

DEVELOPMENT OF A NATURAL LANGUAGE INTERFACE  
TO A SLEEP EEG DATABASE

BY

CHONGTAI KIM

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL  
OF THE UNIVERSITY OF FLORIDA IN PARTIAL  
FULFILLMENT OF THE REQUIREMENTS FOR THE  
DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1990

## ACKNOWLEDGEMENTS

This work would have been impossible without the guidance, endurance, and a long period of support of my adviser, Dr. Jack R. Smith. I sincerely appreciate his advice, comments, and thoughtful instructions during my graduate life.

Dr. Jose Principe helped me a great deal with his kind advice and guidance. He enabled me to convey the result of my research to readers of this dissertation. Dr. Antonio Arroyo's interest and comments clarified my work. I would like to thank Dr. John Staudhammer for his kindness and helpful comments. I also appreciate Dr. Sharma Chakravarthy's enthusiasm and helpful comments on my work.

Discussion with Tae-whan Yoon encouraged me very much. I thank Margaret Green and Russ Walters for their editorial assistance with this dissertation. Thanks are also due to Seung-hun Park and Haan-go Choi for their encouragement and cooperation for years in the Sleep EEG Research Lab at the University of Florida.

Especially, I would like to thank my parents and my family for their endless love and endurance.

## TABLE OF CONTENTS

	<u>page</u>
ACKNOWLEDGEMENTS . . . . .	ii
ABSTRACT . . . . .	v
CHAPTERS	
1 INTRODUCTION . . . . .	1
2 PREVIOUS NATURAL LANGUAGE QUERY SYSTEMS . . . . .	8
2.1 The Conventional Approaches . . . . .	9
2.2 Problems in the Conventional Approaches . . . . .	16
2.3 The Knowledge-Based Approach . . . . .	19
3 A MODEL FOR A NATURAL LANGUAGE QUERY SYSTEM . . . . .	22
3.1 The Overall Structure for the Model . . . . .	23
3.2 Representing Domain Concepts . . . . .	27
3.3 Parsing Natural Language Requests . . . . .	29
3.4 Inferential Analysis . . . . .	44
3.5 Query Translations . . . . .	49
3.6 Database Mapping . . . . .	57
3.7 User-Friendly Design . . . . .	60
3.8 Summary . . . . .	72
4 SEEGER: A SLEEP NATURAL LANGUAGE QUERY SYSTEM . . . . .	74
4.1 The Overall Structure of SEEGER . . . . .	75
4.2 A Sleep Representational System . . . . .	81
4.3 A Sleep Database System . . . . .	88
4.4 Conceptual Coverage . . . . .	92
4.5 Linguistic Coverage . . . . .	103
4.6 Implementation . . . . .	112
5 EVALUATION OF SEEGER . . . . .	115
5.1 Test Descriptions . . . . .	115
5.2 Performance Summary . . . . .	116

5.3	The Sleep Database . . . . .	130
5.4	Timing . . . . .	132
6	SUMMARY, CONCLUSIONS AND GUIDELINES FOR FUTURE DEVELOPMENT . . . . .	135
6.1	Summary . . . . .	135
6.2	Conclusions . . . . .	138
6.3	Guidelines for Future Development . . . . .	141
APPENDICES		
A	USER'S GUIDE TO SEEGER . . . . .	144
B	REQUESTS FROM USERS IN THE TEST OF SEEGER . . . . .	154
C	DEVELOPER'S GUIDE TO SEEGER . . . . .	158
REFERENCES . . . . .		162
BIOGRAPHICAL SKETCH . . . . .		167

Abstract of Dissertation Presented to the Graduate School  
of the University of Florida in Partial Fulfillment of the  
Requirements for the Degree of Doctor of Philosophy

DEVELOPMENT OF A NATURAL LANGUAGE INTERFACE  
TO A SLEEP EEG DATABASE

BY

CHONGTAI KIM

May, 1990

Chairman: Dr. Jack R. Smith  
Major Department: Electrical Engineering

A natural language (NL) system called SEEGER (Sleep EEGEr) has been developed to retrieve information from a sleep database without requiring any database expertise or computer programming knowledge of the operator. A new knowledge-based structure for building a NL system has been developed and designed into SEEGER. It provides a more convenient method for incorporating domain knowledge than the previous approaches to building NL systems. In addition, it provides a more efficient database mapping method for those NL requests that rely on bottom-up parsing more often than top-down parsing. SEEGER handles ungrammatical input, makes inferences, and disambiguates some complicated word senses. It engages in an interactive dialogue to correct spelling errors, to clarify ambiguous or incomplete queries, and to enter a synonym when

a word is not in the lexicon. The database is then retrieved by retrieval query created from SEEGER. The sleep database has been designed to contain information about subject record, channel recording, epoch summaries of waveform occurrences in EEG/EOG/EMG, sleep stage parameters, and night parameters. It has been implemented with the commercially available dBASE IV database management system.

