A Survey of Interfaces to Data Base and Expert Systems.

Galina Datskovsky Moerdler
Technical Report #CUCS-305-87

Table of Contents

1. Introduction 1
2. Natural Language Interfaces to Data Base Systems 4
    2.1 Systems That Use an ATN Syntactic Analyzer 4
        2.1.1 Lunar 4
        2.1.2 Intellect 7
        2.1.3 Co-op 7
        2.1.4 Summary 9
    2.2 Semantic Grammars 10
        2.2.1 Lifer-Ladder 10
        2.2.2 Planes 13
        2.2.3 Jets 14
        2.2.4 Rendezvous 14
        2.2.5 Summary 15
    2.3 More Recent Natural Language Interfaces 16
        2.3.1 Nlmenu 16
        2.3.2 Team 17
        2.3.3 Summary 19
    2.4 Conclusions -- Data Base Domain 19
3. Interfaces to Expert Systems 22
    3.1 Systems with a Menu-Like Interfaces 22
        3.1.1 Mycin 23
        3.1.2 Guidon 28
        3.1.3 Oncocin 32
        3.1.4 Summary 33
    3.2 Expert Systems with Natural Language Interfaces 34
        3.2.1 Prospector 34
        3.2.2 Xcalibur 37
        3.2.3 Summary 40
    3.3 Conclusions -- Expert Systems Domain 40
4. Conclusions 42

List of Figures

Figure 2-1: Diagram of a typical system with an ATN parser 5
Figure 2-2: Structure built for the noun phrase "The big white ball with the red stripes". 6
Figure 2-3: A typical loaded question. 8
Figure 2-4: MQL representation of the query 'Which students received a grade of F in linguistic courses'. 8
Figure 2-5: Diagram of a typical system with a semantic grammar 11
Figure 2-6: A set of rules and an augmented transition network 12
Figure 2-7: The Acquisition menu. 18
Figure 3-1: A Mycin rule 24
Figure 3-2: A transcript of interaction with Mycin 25
Figure 3-3: Analysis of a typical question. 29
Figure 3-4: A typical interaction with Guidon 31
Figure 3-5: Interaction between Prospector and a user 35
Figure 3-6: Overview of Xcalibur and underlying expert systems 37
Figure 3-7: The output for 'What is the price of the 2 largest dual port fixed media disks?' 38

1. Introduction

The problem of natural language recognition has been studied by many researchers since the early 1960's. In particular, researchers are concerned with building interfaces to existing systems, such as data base and expert systems. Such interfaces focus on easing communications between the underlying systems and the user. In this paper we survey various interfaces to both expert and data base systems. Natural language interfaces to data base systems have been quite successful. In contrast, natural language interfaces for expert systems are still in the initial stages of design.[1]

One mode of communication between an expert system and a user is via a menu interface. Such an interface presents many limitations [Datskovsky 84] [Datskovsky 85]. A natural language interface to expert systems that would not only alleviate such limitations, but also be domain independent and easily transportable from one system to another is desirable. By studying natural language interfaces to data base systems as well as existing interfaces to expert systems, we hope to gain insights on requirements for expert system interfaces and existing natural language technology that may help us in building natural language interfaces to expert systems.

There are major differences between data base and expert systems. A data base system is not expected to know or solve a user's problem, but only supply the information that the user requests. The user is solely responsible for a problem solution. Consequently, an interface to a data base system must simply be able to retrieve information requested by the user. On the other hand, an expert system is designed to be a problem solver. It has to 'know' the user's intentions and gather information in order to come up with a solution. Therefore, a natural language interface to an expert system must be able to take initiative in guiding the user toward a solution to a problem. The interface must be able to derive a user's goal (i.e. what the problem that the user is trying to solve is), as well as facts from any given question and add these facts to the data base.

Natural language interfaces to data base systems have been designed to address various issues,

---

[1] In this paper we are primarily concerned in the system / end-user interfaces, as opposed to other interfaces (for example, those used by knowledge engineers). The major emphasis of this paper is on natural language understanding and not natural language generation.

including linguistic and conceptual completeness and domain independence. Linguistic completeness refers to the degree to which the language of a set of users is appropriately covered by the system, whereas conceptual completeness refers to the degree to which the concepts that are expected by the users can be found in the system's conceptual coverage. For example, some systems allow the user to ask questions about the structure of the underlying data base, while others do not.

Domain independence is a desirable property, because an interface that does not depend on specific domain knowledge can be moved from one underlying system to another with only minor modifications. Time must be spent in order to adapt a natural language interface to each new data base. In some cases only a new lexicon has to be constructed for the interface to function in the new domain. In other cases, new sets of semantic rules have to be added.

Overall, the issue of domain independence has been successfully addressed in natural language interfaces to data base systems. Data base systems tend to be fairly uniform in structure. A major part of the description of a data base system comes from a schema containing information about every field and table in the data base. Most natural language interfaces utilize this description when constructing a lexicon and semantic rules. In this paper we look at a number of natural language interfaces to data base systems and analyze their performance in terms of domain independence as well as conceptual and linguistic coverage.

We evaluate natural language interfaces to data base systems based on their success in achieving conceptual and linguistic completeness, as well as domain independence. We also examine how semantics is achieved in these systems in order to see whether any of the semantic techniques can be applied in the expert systems domain.

In contrast to data base systems, expert systems are not uniform in structure. The concepts addressed by an expert system are less structured and there is no schema describing "fields" in the systems, such as in a data base system. The communication between an expert system and a user is of a different form as well. In a data base system the user asks the question and the system simply supplies the answers, while an expert system usually does a large percentage of the asking.

We study existing interfaces to expert systems in order to identify their inadequacies and to determine the requirements for a natural language interface to expert systems. We also try to determine the major differences in requirements for a natural language interface to data base system and a natural language interface to an expert system.

The rest of this paper is structured as follows. In section two we look at the work done in the data base domain, in section three we look at the expert systems domain, and finally we draw our conclusions.

2. Natural Language Interfaces to Data Base Systems

In this section we examine natural language interfaces to data base systems. Early natural language interfaces were generally unsophisticated and had simple syntactic and semantic components. [Green et. al. 63] [Simmons 65] [Simmons 66]. Later systems used more sophisticated techniques. The systems we survey here include systems with an Augmented Transition Network [Woods 73a] [Woods 70] syntactic parser, e.g. Lunar [Woods, Kaplan and Nash-Webber 72]; systems built with semantic grammars (e.g. Lifer [Hendrix et. al. 78]); and finally some recent systems, e.g. Team [Martin, Appelt and Pereira 83]. These systems provide a good overview of the various techniques available for building natural language interfaces to data base systems.

2.1 Systems That Use an ATN Syntactic Analyzer

In this section we look at three systems, Lunar [Woods, Kaplan and Nash-Webber 72] [Woods 73b], Intellect [Harris 77] and Co-op [Kaplan 79]. Each of these systems uses an Augmented Transition Network (ATN) syntactic parser [Woods 73a] [Woods 70], which is a domain independent component[2]. For overall system structure see figure 2-1.

Such a system first analyzes a query for its syntactic constituents using an ATN parser. The ATN parser returns a syntactic analysis of a sentence in the form of a tree (see figure 2-2 for a typical structure built for a noun phrase). Next, the output of the syntactic parser is analyzed by the semantic component which then translates it into a data base query in the formal data base query language, and finally the query is executed and the appropriate information is retrieved from the data base.

2.1.1 Lunar

Lunar [Woods, Kaplan and Nash-Webber 72] [Woods 73b] is designed as an interface to a simple data base that describes moon rocks. This data base consists of seven mutually exclusive data domains and allows for a fairly small vocabulary. As already mentioned, Lunar uses an ATN parser in the syntactic analysis of the queries.

---

[2] A syntactic parser such as the ATN captures the general syntactic structure of English. An ATN is a graph representation of a grammar with actions on its arcs that allow one to test or set various registers. Some of the notable features of the ATN are recursion and backtracking.

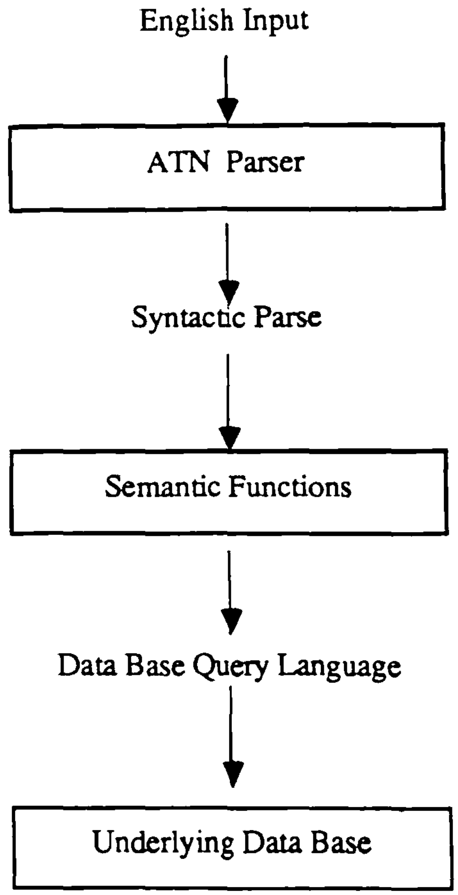

Figure 2-1: Diagram of a typical system with an ATN parser

Lunar's semantic analyzer is similar to that used in Woods's Airline Guide [Woods 67] [Woods 68]. The purpose of this semantic technique is to relate words and syntactic structures to concepts that they reference in the data base. A query is built by using procedures that represent the conceptual primitives of the data base. These semantic procedures gather information from verbs, nouns, noun modifiers and determiners. Consider the following example from Wood's Airline Guide:

```
Query: Is AA-57 an American Airlines flight?
```

The head noun *flight* indicates that the phrase refers to the data base domain FLIGHT. The modifying compound noun *American Airlines* restricts the class of data elements under consideration from all FLIGHTS to only those that have an owner relation to the data element AMERICAN. The determiner *an* indicates the corresponding quantifier SOME. Finally, the sentence is a yes/no question, so

