, North Holland, 1980.

10. Allen, J. "A Perspective on Man-Machine Communication by Speech", Proceedings of IEEE, Vol. 73, No. 11, November 1985, pp. 1541 - 1550.

11. Zue, V. "The Use of Speech Knowledge in Automatic Speech Recognition", Proceedings of IEEE, Vol. 73, No. 11, November 1985, pp.1602 - 1615.

12. Chen, Y. "Cepstral Domain Talker Stress Compensation for Robust Speech Recognition", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 36, No. 4, April 1988, pp. 433 - 439.

13. Stern, R. "Dynamic Speaker Adaptation for Feature-Based Isolated Word Recognition", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 35, No. 6, June 1987, pp. 751 - 763.

Appendix 1

```
/*  NLMENU.LAN

    This is the grammar used by the NLMENU system
    prototype.
    It is compiled by VoiceScribe's VOCL Compiler, giving
    a language description file, NLMENU.LDF. The LDF
    file contains the ATN-like finite-state machine used
    by VoiceScribe during recognition.  Compilation also
    produces a listing of the ATN, NLMENU.LST.
*/


#listing;

/*  The root production, with 'backup' always active */

Q [backup] = find (Q1, Q2, Q3);
Q1 = ATTRIBUTE1;
Q2 = NOUN;
Q3 = ATTRIBUTE2;

/* The nouns */

NOUN = (N1, N2, N3);

N1 = instructors INST_MOD;
N2 = courses CRS_MOD;
N3 = topics TOP_MOD;

/* The attributes */

ATTRIBUTE1 = (INST_ATT1, CRS_ATT1) ;

INST_ATT1 = ((A1 (and A2)#), (A2 (and A1)#)) of EX_INST;
CRS_ATT1 = ((A3 (and A4)#), (A4 (and A3)#)) of EX_CRS;

ATTRIBUTE2 = (INST_ATT2 ,CRS_ATT2);

INST_ATT2 =  ((A1 (and A2)#), (A2 (and A1)#)) of N1;
CRS_ATT2 =  ((A3 (and A4)#), (A4 (and A3)#)) of N2;

A1 = office number;
A2 = phone number;
A3 = prerequisites;
A4 = hours;


/* The modifiers */
```

```
INST_MOD = (M1, M2);
CRS_MOD = (M3, M4, M7, M8);
TOP_MOD = (M5, M6);

M1 = whose interests are EX_TOP;
M2 = who teach EX_CRS;
b3 = which are taught by EX_INST;
M4 = which are related to EX_TOP;
M5 = which are covered by EX_CRS;
M6 = which are interests of EX_INST;
M7 = whose prerequisites are EX_CRS;
M8 = which are prerequisites of EX_CRS;


/* The numbers for the expert menus. */

EX_TOP = (one, zero, oh) DIGIT;
EX_CRS = comp (six, seven) DIGIT DIGIT;
EX_INST = (zero, one, two, oh) DIGIT;

DIGIT = (one, two, three, four, five, six, seven, eight,
              nine, zero, oh);


/* Production for system command */

*SYSTEM = (enter, edit, continue, done);


/* Terminal symbols. The words in quotes are the
   output strings that are associated with the
   terminal. */

find             "Find ";
instructors      "instructors ";
courses          "courses ";
topics           "topics ";
office           "office ";
phone            "phone ";
number           "number ";
prerequisites    "prerequisites ";
whose            "whose ";
interests        "interests ";
are              "are ";
who              "who ";
teach            "teach ";
which            "which ";
taught           "taught ";
by               "by ";
```

```
related         "related ";
to              "to ";
covered         "covered ";
of              "of ";
comp            "Comp";
one             "1";
two             "2";
three           "3";
four            "4";
five            "5";
six             "6";
seven           "7";
eight           "8";
nine            "9";
zero            "0";
oh              "0";
backup          "backup";
yes             "yes";
no              "no";
done            "done";
hours           "hours ";
and             "and ";
or              "or ";
enter           "enter";
edit            "edit";
continue        "continue";
quit            "quit";
```

Appendix 2: System Examples

The following pages show the screen movement during the input of two sample queries. The screen highlighting is shown here by underlining. The examples show the queries formed by the following sequences of utterances: "Find office number of instructors who teach Comp 6 5 1 enter continue" and " Find courses which are prerequisites backup taught by 0 2 enter".