SEEGER is now in the experimental stage, and it was evaluated using four users familiar with sleep data analysis. It gave correct answers at the performance level of 63%, achieving the preliminary design goal. The entire program has been written in about 10,000 lines of LISP. The system performance was analyzed in terms of the elements of the syntactic variations, semantic complexities, and interactive dialogues based on the test results. These tasks show that future improvements can lead to a sleep NL system performing at the level of a data processing technician, but it would take at least two man-years of effort, assuming the developers already possess some background in artificial intelligence programming. SEEGER has been implemented in a personal computer; it requires 6 megabytes of system memory and 15 megabytes of hard disk storage for the LISP environment and dBASE IV. Finally, the guidelines for future development are suggested.

## CHAPTER 1 INTRODUCTION

An automatic sleep waveform analyzing system (Smith, 1986; Principe & Smith, 1986; Chang, 1987) detects the most commonly occurring waveforms in the human sleep electroencephalogram (EEG), electrooculogram (EOG), and electromyogram (EMG). It produces a large amount of data which requires further reduction. Sleep clinicians or researchers have diverse purposes for utilizing the sleep data. One of the most flexible and convenient ways to manage sleep data is through the use of a database management system (DBMS). It provides efficient storage and manipulation of data, but knowledge about the database (DB) structure and formal query language is required in order to manipulate the database. The user has to learn the formal query syntax as well as DB file names and DB attribute names. Unless he uses the database often, he is likely to forget them. It imposes an unacceptable time commitment to most sleep clinicians or researchers.

Menu systems are much simpler to learn than DB query systems. They present the user with a sequence of fixed choices easy to recognize and to move through the options. It

is simpler and easier than the syntactic construction of a query language. However, it would not be adequate, in prescribed choices, to accommodate the variety of tasks that the sleep clinician might want to perform.

The main purpose of this research is to develop a NL interface (SEEGER) performing at the level of a data processing technician who provides the data in a sleep database for the sleep clinician or researcher. A natural language interface to the sleep database can solve the problem of training requirements for programming languages, DB structures, and formal DB syntax. An automatic language analysis program takes a NL request as input and transforms it into the machine's internal form which represents the real world content. This content is converted to the query language and then evaluated using the database. Therefore, automatic query language generation makes training unnecessary and enables the sleep clinician/researcher to access the information efficiently and effectively.

A great deal of research has already been done in building natural language interface (NLI) to DB query systems. The NLI systems have been developed to incorporate more and more knowledge and support intelligent language-processing tasks. In addition, considerable effort has been expended to isolate domain-independent components in order to enhance the portability to new domains. The approaches to building the NL query systems can be considered as different choices of what



knowledge is to be applied and in what manner and how it is to be interpreted. Many systems produce an intermediate stage that is called "intermediate representational language" (IRL) in order to refer to the language in which sentence meaning is expressed.

In the semantic-grammar-based approach, NL questions are transformed directly into a query language (QL) or a query form without using any intermediate stage. All knowledge sources are incorporated into a grammar so that a very large number of grammars are required to cover more general syntactic variations.

The domain-independent IRL-free approach transforms directly NL input into a QL without using an IRL stage. It has the same mapping style as the semantic-grammar-based approach, but the knowledge sources are stored in a lexicon. The direct transformation of NL strings into a QL causes the problem of inference which occurs at the level of conceptual content.

The domain-independent IRL approach constructs three separate representations: a syntactic construction, an IRL expression, and a QL. It utilizes the modular approach in which syntax and semantics are separable. This approach shifts the burden of determining sentence meaning to subsequent semantic interpretation. But semantic analysis faces the problems of prepositional phrase attachment, noun phrase meaning, pronouns, and ellipses.

The knowledge-based (KB) approach transforms a NL request into an IRL expression directly without its independent syntactic construction. The advances in representational strategies, knowledge structures, and expectation-based parsing techniques make it possible to incorporate much more domain-specific knowledge into a specific application than is possible in the previous approaches. Syntactic, semantic, pragmatic, and contextual types of information are utilized in an integrated fashion to build an IRL during parsing. The inferential analysis is performed on the IRL to infer the implicit information. This IRL is independent of the physical DB structure and is translated into a QL.