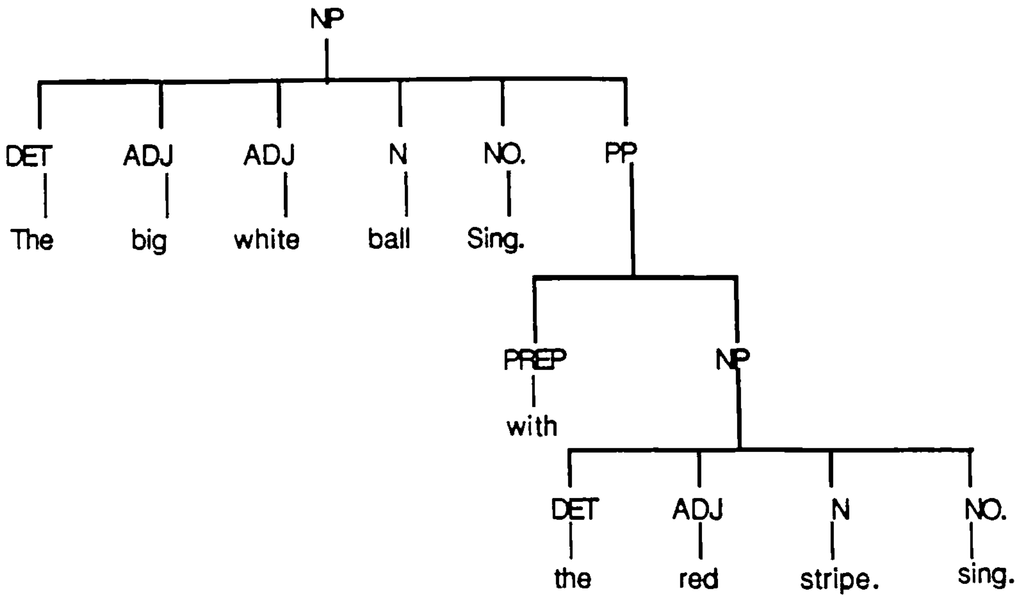

Figure 2-2: Structure built for the noun phrase "The big white ball with the red stripes".

the TEST primitive is used. The final query is:

```
(Test ((For some x/
        Flight : Equal (Owner(x), American)
        Equal (AA-57, x))))
```

Lunar is able to distinguish between two kinds of anaphoric references; semi-anaphora and complete anaphora. Semi-anaphora is a reference by a noun phrase to a part of the previous noun phrase, while a complete anaphora has no modifiers on the noun substitute. Consider the following set of queries;

"Give me all analyses of sample 10046 for hydrogen."

"Give me those for oxygen".

The second sentence contains a semi-anaphoric reference to the noun phrase 'all analyses of sample 10046 for hydrogen', where oxygen should be substituted for hydrogen. To resolve the semi-anaphoric references, a list of all the noun phrases is kept throughout the discourse and the noun phrase containing the noun substitute is compared to all the previously stored phrases, in order to find the appropriate one.

For complete anaphora, if a noun is present the stored structures are searched on the basis of the noun. If no noun is present, the semantic interpretations of the rest of the sentence is examined for possible semantic types that apply to the pronoun.

Lunar is one of few systems to undergo extensive testing in a "real world" setting. It was demonstrated during the Second Annual Lunar Science Conference in Houston, Texas. During this demonstration, the system successfully answered 80% of the questions asked by geologists that fell within the scope of the data base and another 10% would have been successful but for trivial dictionary coding errors. It did not do quite as well on questions asked by non-geologists, because their questions tended to drift out of the range of the data base.

2.1.2 Intellect

Another issue that concerns natural language interface designers is transportability. Intellect [Harris 77] is one of the first systems to achieve this goal. The system is based on the Robot system [Harris 77] and is commercially available from the AIC corporation. A user's query is first processed by an ATN based syntactic parser followed by semantic analysis that results in a formal language representation of the query. Intellect is designed to deal with large vocabularies necessary for most question answering domains. Each entry in the vocabulary file indicates the data domain in which this entry occurs. In order to handle this large vocabulary, Intellect builds an inverted file of data element names. Since it is easy to build an inverted file automatically, this feature makes the system very portable. It takes an average of one week to adapt Intellect to a new domain. Note that Intellect is not concerned with conceptual completeness. Each user query is translated directly into a data base query.

2.1.3 Co-op

The Co-op [Kaplan 79] system deals with the problem of responding to loaded questions. A loaded question is one where the questioner assumes something to be true of the domain that is actually false. A natural language system that can not detect such a question may provide meaningless responses to certain queries. For example, consider the query in figure 2-3. Although 1a is a correct direct response, it is not very cooperative. Question 1 presumes that cs4521y was taught last year. To be cooperative rather than misleading, the system should correct the user's misimpression with a corrective indirect response to question 1 such as 'Last year cs4521y was not taught"

1. Question: Which students received a grade of F in CS4521y last year?

1a. Answer: None

2. Question: Was cs4521y taught last year?

2a. Answer: No

Figure 2-3: A typical loaded question.

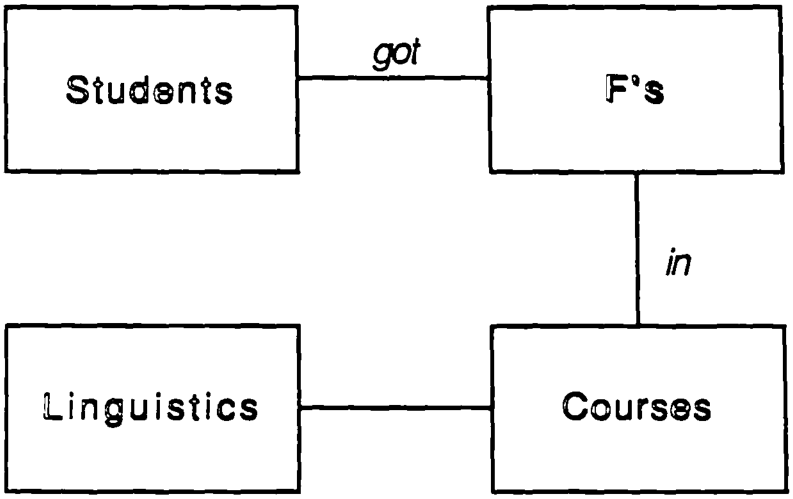

Figure 2-4: MQL representation of the query 'Which students received a grade of F in linguistic courses'. Adapted from [Kaplan 78]

Co-op provides corrective indirect responses to loaded questions. As all the systems described in this section, Co-op uses an ATN parser for the syntactic processing of a query. The query is then translated into a meta query language - MQL. In MQL, each query is viewed as a graph. A representation of 'Which students received F's in linguistics courses ' is shown in figure 2-4. Each of the nouns in the query generates a set, whereas verbs and prepositions generate links as shown in the figure above.

Each of the subgraphs of this representation must be non empty in order for a direct response to the query to be meaningful. If the initial query returns a null, the system's control structure passes each of the subgraphs to the interpretive component to check for non-emptiness.

There are a few key points about the Co-op system. The computation of the presumptions is totally independent of the domain specific knowledge. The mechanism for selecting an indirect response is identical to the procedure for executing a query. If the initial query is successful, no penalty is paid for the ability to produce an indirect response.

All the domain specific knowledge of Co-op is stored in the lexicon. Each noun in the lexicon has a SETDEF associated with it. A SETDEF identifies all the fields of the data base a given word can refer to. Translating the MQL to formal query language is done by traversing the MQL structure and recursively constructing the formal query from the information in the SETDEFs of the lexical labels of the MQL nodes. First, a stream for the root node in the MQL is created. Next, the MQL is traversed in preorder, recursively and substreams for the nodes traversed as domains of the streams of their dominating nodes are recursively embedded. In order to transport Co-op to a new domain, one has to build up a new lexicon. The MQL structure itself is totally domain independent. This makes Co-op a domain independent system with an important ability to generate indirect responses.

2.1.4 Summary

In this section we looked at three major systems; Lunar, Intellect and Co-op. Lunar is a domain dependent system that is able to handle single user questions and interpret elliptic and anaphoric references. The system's semantic rules, although well suited for the domain, can not be transferred to another domain. A new set of these rules must be written for each new system. Lunar does not have a very large vocabulary, nor does it do well with questions not asked by geologists. Both Intellect and Co-op are transportable to other domains. However, each achieves domain independence in a different way. Intellect is designed to deal with large vocabularies and achieves domain independence by building an inverted file of data element names. Co-op uses a domain independent graph structure, and stores all the domain dependent information in its lexicon. In order to transport the system, time is needed to encode this domain dependent information.[3] Intellect tries to address the issue of linguistic coverage. It works with large vocabularies which are necessary for most question answering domains.

2.2 Semantic Grammars

In the previous section we looked at systems that used syntactic parsers as the first stage in the analysis of a user's query. In this section we look at systems that do not concern themselves with syntax, but do only semantic analysis. The systems we discuss here primarily use semantic grammars. For a general overview of a system structure see figure 2-5). A semantic grammar is a context free grammar in which the choice of non terminals and production rules is governed by semantic function. There are certain advantages to using a semantic grammar: after a sentence is parsed with a semantic grammar the result can be used immediately; some ambiguities that arise after a strictly syntactic parse can be avoided; and finally, syntax issues that do not affect the semantics can be completely ignored. The main disadvantage is that the number of rules may quickly become too large because syntactic generalizations are missing and this may cause the parser to become more expensive and very slow.

In this section we consider four systems, Lifer-Ladder [Hendrix et. al. 78], Planes [Waltz 75] [Waltz 77], Jets [Finin, Goodman and Tennant 79] and Rendezvous [Codd 74] [Codd 78] that use semantic grammars as a primary mechanism for analyzing a user's query.

2.2.1 Lifer-Ladder

The Lifer [Hendrix et. al. 78] system is designed to facilitate the construction of natural language interfaces to data base systems. Lifer is a semantic grammar based parser that attempts to match a query to a sentence template. Lifer has two major components: a set of interactive functions for specifying the language and a semantic parser. The user specifies an application language appropriate for his domain using the interactive functions. The system then uses this language specification to process natural language commands. Naturally, a new dictionary must be specified for every new domain.

---

[3] Unlike Lunar, Co-op employs a domain independent algorithm for processing its queries.