COMMAND	NOUN	MODIFIER	EXPERT
Find	instructors courses topics	whose interests are who teach which are taught by which are related to which are covered by which are interests of whose prerequisites are which are prerequisites of of	<specific instructors> <specific courses> <specific topics>
ATTRIBUTES	CONNECTOR		
office number phone number prerequisites	and		
SYSTEM COMMANDS : Backup, Enter, Continue, Done			

COMMAND	NOUN	MODIFIER	EXPERT
Find	instructors courses topics	whose interests are who teach which are taught by which are related to which are covered by which are interests of whose prerequisites are which are prerequisites of of	<specific instructors> <specific courses> <specific topics>
ATTRIBUTES	CONNECTOR		
office number phone number prerequisites	and		
SYSTEM COMMANDS : Backup, Enter, Continue, Done			

Find

COMMAND	NOUN	MODIFIER	EXPERT
Find	instructors courses topics	whose interests are who teach which are taught by which are related to which are covered by which are interests of whose prerequisites are which are prerequisites of of	<specific instructors> <specific courses> <specific topics>
ATTRIBUTES	CONNECTOR		
office number phone number prerequisites	and		
SYSTEM COMMANDS : Backup, Enter, Continue, Done			

Find office number

COMMAND	NOUN	MODIFIER	EXPERT
Find	instructors courses topics	whose in who teac which ar which ar which ar which ar whose p which ar of	
ATTRIBUTES	CONNECTOR		
office number phone number prerequisites	and		
SYSTEM COMMANDS : Backup, Enter, C			

01	Alagar
02	Atwood
03	Boom
04	Bui
05	Cheng
06	Fancott
07	Ford
08	Goyal
09	Grogono
10	Jaworske
11	Kasyand

Find office number of

COMMAND	NOUN	MODIFIER	EXPERT
Find	instructors courses topics	whose interests are who teach which are taught by which are related to which are covered by which are interests of whose prerequisites are which are prerequisites of of	<specific instructors> <specific courses> <specific topics>
ATTRIBUTES	CONNECTOR		
office number phone number prerequisites	and		
SYSTEM COMMANDS : Backup, Enter, Continue, Done			

Find office number of instructors

COMMAND	NOUN	MODIFIE
Find	instructors courses topics	whose ir who teac which ar which ar which ar which ar whose p which ar of
ATTRIBUTES	CONNECTOR	
office, number phone number prerequisites	and	
♦SYSTEM COMMANDS : Backup, Enter, C		

Comp627 Microprocessor System Arch
Comp628 Computer Systems Design
Comp641 Compar Study of Prog Langs
Comp642 Compiler Design
Comp645 Data Comm & Comp Networks
Comp646 System Software Design
Comp647 Software Design Methods
Comp651 DataBase Design
Comp656 Information Retrieval
Comp657 Office Automation
Comp658 Struct of Information System

Find office number of instructors who teach

COMMAND	NOUN	MODIFIER	EXPERT
Find	instructors courses topics	whose interests are who teach which are taught by which are related to which are covered by which are interests of whose prerequisites are which are prerequisites of of	<specific instructors> <specific courses> <specific topics>
ATTRIBUTES	CONNECTOR		
office number phone number prerequisites	and		
SYSTEM COMMANDS : Backup, Enter, Continue, Done			

Find office number of instructors who teach COMP651

COMMAND	NOUN	MODIFIER	EXPERT
Find	instructors courses topics	whose interests are who teach which are taught by which are related to which are covered by which are interests of whose prerequisites are which are prerequisites of of	<specific instructors> <specific courses> <specific topics>
ATTRIBUTES	CONNECTOR		
office number phone number prerequisites	and		
SYSTEM COMMANDS : Backup, Enter, Continue, Done			

Find office number of instructors who teach Comp651

Instructors who teach Comp651: Sadri H961

COMMAND	NOUN	MODIFIER	EXPERT
Find	instructors courses topics	whose interests are who teach which are taught by which are related to which are covered by which are interests of whose prerequisites are which are prerequisites of of	<specific instructors> <specific courses> <specific topics>
ATTRIBUTES	CONNECTOR		
office number phone number prerequisites	and		
SYSTEM COMMANDS : Backup, Enter, Continue, Done			