A new model for a knowledge-based approach has been developed in this dissertation. It provides a more convenient method for incorporating domain knowledge than the previous approaches to building NL systems. In addition, it provides a more efficient database mapping method for those NL requests that rely on bottom-up parsing more often than top-down parsing. As a general model, it was applied to build a sleep NL query system that is SEEGER.

A word meaning is combined with other word meanings so that a whole meaning structure of the sentence is formed. Hence, it is necessary for the word meaning to be represented in terms of the domain concepts. The representational system of the sleep domain is represented in a semantic network called "ISA-hierarchy." It reflects the property of the

concepts to be classified and the operations to be performed on it. The top concept is classified into four categories: entities, actions, states, and relationships. These are further broken down successively into their subordinate classes until each item arrives at a primitive component that cannot be further analyzed. Thus, the word meaning is represented into a set of primitive units.

The goal of the conceptual analysis in the knowledge-based approach is to map NL input into a Conceptual Dependency (CD) (Schank & Abelson, 1977) representation of its meaning. It is based on the integrated approach in which syntax and semantics are jointly applied in order to construct the meaning representation without an independent syntactic phase. The knowledge of how to process words is written in the form of "demons." Demons are primarily used to link a fragmentary structure into another in the working memory or are used to disambiguate word senses. A noun group is processed independently to decide the boundary, the head of a noun group, and then finally the meaning of the noun group constituent. At the end of a sentence, a single final structure will be constructed to represent the meaning of a sentence.

The representation produced by the parser is a meaning representation of what was explicitly stated in the user's requests. The inferential analysis is applied to the level of meaning representation by a set of rules. It specifies

implicit information explicitly, removes unnecessary information, and shifts the meaning of concepts by combining or expanding concepts.

Since there is a large gap between the meaning representation and its retrieval query, an intermediate stage of transformation is needed. The meaning representation is first transformed into a set of linear specifications. This involves grouping the same kind of features into one place. Some patterns in the linear forms are replaced with other patterns that can be transformed into the underlying DB language. The set of linear specifications are transformed into a target DB query language using a domain-to-database mapping table.

The sleep DB contains information about subjects, channel recording, 30-second or 60-second epoch-wise summaries of waveform occurrences in EEG/EOG/EMG, and stage-wise and night-wise parameters. The sleep DB is consulted by retrieval queries from SEEGER or edited by a menu-driven system. The menu-driven system is used to add new subject data or update the existing data in the DB. The epoch-wise, stage-wise, and night-wise data of a new subject are supplied from the sleep waveform analyzing system and added in the DB automatically.

Naive or infrequent users often need to be guided with an interactive dialogue to minimize user frustration. This facility requires a considerable domain-specific knowledge. Context-sensitive spelling correction is performed to correct

misspelled words quickly and easily. When a word cannot be found in the lexicon, the user can enter a synonym. When there is an ambiguity or missing information, a clarification dialogue is initiated. Finally, the user is given an opportunity to verify the system's interpretation.

The goal of this dissertation is to develop a model of a knowledge-based query system. An experimental system has been designed and implemented based on a new knowledge-based approach, and it was evaluated by four casual users. Chapter 2 describes the approaches of the previous NL query systems. Their structures for applying knowledge and their limitations are discussed. Chapter 3 describes a new model for a knowledge-based NL query system. It includes the conceptual parsing, the inferential analysis to obtain the meaning of NL input, the transformation of the meaning representation into query retrieval, and the user-friendly design. Chapter 4 describes the design and implementation of SEEGER, a NL system based on the model of Chapter 3. The system structure and a sleep DB system are included. The elements covering a variety of syntactic and semantic facilities for sleep domain are discussed along with the current implementation of SEEGER. Chapter 5 describes the evaluation of SEEGER. It includes the test description and a summary of SEEGER's performance. Chapter 6 contains the summary, conclusions, and the guidelines for future development.

## CHAPTER 2 PREVIOUS NATURAL LANGUAGE QUERY SYSTEMS

A great amount of research has been done in building NLIs to database (DB) query systems. Although there is no general agreement on research goals and methodology in natural language processing (NLP), the main line of development in NLP has been in the incorporation of more different sources of knowledge in support of intelligent language-processing tasks. Therefore, NLIs have been built in a direction to incorporate more knowledge so that users will not require specialized training to be able to use the system. In addition, a considerable effort has been made to isolate domain-independent components in order to enhance the portability to new domains.

In Section 2.1 the approaches taken by the previous NLIs are described in a view of what knowledge is applied and in what manner, providing an evolutionary trace of the NLI systems. Problems encountered with the previous NLI systems are discussed in Section 2.2. Section 2.3 describes the current existing knowledge-based approach.