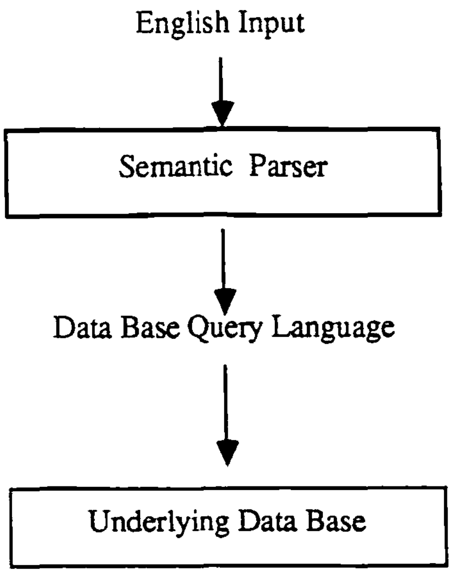

Figure 2-5: Diagram of a typical system with a semantic grammar

The language is specified in terms of rewrite rules that are automatically translated into an augmented transition network (See figure 2-6 for an example). In order to specify a network, Lifer provides several interactive functions. Two such functions are MS (meta symbol) and PD (pattern definition). Lifer also allows for personalizing the application language using the SYNONYM function. Using this function a user may introduce structures into the application language that allow users to define their own synonyms at run time. For example,

```
PD[<LTG>
   (Define <New-World> like <Old-World>)
   (SYNONYM <New-World> <Old-World>)]
```

allows the parser to accept inputs such as *Define JFK like Kennedy.*

To specify a network like the one depicted in figure 2-6 the user may apply the functions MS and PD. The first pattern in the figure may be specified as follows:

```
PD[<LTG>
   (What is the <Attribute> of <Person>)
   (F1)]
```

The metasymbols <Person> and <Attribute> are defined with the MS function as follows:

```
MS[<Attribute>
(Age Sex Hight Weight)]
MS[<Person>
(Mary John Dan Ron)]
```

Lifer starts at the top of the net and tries to match the sentence to a template. If an input matches a template, the expression on the right (the unspecified functions F1,F2,F3 in figure 2-6) is evaluated. Note that unlike the systems described in the previous sections, Lifer does not build an intermediate syntactic representation of a user's query.

```
<ltg> --> What is the <Attribute> of <Person> | F1
<ltg> --> What is <Person> <Attribute> | F2
<ltg> --> How <Attribute> <Person>     | F3


<ltg>---------What is -- the <ATTR> of <Person> | F1
    |                |
    |                -- <Person> <ATTR> | F2
    |
    ----------How <ATTR> is <Person> | F3
```

Figure 2-6: A set of rules and an augmented transition network

The set of rules in figure 2-6 allows a user to ask questions like *How old is John? What is Mary's hair color? What is the age of Margie?* etc.

One of the applications of Lifer is the Ladder system [Hendrix, Sacerdoti, Sagalowica and Slocum 1978; Sacerdoti 1977] which is designed to answer questions about a naval data base. The Lifer system was used in order to build the rewrite rules of Ladder. A typical template used by the Ladder system is:

What is the <attribute> of the <ship>? == ((Nam eq <ship>(?<attrib>))). In this template, the part to the left of the == corresponds to the template in figure 2-6, and the part to the right of the == corresponds to the expression to be evaluated.

One of the problems of this formalism is extensibility. In order to allow the user to ask about several attributes of a ship in one sentence, the grammar has to be extended as follows:

1. <ATTRIBUTE> --> <ATTRIB> <ATTRIBUTE>

2. <ATTRIBUTE> --> <ATTRIB> AND <ATTRIBUTE>

3. <ATTRIBUTE>--><ATTRIB>

The addition of these rules allows the system to process queries of the form *'What is the length and the home port of the Constellation?'*.

In order to allow the user to combine other non terminals a set of such rules must be added for each such non terminal. A great number of such rules has to be rewritten for each new domain, thus making transport fairly difficult. Lifer is able to deal with some elliptical and pronominal references. If all the sentence templates fail, it examines the parse tree of the previous sentence. If a match is made with a fragment of the parse tree, the ellipsis is inserted into the tree and a new query is thus generated. Lifer is a syntactically weak, but semantically adequate system which requires several weeks and a skillful applications designer to bring it up in a new domain [Tennant 81].

2.2.2 Planes

Planes [Waltz 75] [Waltz 77] is constructed as a natural language interface to a large Navy data base that contains the flight and maintenance records for all naval aircraft. Whereas the main goal of the Lifer system is transportability, the main goal of Planes is linguistic completeness, i.e. to be tolerant to all user inputs, even those not stated in standard English. The analysis of a user's query proceeds in three steps: the user's query is analyzed for semantic constituents; a formal query is constructed; and, if approved by the user, that query is executed. The analysis of a sentence is carried out in a top down fashion. During the semantic scan, a one-state ATN is used at the top level to repeatedly call each subnet. Planes is able to handle pragmatic and substructural ellipses and is tolerant to user inputs that are not stated in standard English. However, because Planes ignores detailed syntax, it has problems with word sense selection and modifier attachment. Domain independence is not one of the goals of its design and is not addressed by the system.

2.2.3 Jets

Planes was developed to answer the problem of linguistic completeness, however, testing revealed that conceptual completeness, i.e. the set of concepts a system can handle, is important in a question answering system as well. For example, the user should be allowed to ask questions about the data base, as well as refer to concepts he previously mentioned. Jets [Finin, Goodman and Tennant 79][4]centers on establishing adequate conceptual coverage of the data base and its contents as well as the events described by the data base. It is intended to enlarge the domain of discourse, provide pragmatic knowledge, and to break away from the tradition of interpreting each user utterance as a data base query. Jets allows the user to refer to certain concepts that cannot be referred to in Planes, such as the structure of the data base or certain definitions.

The system achieves the goal of greater conceptual completeness by being able to provide knowledge about the concepts of the data base. The system's knowledge is organized into a hierarchical network of frames, where the more specific frames inherit properties from the more general ones. The main advantage of this abstraction hierarchy is that the classification of specific concepts into more general ones promotes the identification of regularities among these concepts. Jets has a set of rules which make use of these frames. These rules are likewise stored in frames and have three basic slots; pattern, action and condition. The rules are likewise organized hierarchical by generality. The system also has problem solving frames which help it develop a plan to extract the required information from the data base. Like Planes, Jets does not address the issue of domain independence.

2.2.4 Rendezvous

The Rendezvous [Codd 74] [Codd 78] system is designed specifically for interaction between a data base system and a casual user, i.e., a user who is not very familiar with the contents of the data base. Like Lifer, Rendezvous is based on a semantic grammar parser. The system assumes that queries issued by the user may be incomplete, misunderstood by the system, or based on false assumptions about the data base. During its dialog with the user the system tries to clarify the user queries by resorting to a menu. The system also resorts to a menu if it is unable to "understand" the query. After the query is

---

[4] Jets uses the same data base as Planes.

finally formulated, the system paraphrases it to make sure it is what the user intended, and only after that is the query executed. One of the major goals of the Rendezvous system is to accept any relevant query that the user may issue. This is achieved by analyzing the user's queries with minimal regard to either the syntactic or semantic content. This aspect of Rendezvous is similar to Planes. The Rendezvous system consists of three major components, the Analyzer, the Synthesizer, and Dialog Control component. The principal function of the Analyzer is to translate as much as possible of the user's query into a data base sublanguage, ALPHA. The Analyzer itself contains three components, the Word Transformer, the Phrase transformer and the Diagnostic component. The Word Transformer reduces each word to a normalized form by means of suffix transformations and by replacing words that have a single canonical synonym by that synonym. The phrase transformer changes certain phrases in the text closer to a form required by ALPHA. Some of the phrase transformation rules require that phrase components be semantically compatible and because of this strong semantic orientation the left hand sides of these rules are referred to as semantic templates. These rules are similar to the rules of the Lifer system. The Diagnostic component is invoked if part of the user's query is not properly understood by the system. If this happens, a dialog with the user is initiated in order to clarify the misunderstanding. Finally, the synthesizer is responsible for translating the ALPHA representation of the query back to English and displaying this paraphrase to the user. Thus, Rendezvous provides a carefully controlled natural language interface that seems to be particularly appropriate for casual users.

2.2.5 Summary

In this section we have examined systems based on semantic grammars. The Lifer system is designed with the issue of transportability in mind; however it is not extremely successful in achieving this goal because the person trying to adapt the system to a new domain has to come up with rules that cover every possible combination of non terminals for each concept of the domain. Rewriting the rules for each new domain may prove to be a difficult task. Nevertheless, Lifer is one of the first systems to achieve any degree of domain independence, primarily by allowing the user to build the interfaces himself.

Planes is concerned with the problem of linguistic completeness, i.e. it is tolerant to all user input, even if syntactically incorrect. During the testing stage of Planes it was determined that conceptual completeness is also important to the users [Finin, Goodman and Tennant 79]. Jets is a system designed as a result of these tests and it addresses the problem of conceptual completeness by encoding knowledge about its data base. Neither system addressed the issue of domain independence.

The Rendezvous system is also not concerned with domain independence issues. Its primary concern is with providing a tight interface for use by persons unfamiliar with the contents of the data base. It resorts to menus when necessary in order to clarify a user's input. The system is geared for use with large data bases.

2.3 More Recent Natural Language Interfaces

In this section we consider two more recent natural language systems, namely Nlmenu [Tennant et. al. 83] and Team [Martin, Appelt and Pereira 83] [Grosz et. al. 85]. Both of the systems are domain independent, but they use fundamentally different approaches to achieve this goal. Nlmenu uses a different strategy than any discussed so far. Team, on the other hand builds on the previous work to make a better and more robust system.

2.3.1 Nlmenu

A system offering a totally different approach to natural language interfacing is the Nlmenu system [Tennant 81]. This menu based natural language system works as follows. At every point the user has a limited choice of words to select from in order to form some part of a given query. For example, in the first menu the choice may be between the words FIND, DELETE, INSERT. Suppose the user chooses the FIND command. Next, the user gets to select from another menu, for instance a noun menu. Here he may have to choose from words like PART, SUPPLIER, etc.. This selection process assures that there is only a finite number of queries that a user may generate and that the system can handle every one of those queries. This interface is transportable from domain to domain because it can automatically generate the menus by using the underlying description of the data base.