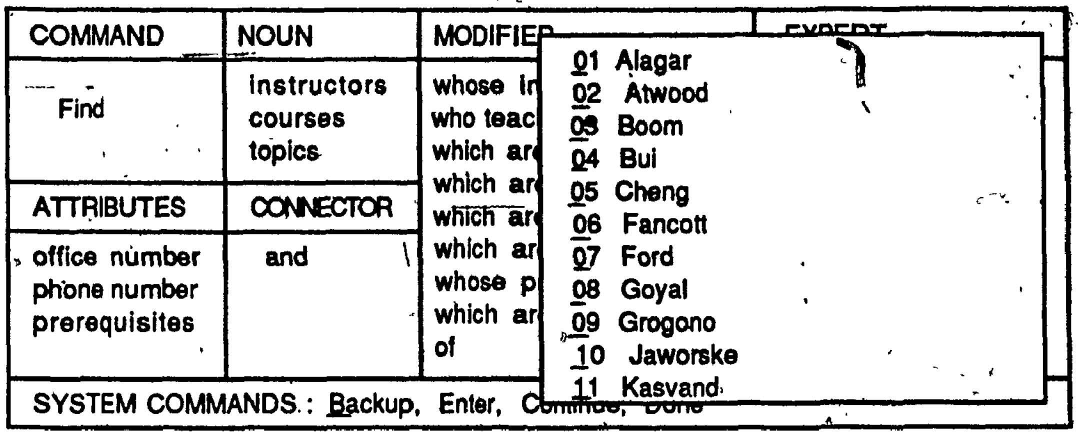
Find courses

COMMAND	NOUN	MODIFIER	EXPERT
Find	instructors courses topics	whose interests are who teach which are taught by which are related to which are covered by which are interests of whose prerequisites are which are prerequisites of of	<specific instructors> <specific courses> <specific topics>
ATTRIBUTES	CONNECTOR		
office number phone number prerequisites	and		
SYSTEM COMMANDS : Backup, Enter, Continue, Done			

Find courses which are

COMMAND	NOUN	MODIFIER	EXPERT
Find	instructors courses topics	whose interests are who teach which are taught by which are related to which are covered by which are interests of whose prerequisites are which are prerequisites of of	<specific instructors> <specific courses> <specific topics>
ATTRIBUTES	CONNECTOR		
office number phone number prerequisites	and		
SYSTEM COMMANDS : Backup, Enter, Continue, Done			

Find courses which are prerequisites

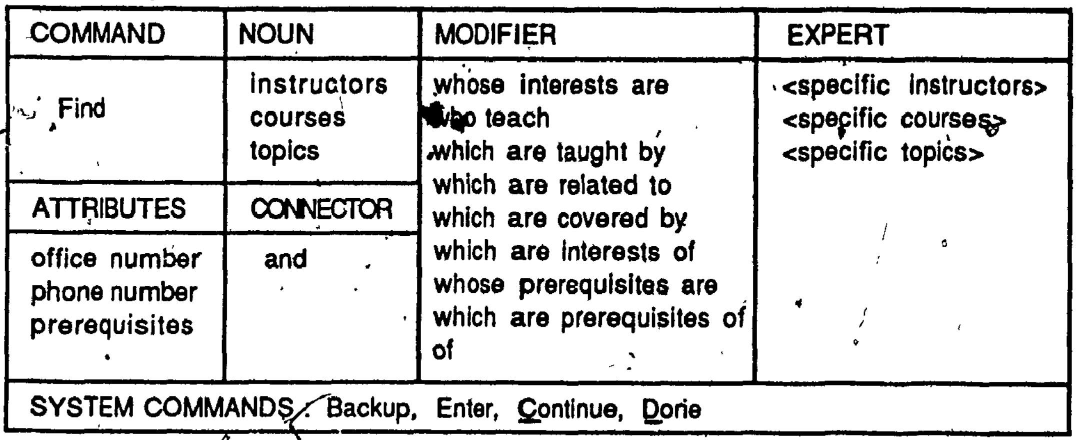


COMMAND	NOUN	MODIFIER
Find	instructors courses topics	whose in who teac which ar which ar which ar which ar whose p which ar of
ATTRIBUTES	CONNECTOR	
office number phone number prerequisites	and	

01	Alagar
02	Atwood
03	Boom
04	Bui
05	Cheng
06	Fancott
07	Ford
08	Goyal
09	Grogono
10	Jaworske
11	Kasvand

SYSTEM COMMANDS.: Backup, Enter, C

Find courses which are taught by



COMMAND	NOUN	MODIFIER	EXPERT
Find	instructors courses topics	whose interests are who teach which are taught by which are related to which are covered by which are interests of whose prerequisites are which are prerequisites of of	<specific instructors> <specific courses> <specific topics>
ATTRIBUTES	CONNECTOR		
office number phone number prerequisites	and		

SYSTEM COMMANDS: Backup, Enter, Continue, Done

Find courses which are taught by Atwood

Courses taught by Atwood: Systems Software Design

Appendix 3

Results of System Prototype Testing

The following table presents the average confidence levels of queries made using three versions of the system prototype.

Version 1 presented the user with a blank screen, Version 2 presented the inital NLMenu screen, without highlighting, and Version 3 presented the screen with highlighting.

The percentages given are averages of the confidence levels recorded for between two and four queries for each version of the sustem prototype.

Subject	1	2	3	4	5	6	7	8	9	10
Version 1	60%	51%	60%	59%	69%	56%	45%	60%	40%	57%
Version 2	63%	64%	62%	70%	82%	70%	66%	66%	66%	69%
Version 3	95%	80%	80%	86%	89%	81%	84%	86%	80%	89%

Overall Averages for all Subjects

Version 1:	55.7 %
Version 2:	67.8 %
Version 3:	85.0 %