## 2.1 The Conventional Approaches

A NLI system should utilize at least three different kinds of knowledge to accomplish its task: syntactic knowledge, domain-specific knowledge (or semantic knowledge), and knowledge of database structure. Domain-specific knowledge includes lexical knowledge, world knowledge, and contextual knowledge. All the NLI systems discussed in this chapter have the goal of producing a query language. A query language is either a formal database language or an intermediate form prior to a formal database language. The approaches to building the NLI systems can be considered as different choices of what knowledge is to be applied and in what manner, and how it is to be interpreted. The NLI systems have taken one of the following approaches: LUNAR-Like, semantic-grammar-based, or domain-independent. The characteristics of each approach are described below.

### 2.1.1 LUNAR-like Approach

This approach takes two separate procedures: syntactic and semantic analysis as shown in Figure 2.1.

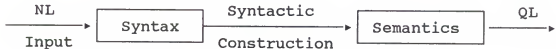


Figure 2.1. The Structure of LUNAR-Like Approach

A domain-independent grammar produces a syntactic construction which is translated directly into a query language without any intermediate stages. It is not transportable to other databases or applications.

One of the earliest and widely known systems is Woods's LUNAR (Woods, Kaplan, & Nash-Webber, 1972). It was the first NLI to real world problems. It allows geologists to conveniently access the chemical analysis data on lunar rocks obtained from the Apollo-11 mission. LUNAR translates NL questions into a query language in the following two separated steps: 1) syntactic analysis using an augmented transition network (ATN) parser to produce a syntactic construction or a parse tree, 2) semantic interpretation to map the syntactic construction into a query language. The ATN grammar (Woods, 1970) is independent of a specific domain and transferable to other applications. All of the domain-specific information is contained in the lexicon and the semantic interpretation rules. The lexicon entries contain a variety of syntactic, semantic, and morphological information about the words. In LUNAR, semantic knowledge and database structure knowledge are merged into one in the semantic rules.

In a LUNAR-Like approach, a new set of semantic rules must be rewritten for a new database as well as a new domain since database structure knowledge and domain-specific knowledge are combined into the semantic rules.



### 2.1.2 Semantic-Grammar-Based Approach

In this approach, NL questions are transformed into a query language directly without any intermediate stages as shown in Figure 2.2. As a domain-specific approach, a grammar is designed for just one application use. The main characteristic of a semantic grammar (Burton, 1976) is that it intentionally embodies all knowledge sources into a grammar. It classifies words and phrases into a combination of syntactic, semantic, and database knowledge. Thus, a semantic interpretation can be performed during the parse without additional stages of processing. There are two typical examples: LADDER and PLANES.

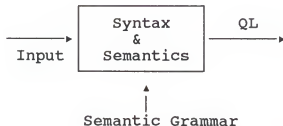


Figure 2.2 The Structure of Semantic-Grammar-Based Approach

LADDER (Hendrix, Sacerdoti, Sagalowicz, & Slocum, 1978) provides NL access to a distributed DB of Navy ship information. The system used LIFER (Hendrix, 1977) as a development tool. LIFER compiles a set of user-defined grammar rules into a semantic grammar because the specific semantic entities are incorporated directly into the grammar. This

system can translate a query into an equivalent program language or a query language.

PLANES (Waltz, 1978) answers NL questions from a large relational DB consisting of maintenance and flight records of Navy aircrafts. The parsing mechanism of a sentence or phrase is based on an ATN formalism. That is, one state of ATN at the top level calls each ATN subnetwork repeatedly during analysis of a sentence. A subnetwork recognizes a phrase or basic meaning unit. But the basic unit is mapped directly into a part of a query language. The notion of direct mapping of a subnetwork is the same as that of semantic grammar. Therefore, syntactic, semantic and database knowledge is imbedded in each ATN subnetwork so that new subnetwork grammars must be rewritten for other domains.

In a semantic grammar approach, all knowledge sources are incorporated into a grammar and there is no separate semantic interpretation phase in order to obtain a query language. A grammar is not transportable and can only be applied to one application.

### 2.1.3 Domain-Independent Approach

Since the late 1970s, many NLI's have been developed as transportable domain-independent systems to minimize development costs. This approach isolates the domain-dependent information, providing modular design and greatly enhancing the portability to new domains. Many systems produce an IRL

of a question in terms of the concepts of a domain, independent of the DB structure. The systems in this approach can be divided into two systems, depending on the mapping methods: systems without IRL (or IRL-free systems) and systems with IRL (or IRL systems).

2.1.3.1 The IRL-free systems. The systems in this approach transform NL questions directly into a query language as shown in Figure 2.3. This mapping style is the same as that of the semantic grammar systems, but the knowledge sources are stored in a different place. The knowledge is stored at the lexicon in the IRL-free systems, but all knowledge sources are stored at the grammar in the semantic grammar systems.