Nlmenu is not a true natural language interface. The number of queries that a user can formulate is limited and the construction of such queries is carefully controlled by the system. This approach is an interesting alternative to a natural language interface and therefore we feel that it is important to include it in this survey. A menu-based approach seems to work for small data bases, but may not function very well in a large domain because the menus become too large.

2.3.2 Team

The system which perhaps is most successful in achieving transportability is Team [Martin, Appelt and Pereira 83] [Grosz et. al. 85]. Team works in two stages: first a data base expert answers questions about a data base and the linguistic expressions used to refer to concepts in that data base, and then the end-user may use the system as a question answering facility. The Team system has three major components: Acquisition, Dialogic and Database Access. Acquisition obtains a description of a data base from a data base expert, Dialogic translates English input into a logical from that represent its literal meaning, and finally, the Access component converts this logical form to a formal database query.

The system acquires new information, such as lexical, conceptual and data base schemas during the acquisition phase. The Acquisition system consists of a menu depicted in figure 2-7. The figure shows a menu of general commands, menus associated with relations, fields and lexical items. The acquisition is centered around the relations and fields of a data base, since this is the information most familiar to the data base expert. The data base expert is assumed to have only some general knowledge about English grammar. He is expected to know what proper nouns, verbs, plurals and tense are, but nothing more detailed than that.

In the case of verb acquisitions, however, more sophisticated linguistic knowledge is required. When acquiring new verbs, Team proceeds by asking questions about sample sentences, and extracting the information it needs form the data base expert's response. For instance, consider the acquisition of the verb *to cover*, with an underlying data base containing names of countries, their contents, capitals, area and population. First, Team asks three questions to determine the verb's third person singular, past, and past-participle forms. In this case the system defaults *covers, covered and covered* are the correct answers . Next, Team asks the data base expert to use the verb in a declarative sentence with indefinite noun phrases in the most general manner. Such a sentence might be *A country covers an area.* This response tells Team that *cover* is a transitive verb, that a new predicate *Cover* should be added to the conceptual schema with two arguments, *country and area-measure*, where the subject of the active form

3. Interfaces to Expert Systems

The area of expert systems investigates techniques for building systems with specialized problem solving expertise. Some of the fields in which expert systems have been written include medicine, organic chemistry, tutoring, mathematics, geology, and many others (For more information see [Barr and Feigenbaum 82]).

In order for a human expert to be able to answer a person's question, he often has to carry out extensive dialogs with that person in order to gather information about that person's needs. Extensive interaction and clarification of user needs are also required in most expert systems in order for the system to do problem solving. One way expert systems communicate with their users is via a menu interface. To gather information the system poses a question and the user is given a number of possible answers and he then proceeds to choose the one that he thinks best corresponds with the correct answer (For more information see [Datskovsky 84]). Natural language interfaces to expert systems are also being investigated [Carbonell et.al. 83] [Datskovsky and Ensor 86]. There are systems that do not communicate with their users, such as Ace [Stolfo and Vesonder 82] and Dendral [Lederberger 82]. These systems either gather on-site data or start from an initial model.

In this section of the paper we examine several expert systems and their interfaces. We survey the interfaces between the expert systems and the end-users, as opposed to interfaces used by knowledge engineers to build the knowledge base of an expert system. We look at systems with menu interfaces as well as systems that have some form of natural language interface, in order to better understand what the requirements for an expert system - user interface are, and how to replace the existing interfaces with an efficient natural language interface that would meet these requirements.

3.1 Systems with a Menu-Like Interfaces

Perhaps the most common means of communication between an expert system and the user is via a menu or a menu based interface. (e.g. Digitalis Therapy Advisor , Puff, Internist, PIP). For more information about these systems see [Barr and Feigenbaum 82]. Typically a system poses a question and presents a number of possible answers to the user, who is expected to select one of those answers. In this section we take a close look at two such systems, Mycin [Shortliffe 76] [Scott et al 84] and Guidon sentence is mapped onto the first argument, and the object is mapped onto the second argument.

The Dialogic parser consists of several stages as well. All the domain dependent information the system needs can be found in the conceptual schema and the lexicon. The syntactic part of Dialogic consists of the Diamond parser and the Diagram grammar. The grammar consists of a set of context free rules with constructors that enforce context sensitive syntactic constraints. These constructors are used to assign scores to different parses. Each rule of the grammar has a translator function associated with it. These functions compute the predicate relations of the sentence. The semantic component of the system thus translates the English input into a logic form which is then transformed by the schema translator into the underlying data base query language.

Team seems to be successful in achieving domain independence. The domain independent portion of Team includes the parser, grammar, semantic translators, pragmatic and scope determining processes, schema translator and the basic vocabulary. The rest of the required information is automatically acquired by the system.

Figure 2-7: The Acquisition menu.
Adapted from [Grosz et. al. 85]

2.3.3 Summary

In this section we examined two recent, but completely different systems that use fundamentally different approaches to natural language processing. The Nlmenu system resorts to a combined natural language - menu approach. At each stage of the query formation process the user has a finite number of inputs to choose from. This approach, although rather restrictive of the range of user input, guarantees that the system is able to handle any input entered by the user. NL menu is transportable from domain to domain. However, it is not well suited for use with large data bases, because the menu for each stage becomes too large.

The second system considered is the Team system. Team is very successful in achieving domain independence. Team makes a clear division between domain dependent and domain independent components and provide means for automatic acquisition of the domain dependent components. Team is quickly transportable from domain to domain, as has been demonstrated by some informal testing [Grosz , et. al. 85].

2.4 Conclusions -- Data Base Domain

In this part of the paper we have looked at natural language interfaces to data base systems. Starting with the early work in that area, we next looked at systems that used an ATN syntactic component, systems that used semantic grammars and finally some of the more recent systems.

We have described these systems based on their transportability as well as their linguistic and conceptual coverage. It seems that systems that use syntactic generalizations perform better in large domains than systems that do not. For small domains, however, systems employing semantic grammars often manage to achieve a greater degree of linguistic and conceptual coverage.

All natural language interfaces to data base systems rely on the underlying data base schema describing the fields and tables of the system. For example, a natural language interface may, during the acquisition phase, ask the data base expert to describe what each field name of a given table refers to [Team]. The means of doing semantic evaluation is different in every system. We will review and evaluate here the different semantic techniques we discussed.

One approach we looked at in this section is the menu based approach to natural language [Nlmenu]. In a window based system, lexical analysis, parsing and semantic processing is fairly easy and the system runs efficiently because the options available to the user are rigorously constrained. However, this method does not seem to work well in semantically rich domains [Rich 84] because the number of options becomes too large and it is difficult to encode in the grammar all the information that is required to determine whether a particular statement can be executed correctly.

A different and more flexible approach to natural language understanding is the semantic grammar approach. The rules of a semantic grammar are designed to produce a parser output that closely corresponds to the input of the desired target program. The rules are also designed so that the set of statements that can be parsed corresponds as closely as possible to the set of actions in the target program. A query in a system employing semantic grammars can be generated by an action attached to the rule itself and executed when that rule is applied [Lifer-Ladder, Planes, Jets]. The interfaces that employ semantic grammars are usually domain dependent [Planes, Jets], however for a domain independent system not only a new vocabulary must be acquired, but also a new set of pattern action rules must be constructed. The Lifer system facilitates this acquisition with a set of on-line interactive functions. Semantic grammars seem to work well in small domains. Because this method does not capture the syntactic regularity of the language, it does not seem to be applicable in domains requiring large subsets of the language [Rich 84].

The third class of systems we reviewed in this paper employs a separate syntactic parser and semantic analyzer. We looked at several approaches to semantic analysis in systems of this kind. One such approach encodes semantics as a set of pattern action rules [Lunar]. The purpose of this semantic technique is to relate words and syntactic structures in which they appear to the concepts and relationships that they reference in the data base. This method works well in most domains; however, a new set of rules has to be written and a new set of associations made for every new data base.

Some semantic techniques are aimed primarily at achieving domain independence. This has been successfully accomplished in natural language interfaces to data base systems. In such interfaces, all the domain dependent information is generally stored in the lexicon [Co-op, Team]. The dictionary is either built by hand [Co-op], or interactively acquired [Team] with the help of a data base expert. Generally, the domain independent parts of a natural language interface include a parser, a basic vocabulary and semantic translators. Semantic routines build some sort of an intermediate, domain independent representation which is then translated into a formal data base query.

In the next part of this paper we look at some of the existing interfaces to expert systems, and consider whether any of the techniques applied in building natural language interfaces to data base systems can be applied in the expert systems domain.

[Clancey 84] [Clancey 79]. We also look at Oncocin [Shortliffe et. al. 81] [Shortliffe et. al. 83] [Hickam et. al. 85], which gathers information as a physician fills out an on-line form during a session with a patient. Mycin uses a menu interface as its primary means of communication, but it also has a limited natural language question answering mode available. Guidon uses a mixed initiative interface in communication with the user.

3.1.1 Mycin

The Mycin system is designed to provide advice on diagnosis and therapy for infectious blood diseases. Mycin's knowledge base is encoded in production rules.[5] The rules are highly stylized which makes it easy for the system to examine them in order to answer questions. English translations of the rules are also available. This translation is used for explanation generation. Each rule represents a single chunk of domain-specific knowledge. If the premise (the left hand side) proves to be correct, then the conclusion (the right hand side) is executed. For an example of a typical rule see figure 3-1. Menu questions are associated with the rules. When the system lacks required information on the left hand side of a rule, and there is a menu question associated with the rule, the system asks the user to provide the required information. A frame-like representation, called the schema, drives the menu process.

Mycin's rules are evaluated by a backward chaining mechanism. This results in setting and exhausting each subject as it is encountered and grouping together all questions about a given topic. This is so because a backward chaining mechanism first collects all the required data to prove a given goal, before going to the next goal (for a detailed discussion of the backward chaining mechanism see [Van Melle 81]). Here, backward chaining proves to be better in this case than forward chaining, because the questions posed by the system seem more focused and connected from the point of view of the physician.

Mycin essentially works in two phases. It first gathers all the necessary data about the patient and

---

[5] The knowledge base of an expert system generally consists of a collection of rules of the form *IF <premise 1> ... <premise n> THEN <conclusion>*. The predicates in the left hand side of the rule may be connected by either conjunctions or disjunctions.

Each rule stores the knowledge that if the premise (left-hand-side) is true then the action (right-hand-side) should be performed. The selection and evaluation of the rules is controlled by the interpreter. As these rules are evaluated, they alter the values of the predicates in the knowledge base and generate input and output. Common interpreters for rule-based systems are based on sequential statement evaluation (e.g. [Bobrow and Stefik 83]), forward chaining (e.g. [Forgy 81]) or backward chaining (e.g. [Van Melle 81]).

```
PREMISE:  (And (Same cntxt infect primary-bacterimia)
               (Membf cntxt site sterilesites)
               (Sam cntxt portal gi)
ACTION:   (Conclude cntxt ident bacteroides tally .7)


IF   the infection is primary-bacterimia               AND
     the site of the culture is one of the sterile sites  AND
     the suspected portal of entry of the organism is the
     gastrointestinal tract
THEN there is suggestive evidence (.7) that the identity of the
     organism is bacteroides.
```

Figure 3-1: A Mycin rule

solicits a proposed diagnosis by asking numerous questions, and then proceeds to verify the diagnoses and give a therapy recommendation. Mycin employs a menu interface for interaction with the user. The menu questions are generated when the system has to find out a certain premise in order to evaluate a rule. The system poses a question and expects to receive an answer from an implicit menu. The user has only a limited choice of answers to every question, even though all of the choices are not displayed to him. Often the choice is between a *yes* or a *no*, as in question 5 in figure 3-2. In the figure, the first four questions have to do with the patient's personal data: sex, age, etc. The next questions check all the cultures taken, organisms discovered in those cultures, as well as other relevant data. When all the necessary information is gathered, the system makes a therapy recommendation. The dialog between the system and the physician is rather long and tedious. If at any point the physician were to choose the wrong answer to a menu question and realize it only after several other questions were asked, it would be difficult for him to back up to the point where the wrong choice was made.

In addition to the menu interaction, the system has an extended question answering mode and an explanation facility. During a consultation, Mycin may ask questions which the physician finds irrelevant. In such a situation the physician may enter a WHY instead of answering the question. If a WHY question is entered, the system responds with a translation of the rule which generated the question. If the user is not satisfied by the explanation, he may continue to investigate the current reasoning chain

```
    MYCIN #9B - 80 YEAR OLD DIABETIC WHITE MALE , S/P IRRADIATION FOR
LARYNGEAL CARCINOMA IN 1974 , ADMITTED WITH FEVER AND LETHARGY.  HAS
DEVELOPED  LLL INFILTRATE AND PRODUCTIVE COUGH WITH KLEBSIELLA GROWING
ON TTA CULTURE.  NOW 4 OF 4 BLOOD CULTURES ARE POSITIVE FOR GRAM
NEGATIVE ROD.

   --------PATIENT-244--------
   1) Patient's name:
   ** PT244
   2) Age:
   ** 80 YEARS
   3) Sex:
   ** MALE
   4) Race:
   ** CAUCASIAN
   5) Are there any cultures for Pt244 which may be related to the
     present illness, and from which organisms have been grown
     successfully in the  microbiology laboratory?
   ** YES

      --------CULTURE-1--------
      6) From what site was the specimen for CULTURE-1 taken?
      ** SPUTUM
      7) Please give the date and time when the sputum culture was
         obtained. (mo/da/yr time)
      ** 25-Oct-75 11:54AM
      8) What was the method of collection of the sputum culture?
      ** TRANSTRACHEAL-ASPIRATION
      9) What is the infection which you feel would account for any
         organism isolated from the sputum culture?
      ** PNEUMONIA
      The first organism isolated from the sputum culture will be
      referred to  as:
         --------ORGANISM-1--------
         10) Please enter the laboratory-reported identity of
             ORGANISM-1, including species or subtype, if known:
         ** KLEBSIELLA
      11) Any other organisms isolated from the sputum culture?
      ** NO
```

Figure 3-2: A transcript of interaction with Mycin

The system continues with similar questions ...

```
   83) On a scale of 0 to 4, where higher numbers indicate increasing
       severity, how would you rate Pt244's degree of sickness?
   ** 3
   84) Is Pt244 febrile?
   ** YES
   85) Do you know the results of a recent CBC of Pt244?
   ** YES
   86) What was Pt244's peripheral white count (in thousands)?
   ** 6.5
   87) Percent of WBC's which were PMN's:
   ** 24
   88) Percent of the peripheral WBC's which are immature forms:
   ** 0
[Considering organisms (other than those isolated from cultures) which
might be present...]

     [Determining which drugs are desirable for use against the
      Klebsiella- pneumoniae...]
```

Finally, a recomendation of therapy is made...

```
[REC-1] My preferred therapy recommendation is as follows:
   In order to cover for Items <1 2 3 4 5 6>:
        Give the following in combination:
        1) CEFAZOLIN
        Dose: 1.06g (5.2 ml) q6h IV for 10 days [calculated on basis
        of 20 mg/kg]
        [The dose should be modified in SEVERE renal failure.  I can
adjust the dose only if a creatinine clearance or a serum creatinine
is obtained.]

        2) GENTAMICIN
        Dose: 90 mg (2.2 ml, 80mg/2ml ampule) q8h IV for 10 days
        [calculated on basis of 1.7 mg/kg]
        [The dose should be modified in renal failure.  I can adjust
        the dose  only if a creatinine clearance or a serum creatinine
        is obtained.]
```

figure 3-2 continued[6]

---

[6] This transcript was created by the author when running the Mycin system. We would like to thank Dr. Shortliffe for giving us an opportunity to do so.

by repeating the WHY command several times. Each such command is interpreted by Mycin as a request for display of the next highest rule in the current reasoning chain. In addition to the WHY command, the system also has a HOW command. If the user does not understand how a specified condition was met, he may issue the HOW command followed by the identifying member of the premise clause in question.

Mycin also has a more extensive question answering facility which is automatically available to the user at the end of each consultation session. Here a user may ask simple English questions about the system's reasoning, such as why the system wanted to know a certain fact, how a certain fact was relevant, etc. An example of a typical question and its analysis may be found in figure 3-3. The question answering facility is not the primary focus of the research, so the natural language understander is not sophisticated. It uses crude techniques relying strongly on the very specific vocabulary of the domain. The system's dictionary contains approximately 1400 words common to the domain. The dictionary includes all words that are acceptable values for a parameter, common synonyms or abbreviations of these words, as well as words used elsewhere in the system in describing the parameters. There are five major steps to understanding the question, finding the rules and printing the answer:

1. The question is reduced to a list of terminal, or root words[7]
2. Pattern matching classifies the question as a rule-retrieval question and divides it into premise part and an action part.
3. Dictionary properties of the terminal words are used to determine which parameters are relevant to each part of the question.
4. After selecting only the most strongly indicated parameters, the final translation tells which rules can answer the question.
5. The answer consists of finding the rules that meet these restrictions and printing those that the user wants to see.

For the purpose of analyzing the question, a non-terminal word is considered to be equivalent to its terminal synonym. A list of properties and descriptions is associated with each terminal word. This list is useful in determining the meaning of a question that uses a terminal word or one of its synonyms. During this phase of the parsing each word in the original text is replaced by its terminal word. If a word is not found in the dictionary, the system uses the root-extracting algorithm [Winograd 72] to see if the word's lexical root is in the dictionary. The word is then replaced by a terminal word for its root. The resulting

---

[7] Terminal words have associated with them a set of properties that are useful in determining the meaning of a question.

list of terminals is then sent to a function that recognizes phrases and replaces each phrase with a single synonym. Next, the question is classified so that the program can determine which specialist should answer it. Each type of question has an associated template with blanks to be filled in from a given question. Once the blanks in this general template are filled, the system has all the necessary information to answer the question. A number of possible answer templates are associated with each type of question. The specialist that answers the question determines which of the answer templates is most appropriate and fills in the blanks in this template. For example, the two possible templates for *How do you know the value of <param> of <cntxt>?* are

1. I used <rule> to conclude that <param> of <cntxt> is <value>. This gave a cumulative CF of <certainty factor>. The last question asked before conclusion was made was <question number>.
2. In answer to question <question number> you said that <param> of <cntxt> is <value>.

The question answering module is a useful addition to the system, which allows the user to go back and ask questions about the system's reasoning. However, it allows only simple questions, and is rather limited in the scope of its vocabulary. The answer to a question is simply a list of rules, which is not very meaningful to someone unfamiliar with the rule base.

Mycin performs at or near expert level in almost all test cases [Yu et. al. 79], however, there are doubts about the system's clinical utility [Buchanan and Shortliffe 84]. This may be caused by its incomplete knowledge, as well as by its rather limited and tedious interface. Physicians, as most busy professionals, do not like to spend time on questions that are useless from their point of view, nor do they like to deviate from their daily routine. Even so, Mycin is, without doubt, one of the first systems to achieve this level of success in its design.

3.1.2 Guidon

The second system we discuss in this section is Guidon [Clancey 84] [Clancey 79]. Guidon is designed to teach diagnostic problem solving using the knowledge base of the Mycin system.

In a session with Guidon, the student plays the role of a consulting physician. A case of a sick patient is presented to the student who is then expected to ask questions which he thinks are relevant to the case. Before beginning a session with the student, Guidon uses Mycin to evaluate the case that is to be

```
User question:  When do you decide that an organism is a contaminant?
                          Analysis of the Question above:
Terminal words: When do you conclude that an organism is a contaminant

Question type: Rule retrieval

Premise part:  (When do you conclude)

Action part:  (That an organism is a contaminant)

Vocab. clues (premise):   (Wheninfect (any 1) (Whenstop (any) 1)
                          (Whenstart (any) 1) (Duration (any) 1)

Vocab. clues (action):    (Contaminant (any) 4) (form (any) 1)
                          (Samebug (any) 1) (Coverfor (any) 1)

Final translation:  Premise: ANY
                    Action: (Contaminant any)

The rules listed below conclude about whether the organism is a
contaminant
6,31,351,39,41,42,44,347,49,106
Which do you wish to see?

**6

Rule006

IF 1) The culture was taken form a sterile source, and
      20 It is definite that the identity of the organism is one of :
      Staphylococcus-coag-neg bacillus-subtillis
      corynebacterium-non-diphtheriae

THEN: There is strongly suggestive evidence (.8) that the organism is
      a contaminant
```

Figure 3-3: Analysis of a typical question.

presented to the student. All the information that Mycin needs in order to evaluate the case is already available, so Mycin need not ask any questions during the evaluation. After the student is done collecting evidence, he makes his hypothesis, which is then compared with that of Mycin.