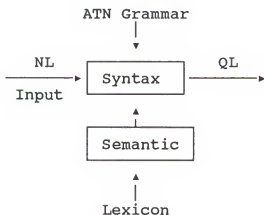


Figure 2.3 The Structure of IRL-free Domain-Independent Approach

In this approach, the linkage between natural language strings and query language code is defined in a lexicon. Typically a dictionary has two parts, the domain-independent

lexicon and the domain-specific lexicon. The first defines words and phrases which have the same query language linkage for any application. A domain-specific word or phrase is defined as its corresponding field name, field value, or database file from the underlying database. Synonyms are also defined in the lexicon. Therefore, domain-specific knowledge is mainly composed of database structure knowledge.

INTELLECT, previously called ROBOT (Harris, 1977), one of the commercially available NL query systems, has a domain-independent grammar based on ATN and translates input words into references to fields, files, or key words to a query language. COOP (Kaplan, 1979) and EUFID (Templeton & Burger, 1983) also used this approach.

This approach can be transportable to other applications. Since a grammar in this approach is domain-independent, it involves defining a new domain-specific lexicon to support a new target database.

2.1.3.2 The IRL systems. The IRL systems construct at least three separate representations of NL questions: a syntactic construction, an IRL expression, and a query language as shown in Figure 2.4.



Figure 2.4 The Structure of IRL Domain-Independent Approach

It is an IRL expression that represents the meaning of a sentence. It is necessary for an IRL to be decoupled from the underlying database because there may be NL sentences for which no corresponding language exists.

PHLIQA1 (Scha, 1982) focused on the formal definition of representations. It used the formal logic language for a syntactic construction, IRL, and a query language. The representation of each step is shifted to the next step representation by the mapping rules of the formal logic.

CHAT (Warren & Pereira, 1982) translates a sentence into a "first-order logic," and then a logic form in PROLOG. This system used the Prolog language since it allows efficient implementation.

Ginsparg (1983) developed a portable NLI at Bell Laboratories. The meaning of a sentence is represented in a case frame form rather than a logic form, and then it is translated into a query language by a set of DB mapping rules.

TEAM (Grosz, 1983) was designed to interact with two different kinds of users: a database person to acquire DB information and an end user to retrieve information from the DB. The knowledge acquisition is performed automatically by interactions with a database person, rather than hand-built by programmers. The intermediate stages are expressed in a logical form.

The systems in this approach produce an IRL expression which is independent of any database as well as any domain. It is therefore possible for them to construct an IRL more expressive than a query language.

## 2.2 Problems with the Conventional Approaches

As discussed in the last section, there have been several different approaches to designing NL query systems. The limitations of each approach are described here. Their limitations are due to the lack of knowledge or the lack of mechanism to incorporate various knowledge sources. The problems in the domain-independent approach are illustrated with specific sentences.

### 2.2.1 LUNAR-Like Approach

This approach encounters the same problems as the domain-independent approach since it uses a domain-independent ATN parser.

### 2.2.2 Semantic-Grammar-Based Approach

This approach has built the successful NLI systems such as LADDER and PLANES in the restricted subject area. All knowledge sources are encoded in a grammar to guide the interpretation of NL questions. It provides capabilities for handling ellipses, ungrammatical input, pronouns, paraphrases, spelling corrections, etc. In addition, many ambiguities can

be avoided during parsing by storing all the allowed patterns in a grammar.

A new grammar must be rewritten for a new application. It can be useful to build a restricted NLI quickly. The main limitation is that a very large number of grammar rules are required to cover more syntactic variations. In other words, it is very difficult to capture important syntactic generalizations in a semantic grammar. Adding more rules requires a high level of training and a considerable effort for interface developers as well.

### 2.2.3 Domain-Independent Approach

There are two controversial NLP theories: modular and integrated. The domain-independent approach is based on modular theory, and the knowledge-base approach is based on integrated theory. The modular approach says that syntax and semantics are separable and an autonomous level of syntactic representation must be computed. This position was supported by Chomsky (1965), Woods (1970), Winograd (1972), and Marcus (1980). In contrast, the integrated approach advocates that syntax and semantics be applied jointly in integrated fashion and no independent syntactic representation is constructed. This was advocated by Schank (1973), Riesbeck (1975), Birnbaum (1986), and Riesbeck and Martin (1986).

Birnbaum (1986) criticized the theory of the modular approach because it lies not in a goal-directed functional

view of language but rather in a descriptive view of language taken by linguistics. He said that ATN models are not true process models for language analysis because they are heavily dependent on arbitrary choice of search and backtracking. It is impossible to correctly determine how constituents of utterance are related using syntactic information alone. Marcus's nondeterminism (Marcus, 1980) also fails to address the problem of ambiguity.