In order to evaluate the student's knowledge the system builds a three- component model of the student. The *use-history* component is the cumulative record of which rules the student knows. The *student-applied* component is the system's record of the rules it believes the student has the evidence to apply. Finally, the *used* component records the system's beliefs that the student will cite a rule in support of his partial solution. All three components are important in determining how the system should respond to a partial solution proposed by the student.

Unlike Mycin, Guidon employs a mixed initiative interface in order to interact with the student. In such interfaces, provisions must be made for all potential kinds of initiative that the student is allowed to make. This includes being able to refer to an earlier topic and provide more detail, changing the topic, and many other options. Guidon provides for student initiative by making available a number of options. It also indexes tutorial remarks to allow the student to easily refer to them later. There are over thirty of these options, like HELP, HINT, TELLME, IKNOW, etc. Guidon can also accept statements and questions in a longer form, in which case the keyword parser of Mycin is invoked to process the command.[8].

In summary, the system must allow the student to specify what he knows, what he wants to know and what he wants to ignore. Guidon is able to respond differently to various partial solutions indicated by the student with the help of discourse procedures that indicate how the system should behave under various circumstances. The knowledge of these procedures is encoded in a special set of rules, called the tutoring rules, or the T-rules. The T-rules are generally invoked as a packet to achieve some tutorial goal.

Figure 3-4 shows a sample session between a student and Guidon. Before beginning the session, the system presents the student with a case history of the patient. The student's goal is to determine the problem and recommend a treatment. In the subset of the session shown in the figure, the system first gives the initiative to the student who is now expected to either enter a question, statement, or a special option. After the student poses a question, the system responds by finding the relevant rules. The student then selects the HELP option. The system responds by presenting a hint of how to proceed. This response is generated with the help of the T-rules. The next question asked by the student is relevant to the goal the system has posed for him, so the question is answered.

As illustrated by the example, the interaction between the system and the student takes place with the help of a menu like interface, and a rather constrained natural language (much like the one in Mycin's

---

[8] Currently only menus and active regions (windows) are used for selecting information. [Clancey, personal communication 86]

System: You should now proceed to ask question that will enable you
to make a hypothesis about the organisms (other than those seen
on the culture smears) which might be causing the infection

The initiative now goes to the student who may ask for more data, make a hypothesis or use one the special options.

Student: How is the fact that the patient is burnt relevant?

System: The fact that patient-236 is seriously burnt will be used to
conclude that Pseudomonas aruginosa is one of the organisms
which might be present in addition to those seen on the
culture or smears [Rule 578].

Student: HELP

a special option is requested by the student.

System: Try to determine the type of infection: bacterial, fungal,
viral, or TB

initiative returned to the student

Student: What is the patient's wbc?

System: The white count from the patient peripheral cbc is 1-9 thousand.

Figure 3-4: A typical interaction with Guidon
Adapted from [Clancey 84]

question answering module). This interface is not a dialog in the sense of a dialog between two people.

The student is limited in what he can say by the menu of choices available. Here, just like in Mycin, if none of the choices seem exactly correct to the user he can not enter anything more satisfactory.

Another weakness of Guidon is that it attempts to explain a student's reasoning entirely in terms of Mycin's rules. If a student is basing his hypothesis on concepts different from those known to Mycin, Guidon has difficulty dealing with his line of reasoning.

3.1.3 Oncocin

The Oncocin system [Shortliffe et. al. 81] [Shortliffe et. al. 83] [Hickam et. al. 85] is designed to assist physicians with management of patients enrolled in cancer chemotherapy protocols. The two major components of the system are the Reasoner and the Interviewer. The Interviewer is responsible for data acquisition, while the Reasoner generates recommendations for appropriate therapy and tests.

The Interviewer is a mix of both hardware and software. It includes a customized keyboard and high speed display terminal. The display simulates the appearance of a flowsheet that currently appears in the clinical chart. This flowsheet is used to record all the data required for analyzing a given case. As the physician enters the information, the relevant patient data is passed to the Reasoner, which simultaneously considers the case and prepares a recommendation to pass back to the Interviewer for display. A major design goal of Oncocin is to have it used by the physician at the time of a patient's visit. One of the ways Oncocin insures that is by using a flow sheet normally used by the physician in day to day work to gather the required data. Another way is to make the system easily accessible, while providing a variety of hard copy reports. The Reasoner and the Interviewer run in parallel allowing the user to receive prompt attention when entering flowsheet values.

There are several reasons why Oncocin was not built as an Emycin [van Mille et.al. 84] system. Oncocin requires serial consideration of each patient at intervals typically spread over many months. Emycin's data structures are not adequate to support this need. Also Oncocin needs to operate in a data driven mode, which is not well supported in Emycin. Overall, the goal of the Oncocin project is to bring AI systems to clinicians, who are intolerant of systems that are slow or difficult to use. It seems to successfully accomplish that goal by giving the physician the mode of interaction he uses in his daily routine, thus requiring a minimum amount of learning on the part of the user.

3.1.4 Summary

In this section we discussed three expert systems that use a menu or a similar interface as a means of interaction with the user. A menu interface is easy to use and implement. It insures that a system can handle everything a person can input. There is a finite number of predetermined answers to each question, and for each of these answers the system knows exactly what to do. There are, however, several problems with such menu interfaces. A person is very limited in choice of input. If none of the choices provided by the system are adequate, the user can not just give a different answer, however more satisfactory it may be. Moreover, since a menu in effect spans out a tree with many paths, a set of multiple choices a user sees at any given point depends on answers to previous questions. Therefore, if none of the choices presented to users satisfy their needs, they may end up answering the wrong set of questions, going down the wrong path and find it difficult to back up to the point where the wrong choice was made. In order to answer a question the user may need some extra information from the system, but there is no facility for him to ask for that information at an arbitrary point. Although the system may be able to provide such information, in order to acquire it the user would have to have chosen a totally different sequence of multiple choice answers.

For a system to be accepted by a user community a more satisfactory form of interface has to be designed. This need is well demonstrated by the fact that Mycin is not accepted by the clinical community, while Oncocin is; it is currently being used by physicians in the Stanford University Medical Center. Mycin requires a user to go through a long and tedious interaction before it gives a therapy recommendation. Oncocin, on the other hand, collects all the information it needs through a flow sheet filled out by the physician as part of his daily routine. This more familiar form of interaction is easier to accept than a new one. For this particular domain, a flow sheet approach is possible. However in other domains there may not be an appropriate form which is used daily and which can be used to gather all the required information. Natural language is another form of interaction used daily and if designed in a natural way, may be another alternative to the menu interface.

3.2 Expert Systems with Natural Language Interfaces

In this section we look at two systems, Prospector and Xcalibur, that employ natural language as a means of interaction with the user. Natural language is not a very common means of interaction between expert systems and users. Because of the complexity of the the underlying system such interaction is still very limited.

3.2.1 Prospector

The Prospector system [Duda, Gasching and Hart 79] is intended to emulate an experienced exploration geologist. Prospector's knowledge is embodied in several models designed by experts in the field. These models are encoded in an inference network where the nodes of the network correspond to assertions and the arcs correspond to inference rules that specify how the probability of one assertion effects the probability of other nodes in the network. The rules are similar to Mycin's production rules.

At the beginning of a consultation session, the user volunteers information about the site in natural language form. After that, Prospector evaluates the various models in its knowledge base and selects the best matched model for further analysis. At this point, Prospector employs backward chaining similar to that of Mycin. Like Mycin, Prospector queries the user for additional information as this information is needed for rule evaluation. The interaction between Prospector and the user takes place in restricted English. The system employs the Lifer parser, described in section 2.3.1 in this paper, in order to analyze the user's input. This input is then represented as a partitioned semantic network, which is in turn compared to the models already stored in the knowledge base. A transcript of interaction between the system and the user is shown in figure 3-5.

Prospector is one of the first expert systems to communicate with its users in natural language. However, we have not found much published information is available that describes Prospector's natural language module. We can only hypothesize that a very simple and limited set of sentences is accepted by the system based on sample system sessions. During the consultation the user simply describes what has been discovered at a given site by using patterns, built with the help of the Lifer system, of the form "There is <deposit>", "There may be <deposit>", etc. The system responds to user queries with canned text.

```
user:   There is quartz monzonite
system: Quartz monzonite (5)
user:   There is cretaceous dike
```

the system acknowledges all of the users statements as above

```
user:   There is sericite and pyrite
user:   There may be biotite
system: Biotite (2)
user:   Done
```

The user is done volunteering information. Certainty factors are assigned to the volunteered information by the system

Next five different models which were not discarded by the system are presented to the user.

```
system:  Presently, the model in Prospector and their scores are:

1.  KNSD (0.0)
2.  PCDA (0.0)
3.  MSD (.001999)
4.  WSSU (0.0)
5.  MVTD (-4.8)

which of the above would you like to rule out?

user:  None
system:  I am considering the possibility of a PCDA in the target
area.  The following questions are intended to establish the nature of
the petrotectonic setting.

To what degree do you believe that the target area is in a continental
margin mobile belt?

user: -2
```

more questions follow

like Mycin, Prospector allows WHY questions. The answer is not a translation of the relevant rules, but merely canned text.

Figure 3-5: Interaction between Prospector and a user

```
system:  To what degree do you believe that igneous rocks in the
region have porphyritic texture?
user:  WHY
system:  I am trying to establish whether some of the intrusive rocks
in your area have textures suggestive of a hypabyssal to subvolcanic
environment etc.
```

Figure 3-5 Continued

Adapted from [Barr and Feigenbaum 82]

3.2.2 Xcalibur

One of the most robust natural language interfaces written for expert systems is Xcalibur [Carbonell et.al. 83] [Carbonell and Hayes 84]. Xcalibur is designed to provide a flexible natural language interface to the Xsel [McDermott 82] expert system. The Xsel system is the Digital Equipment Corporation automatic salesman assistant which advises salesmen on selection of VAX components and produces a sales order which is passed for automatic configuration to the R1 [McDermott 80] system. Part of the Xsel task is to provide the user with information about DEC components, hence subsuming the database query task. Unlike a pure data base query system, Xcalibur also deals with commands to modify equipment orders and information requests about its present task or its database of VAX component parts. Figure 3-5 provides an overview of the Xcalibur system.

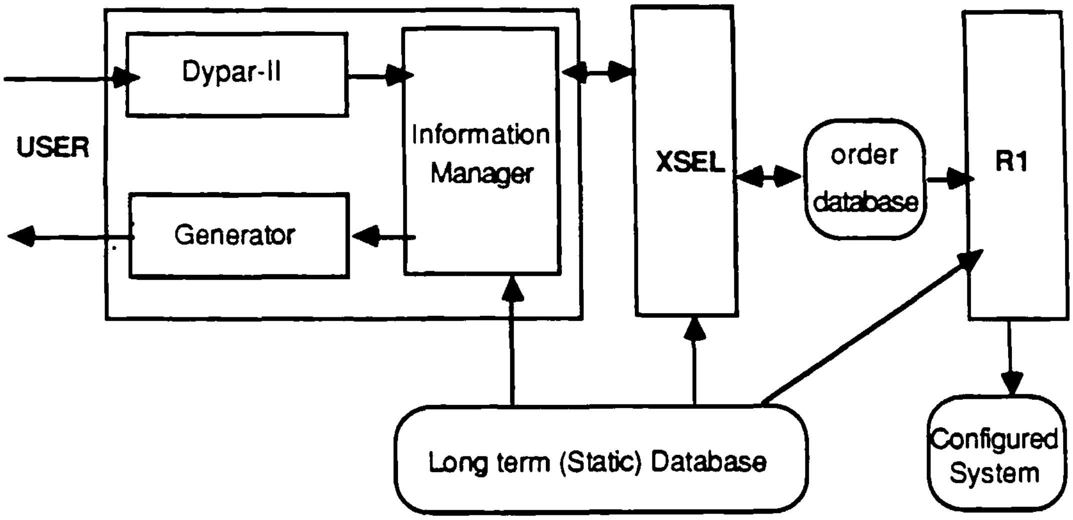

Figure 3-6: Overview of Xcalibur and underlying expert systems
Adapted from [Carbonell et.al. 83]

The Xcalibur system consists of three major parts; the DyparII parser, the Information Manager and the Natural Language Generator.

During a session with the system the user enters questions or statements, and the system either carries out the user's command or supplies him with the necessary information. For example, the user may enter a statement such as *Add two rp07-aa disks to my order*, in which case the system simply does so. The user may also ask questions about the parts in the data base, for example, *What is the largest 11780 fixed disk under $40,000?*, to which the system responds by giving the name of the requested part.

The Dypar-II parser incorporates recursive case-frame instantiation, in addition to semantic grammar, pattern matching and global transformation strategies. A case frame in the system consists of a header pattern as well as a set of expectations for possible case fillers. When analyzing a user's query or request, the parser first activates the appropriate case frames, provided a dialog expectation exists at the case frame level. If no dialog expectation exists, a fast scanning unanchored match of the input is applied against header patterns. These header patterns are pre compiled into a discrimination network and each one is associated with a case frame. Next, the appropriate case frames are used to generate syntactic and semantic expectations in order to recognize the rest of the input. The completed parse is then mapped into a form required by the Information Manager. For an example of the parser output see figure 3-7

```
[Query (Object (Select (disk
                         (ports (Value (2)))
                         (disk-pack-type (VALUE (fixed)))
               (Operation (Sort
                            (Type (*descending))
                            (Attr (size))
                            (Number (2)))
               (Project (price)) )
      (Info-Source (*default)) ]
```

Figure 3-7: The output for 'What is the price of the 2 largest dual port fixed media disks?'

The Information Manager is responsible for mediating communication between the underlying expert system, the parser and the natural language generator. It is responsible for filling in defaults and adding fields to the parser output. It also attempts to disambiguate ambiguous attribute names. For example *large* disk may refer to the disk's memory capacity in one context and to the actual dimensions of the disk in a different context. The Information Manager is responsible for deciding the meaning of the word.

Xcalibur can also handle elliptical references. This is done with the help of two types of rules, discourse expectation rules and substitution rules. In the presence of a strong discourse expectation, discourse expectation rules are tried first. If those should fail, contextual substitution rules are applied. Consider the following dialog:

User: Add a line printer with graphics capabilities.

System: Is 150 lines per minute acceptable?

the possible responses the user may enter are:

1. User : No, 320 is better

OR

2. User: Other options for the speed?

OR

3. User: Too slow, try 300 or faster.

The rule below provides the appropriate expectations for the dialog above.

```
IF     the system generated a query for confirmation of a proposed
       value of a filler of a case in a case frame in focus

THEN   expect one or more of the following:
         a. A confirmation pattern
         b. A different, but semantically permissible filler of the case
            frame in question
         c. A comparative or evaluative pattern
         d. A query for possible fillers or constants on possible
            fillers of the case in question.
```

Response 1 in the above dialog satisfies expectations a, b and c, response 2 satisfies expectation d, and response 3 expectations b and c. With the help of this rule, the system can handle any one of the three elliptical references above.

If no discourse expectation exists, or if the discourse expectation rules fail, contextual substitution rules are applied. For example, consider the dialog below.

user: What is the size of the 3 largest single port fixed media disks?

user: And the price and speed?

To resolve the elliptic reference here, the following rule is applied:

```
IF    an attribute name is present without any corresponding filler or
      case header, and the attribute is semantically permissible
      descriptor of the case frame in the SELECT field of the last
```

```
     query in focus,
THEN Substitute the new attribute name for the old filler of the
     PROJECT field of the last query.
```

In summary, the system's coverage of elliptic reference is a superset of the Lifer/Ladder and Planes systems' coverage. Xcalibur seems to surpass most expert systems' interfaces in its degree of robustness and flexibility.

Xcalibur's interaction with the user greatly resembles that of a natural language interface to a data base system. However it can also deal with commands to modify an order, and information requests about its present task or the data base of VAX component parts. Unlike systems such as Mycin, Xcalibur does not do most of the asking. It is not responsible for solving the user's problem, but rather the user has to know what he wants and query accordingly.

3.2.3 Summary

In this part of the paper we looked at two expert systems that employ natural language as an interface with the user. The Prospector system uses the Lifer natural language parser. Its natural language interface is rather limited. In Prospector, just as in Mycin and Guidon, the interface is not the primary focus of the work, and is not very robust.

In contrast, the Xcalibur system is perhaps the most robust natural language interface written for an expert system. It is able to handle natural language input as well as resolve elliptic references. The system uses an extensive parser and has discourse expectation rules for resolving elliptic references. The interface strongly resembles a natural language interface to a data base system.

3.3 Conclusions -- Expert Systems Domain

In this section of the paper we examined several expert systems and their interfaces. Some of them use menus or other constrained form of interaction; some have limited natural language facilities.

First, we reviewed systems with menu interfaces. Menu interface systems have both advantages and disadvantages. They are easy to use and insure that the system can handle everything a user may input. However, the choice of input is limited and recovery if a wrong choice is made is quite tedious. Also, menu interfaces are not what people use in their day to day routine, so it may be harder for a menu driven interface to be accepted by the user community, as was the case with Mycin. On the other hand, Oncocin, a system that did not deviate a great deal from the daily routine, found a greater degree of success. This seems to indicate that natural language should be a good interface as well because natural language is the most comfortable and natural form of daily interaction.

We also looked at two systems that employ natural language as a means of interacting with the user. One of the systems, Prospector, uses the Lifer system to build a set of patterns for its natural language interface. Although the system has been successfully used in the field, the possible interactions are rather constrained. The second interface we examined, Xcalibur, resembles that of a natural language interface to a data base system, although it does provide more capabilities .

In summary, we have seen the inadequacies of menu interfaces. We have also seen that users are more willing to use systems that use interfaces most familiar to them from their daily routines, as is the case with Oncocin. The two systems that use a form of a natural language interface are also accepted by the user community. These results indicate the need for more work in natural language interfaces for Expert Systems.

4. Conclusions

In this paper we examined a number of interfaces to both data base and expert systems. Natural language interfaces have been successfully constructed to data base systems. However, the design of natural language interfaces to expert systems is only in its initial stages. This can perhaps be attributed to the difference between the two types of underlying systems, as well as to the difference in requirements on natural language interfaces to the two types of underlying systems. Also, data base systems have been around much longer then expert systems.

A data base system is not expected to know or solve a user's problem, but only supply the information that the user requests. The user is solely responsible for a problem solution. Consequently, an interface to a data base system must simply be able to retrieve information requested by the user. On the other hand, an expert system is designed to be a problem solver. It has to 'know' the user's intentions and gather information in order to come up with a solution. Therefore, a natural language interface to an expert system must be able to take initiative in guiding the user toward a solution to a problem. The interface must be able to derive a user's goal (i.e. what the problem that the user is trying to solve is), as well as facts from any given question and add these facts to the data base. The action of extracting a goal and adding facts at the same time has no analogy in a data base system, but would be similar to allowing the user to query and update the data base at the same time. The addition of new information with every user statement means that the system has to pose fewer questions and that the natural language interface is now responsible for managing all the new information.

The existing interfaces to expert systems indicate several requirements for robust natural language interfaces. Such interfaces have to allow users to volunteer information at any time during the interaction as well as to change the focus of a conversation. The user must have more control during the interaction in order to arrive at the answers to his queries in the most informative and least time consuming way. The answers provided by the system should also be more than just a display of the relevant rules.[9]

Issues such as linguistic and conceptual coverage, as well as domain independence apply to natural

---

[9] Xcalibur does not meet the above requirements partially because it is not designed to solve a particular problem, but rather to carry out the user's requests.

language interfaces to expert systems as well as to interfaces to data base systems. The language of a set of users has to be appropriately covered in both types of interfaces. Also, concepts that are expected by a set of users have to be appropriately covered by both types of interfaces. Domain independence is a desirable property for any interface, because an interface that does not depend on specific domain knowledge can be moved form one underlying system to another with only minor modifications.