The syntactic analysis in the IRL domain-independent systems is mainly performed by ATN or its descendant parsers. This approach shifts the burden of determining sentence meaning to subsequent semantic interpretation. Semantic interpretation transforms a syntactic construction into an IRL expressed in the concepts such as objects, relations, and predicates of domain model. But semantic analysis faces the problems of prepositional phrase attachment, noun phrase meaning, scope determination, pronouns, and ellipses. Several different strategies (Perrault & Grosz, 1986) have been proposed.

Grosz (1982) addressed the problems of the lack of inferential capabilities in TEAM. In PHLIQA1, Scha (1982) described that an extensive set of domain-specific rules is required due to the large gap between lexicon and database. Warren (1982) found in CHAT that there is a very large gap between IRL in logic form and database due to the absence of explicit facts in the database.



The IRL-free systems mostly have only database structure knowledge as a domain knowledge in the lexicon. The IRL systems were not incorporated with appropriate inferential, pragmatic knowledge of underlying domain because it seems very difficult to accommodate a mechanism to do it. This very limited domain-specific knowledge causes some severe problems in the domain-independent approach.

### 2.3 The Knowledge-Based Approach

Since the 1970s a great deal of progress has been made toward representational strategies (e.g., logic, semantic network, semantic primitives), knowledge structures (e.g., scripts, plans, goals, MOPs), and expectation-based parsing techniques used to build more recent NLI: EXPLORER (Lehnert & Shwartz, 1983), EASYTALK (Shwartz, 1987). The KB approach transforms NL requests into an IRL expression directly without its syntactic construction as in Figure 2.5.

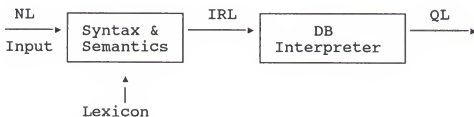


Figure 2.5 Knowledge-Based Approach

This is based on the notion that sentences with identical meaning will have the same underlying conceptual

representation, regardless of differences in grammatical form or language used. Syntactic, semantic, pragmatic, and contextual information are utilized in an integrated fashion to build an IRL during parsing. The inferential analysis is performed on the IRL to infer the implicit information. This IRL is independent of physical structure and is translated into a QL.

EXPLORER is a commercial NL query system for oil exploration which accesses oil well databases and generates maps. These maps are complicated objects, derived from a database and generated dynamically as actual physical maps. EXPLORER was designed to simulate the technician who uses the database to generate maps of particular geographic regions detailing the contour information of the formations within the region as well as the characteristics of the wells within the region. EXPLORER exploits a conceptual parser (Dyer, 1983), combining syntactic and semantic information, along with scripts (Schank & Abelson, 1977) relevant to this domain of oil maps in order to produce an instantiated script as a request representation. The system analyzes an input request and constructs the interpretation by filling the slots in a script or frame. It naturally offers the user an option of inheriting many specifications from the previous map request.

EASYTALK is a commercial NLI system for a business accountant database. It was designed to automate transaction processing for wholesale distributors. It enables the user to

generate reports on an ad hoc basis by simply typing a request for information in English. The report is then stored and accessed through a menu of user-defined reports. This system uses a conceptual parser to obtain a CD representation of requests and then navigates the appropriate relational tables and columns to generate an efficient retrieval query.

The parser used in EXPLORER and EASYTALK allows the rules (or demons) to deal explicitly with information structures representing the current state of the parser. A major problem of the parser is the 'traffic control' of rules (Winograd, 1983) to be activated. To the degree that the parser allows many different rules to be operating in parallel, the parser leads to complex unexpected interactions and can be difficult to debug and understand. In the database mapping process, the KB systems use a set of rules in order to obtain the retrieval queries from the parsing output. But if the structure type of the parsing output is diverse, the number of rules becomes too large to manage. The development cost of KB systems is also very high for a given application (Schank & Shwartz, 1985). The effort to build KB NL query systems will require many more man-years than that of developing portable domain-independent systems.

CHAPTER 3  
A MODEL FOR A NATURAL LANGUAGE QUERY SYSTEM

This chapter describes a new model for a knowledge-based NL query system. In this model, the parser maps NL input directly into a Conceptual Dependency (CD) representation of its meaning without any independent syntactic analysis phase. A word or phrase, which constitutes lexical items in the lexicon, is defined with a set of primitive units. Some lexical items also contain their own processing information in the form of rules called demons. The parser reads NL input from left to right, processing the word definitions and demons in the memory such that a well-formed meaning structure is built. The meaning structure produced by the parser is what was explicitly stated in NL questions. Implicit information will be expressed explicitly by means of the inferential analysis. In this way, the entire meaning of NL questions can be represented.