Linguistic and conceptual coverage, as well as domain independence seem to have been successfully achieved in the data base domain. Can we apply any of the techniques that worked so well in the data base domain to the expert systems domain? The two major techniques we surveyed in this paper are semantic and syntactic grammars. It seems that one of the problems with semantic grammars is that they are inadequate for domains where a large subset of the language is required. This technique is not very useful in the expert systems domain because a robust and domain independent interface to expert systems that would allow for linguistic and conceptual completeness, requires that a large subset of the language be covered. An interface constructed with a semantic grammar would be very limited as well as specific to given expert system. However, we can carry over some of the syntactic grammar approach. The syntactic parser, which does not depend on a given domain, can certainly be adopted, as can be the dictionary. Unfortunately, none of the semantic techniques we described can be directly carried over to the expert systems domain because none of them provide for the special requirements of a natural language interface to an expert system. Most of the semantic techniques rely on the underlying schema describing the data base. Such schema are not available in the case of expert systems. Some modified version of one of the semantic techniques may prove to be applicable. This question is open for further research.

References

[Barr and Feigenbaum 82]
Barr, A. and Feigenbaum, E.A.
The Handbook of Artificial Intelligence Volume II.
William Kaufman, Inc, Los Altos, California, 1982.

[Bobrow and Stefik 83]
Bobrow, D.G., Stefik, M.
The LOOPS Manual
Xerox Corp., Palo Alto, Ca., 1983.

[Buchanan and Shortliffe 84]
Buchanan, B. and Shortliffe, E.
Rule-Based Expert Systems.
Addison-Wesley, 1984, Chapter 32.

[Carbonell and Hayes 84]
Carbonell J., Hayes, P.
Recovery Strategies for Parsing Extragrammatical Language.
Technical Report CMU-CS-84-107, Carngie-Mellon University, 1984.

[Carbonell et.al. 83]
Carbonell J., Boggs W.M., Mauldin M., Anick, P.
Xcalibur Project Report 1.
Technical Report CMU-CS-83-143, Carngie-Mellon University, Digital Equipment Corp., 1983.

[Clancey 84]
Clancey, W.J.
Rule-Based Expert Systems.
Addison-Wesley, 1984, Chapter 26.

[Clancey 79]
Clancey W.
Dialogue Management for Rule-Based Tutorials.
In *Proceedings of IJCAI.* Japan, 1979.

[Clancey, personal communication 86]
Clancey.
1986

[Codd 74]
Codd, E.F.
Seven Steps to Rendezvous with the Casual User.
In Klimbie, J., and Koffema, K. (editor), *Data base Management.* North Holland, 1974.

[Codd 78]
Codd, E.F.
Rendezvous Version 1: An Experimental English-Language Query Formulation System for Casual Users..
Technical Report RJ2144(29407), IBM Research Laboratory, San Jose, California, 1978.

[Date 77]
Date, C.J.
Introduction to Database Systems.
Addison-Wesley, 1977.

[Datskovsky 84] Datskovsky, G.
Menu Interfaces to Expert Systems: Evaluation and Overview.
Technical Report, Columbia University, New York, 1984.

[Datskovsky 85] Datskovsky, G.
Natural Language Interfaces to Expert Systems.
Technical Report, Columbia University, New York, 1985.

[Datskovsky and Ensor 86]
Datskovsky Moerdler, G., Ensor, J.R.
Director -- An Interpreter for Rule Based Expert Systems.
Technical Report, Columbia University, 1986.

[Duda, Gasching and Hart 79]
Duda, R., Gasching, J., Hart, P.
Model Design in the Prospector Consultant System for Mineral Exploration.
In Michie, D. (editor), *Expert Systems in the micro-electronic age.* Edinburgh University Press, 1979.

[Finin, Goodman and Tennant 79]
Finin, T., Goodman, B., Tennant, H.
Jets: Achieving Completeness Through Coverage and Closure.
In *Proceedings of IJCAI.* Japan, 1979.

[Forgy 81] Forgy, C.
OPS5 User's Manual
Carnegie-Mellon University, Pittsburgh, Pa., 1981.

[Green et. al. 63] Green, B., Wolf, A., Chomsky, C., Laughery,K.
Baseball: An Automatic Question Answerer.
Computers and Thought.
McGraw-Hill, 1963.

[Grosz et. al. 85] Grosz, B.,Martin, P., Appelt, D., Pereira, F.
Team: An Experiment in the Design of Transportable Natural Language Interfaces.
Technical Report, SRI International, 1985.

[Harris 77] Harris, L.
User Oriented Data Base Query with the ROBOT Natural Language Query System.
International Journal of Man-Machine Studies 9 , 1977.

[Hendrix et. al. 78]
Hendrix, G., Sacerdoti, E., Sagalowicz D., Slocum J.
Developing a Natural Language Interface to Complex Data.
ACM transactions on Database Systems , 1978.

[Hickam et. al. 85]
Hickam, D., Shortliffe, E., Bischoff, M., Scott, A., and Jacobs, C.
The Treatment Advice of a Computer-Based Cancer Chemotherapy Protocol Advisor.
Annals of Internal Medicine, Vol. 103, No. 6 , 1985.

[Kaplan 78] Kaplan, J.
Indirect Responses to Loaded Questions.
In *Proceedings of the Second Workshop on Theoretical Issues in NLP.* Urbana, Ill., 1978.

[Kaplan 79] Kaplan, S.J.
Cooperative Responses From a Portable Natural Language Data Base Query System.
PhD thesis, University of Pennsylvania, 1979.

[Lederberger 82] Barr, A., Feigenbaum, E.
Handbook of Aritificial Intelligence.
William Kaufmann Inc., 1982, pages 106-110.

[Martin, Appelt and Pereira 83]
Martin, P., Appelt, D., Pereira, F.
Transportability and Generality in a natural Language Interface System.
In *Proceedings of IJCAI.* 1983.

[McDermott 80] McDermott, J.
R1: A Rule-Based Configurer of Computer Systems.
Technical Report, Carngie-Mellon University, 1980.

[McDermott 82] McDermott, J.
Xsel: A Computer Salesperson's Assistant.
Machine Intelligence , 1982.

[McKeown 82] McKeown K.R.
Generating Natural Language Text in Response to Questions About Database Structure.
PhD thesis, University of Pennsylvania, 1982.

[McKeown 86] McKeown, K.
Communication with Expert Systems.
In *Proceedings of JCC.* 1986.

[Rich 84] Rich, E.
Natural Language Interfaces.
IEEE , 1984.

[Scott et al 84] Scott, A.C., Clancey, W.J., Davis, R., and Shortliffe, E.H.
Rule-Based Expert Systems.
Addison-Wesley, 1984, Chapter 18.

[Shortliffe 76] Shortliffe, E.H.
Mycin: A rule-based computer program for advising physicians regarding anitimicrobial therapy selection.
PhD thesis, Stanford University, 1976.

[Shortliffe et. al. 81]
Shortliffe, E., Scott, A.C., Bischoff, M., Campbell A.B., Van Melle, W., Jacobs C.
Oncocin: An Expert System for Oncology Protocol Management.
In *Proceedings of IJCAI.* 1981.

[Shortliffe et. al. 83]
Shortliffe, E., Scott, A.C., Bischoff, M., Carlson R.W., Van Melle, W., Jacobs C.
Integration of a Computer-Based Consultant into the Clinical Setting.
IEEE , 1983.

[Simmons 65] Simmons, R.
Answering English Qestions by Computer: A survey.
CACM 8 , 1965.

[Simmons 66] Simmons, R.
Storage and Retrieval of Aspects of Meaning in Directed Graph Structures.
CACM 9 , 1966.

[Stolfo and Vesonder 82]
Stolfo, S., Vesonder, G.
Ace: An expert system supporting analysis and management decision making.
Technical Report, Columbia University, New York, 1982.

[Tennant 81] Tennant, H.
Natural Language Processing.
Petocelli Books, USA, 1981.

[Tennant et. al. 83]
Tennant, H., Ross, K., Saenz, R., Thomsopson C., Miller, J.
Manu-Based Natural Language Understanding.
In *Proceedings of twenty first ACL conference.* 1983.

[Van Melle 81] Van Melle, W.
The Emycin Manual.
Technical Report Stan-cs-81-885, Stanford University, Stanford, Ca., 1981.

[van Mille et.al. 84]
van Mille, W., Shortliffe, E. and Buchanan B.
Rule-Based Expert Systems.
Addison-Wesley, 1984, Chapter 15.

[Waltz 75] Waltz D.L.
Natural Language Access to Large Data Base System.
In *Proceedings of the IJCAI.* 1975.

[Waltz 77] Waltz D.L.
Writing a Natural Language Data Base System.
In *Proceedings of the IJCAI.* Cambridge, Mass, 1977.

[Winograd 72] Winograd, T.
Understanding Natural Language.
Cognative Psychology , 1972.

[Woods 67] Woods W.A.
Semantics for a Question Answering System..
PhD thesis, Harvard University, 1967.

[Woods 68] Woods, W.A.
Procedural Semantics for Question-Answering Machine.
In *Proceedings of the Fall Joint Computer Conference.* 1968.

[Woods 70] Woods, W.A.
Transition Network Grammars for Natural Language Analysis.
Computational Linguistics , 1970.

[Woods 73a] Woods, W.A.
An Experimental Parsing System for Transition Network Grammars.
Natural Language Processing.
, 1973.

[Woods 73b] Woods, W.A.
An Experimental parsing System for Transition Network Grammars.
In Rustin (editor), *Natural Language Processing*. Algorithmic Press, 1973.

[Woods, Kaplan and Nash-Webber 72]
Woods W., Kaplan R., Nash-Webber B.
The Lunar Sciences Natural Lnaguage Information System: Final Report..
Technical Report 2378, BBN, Cambridge, Mass, 1972.

[Yu et. al. 79] Yu, V.L., Buchanan, B., Shortliffe, E., Wriath S., Davis, R., Scott A., Cohen, C.
Evaluating the performance of a computer-based consultant.
Computer Programs in Biomedicine , 1979.