In this model, the CD representation is transformed into a set of linear specifications which groups the common features into the same place. The linear forms are augmented by a set of rules in order to provide a complete form for

database mapping. A set of domain concepts are then transformed into a target database expression through a concept-to-database function table.

The following section describes briefly the overall structure for a proposed knowledge-based NL query system model. Section 3.2 describes a domain concept representation scheme to represent word meaning and domain knowledge. The most important part of the model is the parser, described in Section 3.3. The inferential process is discussed in Section 3.4. The query transformation is discussed in Section 3.5 and in Section 3.6, and a user-friendly design for an interactive dialogue is described in Section 3.7. Finally, this model is summarized, and the differences from other systems are discussed in Section 3.8.

### 3.1 The Overall Structure of the Model

The structure of the proposed knowledge-based system is shown in Figure 3.1. The components of the structure are described briefly below, and their details are explained in the following sections in this chapter.

#### 3.1.1 Conceptual Parser

The Conceptual Parser (CP) in this model maps NL input into a CD representation of its meaning in the same manner as other knowledge-based systems. The CP, as the integrated approach, is in contrast to syntactic parsers which are in the

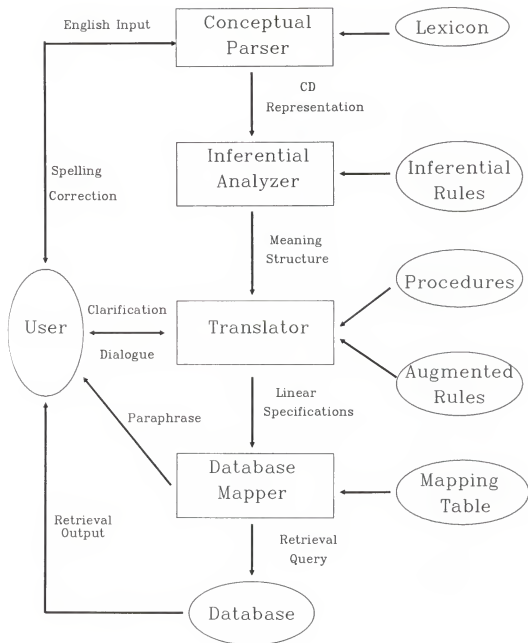


Figure 3.1 The Overall Structure of a Knowledge-Based Natural Language Query Model

modular approach. In the modular approach a separate syntactic structure is built by means of a grammar such as ATN. But the CP does not use any explicit grammar. The knowledge about syntax (i.e., word order) is encoded in the form of demons along with other processing information. Word senses are defined in a declarative form into lexical items in a lexicon. Thus, the lexicon contains demons and word sense definitions. The CP reads NL input from left to right, processing word definitions and demons in the working memory so that a final meaning representation is obtained. The details are described in Section 3.3.

#### 3.1.2 Inferential Analyzer

The output produced by the CP represents the meaning of what was explicitly stated in user requests. However, many requests are expressed with missing information or in ambiguous utterances. Some requests can be understood based on a wide range of information such as contextual components, dialogue conventions, world knowledge, etc. Hence, missing information or what is to be clarified in ambiguous utterances can be inferred.

In this model, the inference is applied to the level of meaning representation by a set of inferential rules. Most inferential rules are domain-dependent contextual rules which embody a lot of domain-specific knowledge. Section 3.4 describes the Inferential Analyzer in detail.

### 3.1.3 Translator

In this model, a NL question has been transformed into a meaning structure (MS) via the conceptual parsing and the inferential analysis. The MS is independent of the physical structure of the database. It should be transformed into the retrieval routine of the underlying database. But the MS consists of domain concepts in the form of a tree structure, and DB language is a non-procedural language.

In this model, since there is a gap between the MS and the target DB language, the MS is first transformed into a set of linear forms of the domain concepts in this model. Then the set of linear forms is augmented by the database knowledge and domain knowledge. Section 3.5 discusses the Translator in detail.

### 3.1.4 Database Mapper

In this model, the set of linear specifications is transformed into a target DB query language by means of a table of mapping functions. This transformation is called "database mapping." The domain-to-database mapping table contains the actual database attribute names as well as access-path information. That is, the mapping table is a function whose input is a domain concept and whose output is a list of its associated DB files and matching attribute names. Section 3.6 describes the DB Mapper in detail.



### 3.1.5 User-Friendly Design

One of the most important requirements in NLI systems is that it interacts in a friendly way with the user to minimize user frustration. Naive or infrequent users often need to be guided with an interactive dialogue to formulate a complete query.

In this model, if there are any typographical errors in a sentence, the system suggests the most likely correct word to the user. When there are any ambiguities or missing information, a clarification dialogue is initiated. Once the system has formulated the query unambiguously, the user will be given an opportunity to verify the system's interpretation. Section 3.7 discusses a user-friendly design in detail.

## 3.2 Representing Domain Concepts

In knowledge-based systems, parsing is a process of representing the meaning of the sentence in terms of domain concepts. A word meaning is combined with other word meanings so that a whole meaning structure of the sentence is formed. Hence, it is necessary for the word meaning to be represented and stored in the system's memory so as to be used during parsing.

How can we represent the meaning of a word in a formal way? Representation can be considered as a process by which a formal, language-independent structure is assigned to the

items of meaning (Schank, 1973). The meaning can be broken down into certain key items within an application domain. The key item is called the "primitive unit" since it is a fundamental component and its meaning is not further analyzed. Thus, the word meaning is composed of a set of primitive units. The choice of a key item is based on the level of detail by which all phenomena of natural language interactions can be covered. This meaning assigning process of a word is called a "conceptualization" (Cullingford, 1986; Schank, 1973).

There are four reasons why a formal representation scheme is needed. First, it assigns a unique formal structure to an identical meaning. That is, there should be only one formal structure for any number of sentences if they have an identical meaning. Second, formal structures are necessary for inferences by which implicit information can be expressed explicitly. Third, formal structure can be used to generate natural language strings. Finally, it can be used to represent the system's own knowledge of its domain. The first two reasons are equal to the axiom of the Conceptual Dependency theory (Schank & Abelson, 1977). Thus, designing a good representational scheme is important for building a NLI system.

Cullingford (1986) suggested three representational criteria: 1) coverage--all the entities and phenomena in the domain can be represented, 2) economy--the smallest set of

primitive types has been chosen, and 3) orthogonality--the primitive types are designed to minimize the overlap between the inferences. These are informal criteria for what representation content constitutes a good representational system.

### 3.3 Parsing Natural Language Requests

This section describes the evolution of the Conceptual Parser and then its structure and operations in detail. The notion of the Conceptual Parser in this model is based on the McDYPAR (Micro-version of DYPAR) (Dyer, 1983) with additional features: noun group parsing, based on the noun group processing method developed by Gershman (1979), morphological analysis, based on the algorithm used by Winograd (1983), phrase and synonym handling, conjunction and elliptic analysis, and spelling correction. They are incorporated in the Conceptual Parser to expand the McDYPAR.

#### 3.3.1 Background of the Conceptual Parser

As described in Chapter 2, there have been two different approaches of parsing NL sentences. One is the syntactic parser advocated by the syntactic autonomous camp and the other is the conceptual parser advocated by the integrated camp. The goal of the syntactic parser is to obtain the parsing tree (or syntactic structure) of a sentence and then interpret the meaning of the sentence from the parsing tree.

This parser thus separates the syntactic and semantic parts of the analysis into two consecutive stages and pays more attention to the syntactic part at the expense of semantics.

In contrast, the conceptual parser takes a sentence as input and produces the meaning representation of the sentence at one pass. It has no independent syntactic analysis phase since meaning is a primary issue at the earliest stage, and syntax can be used to aid in obtaining a meaning. This analysis pays attention only to the words and expression relevant to the tasks.

The basic ideas of the conceptual parser were first developed in the MARGIE system (Riesbeck, 1975) which makes inferences and paraphrases from NL sentences. In the MARGIE parser, the parsing knowledge is encoded in small executable programs, called "requests," which are stored in the dictionary entries. These requests are then used to connect the semantic representation of the word into a representation of the whole sentence.

The next conceptual parser was ELI (Riesbeck, 1978). ELI was an attempt to show how requests could be used both to disambiguate words and to connect structures by making the parser extremely top-down. Thus, a conceptual structure was only accepted for processing if it satisfied prior expectations. This parser was expanded by the addition of the Noun Group Processor program to handle complex noun groups and post-nominal modifiers in ELI-2 (Gershman, 1979).

The Conceptual Analyzer (CA) (Birnbaum & Selfridge, 1981) tried to make homogeneous the kinds of processing performed by ELI and by Noun Group Processor. In ELI, all the requests are kept in one list and everything in the list is tested when new input arrives. In many cases, more than one request is activated at the same time, and hence these requests have the same recency. Such a set is called a "request pool" and an ordered list of request pools was used for flexible control strategies in CA. Almost everything in ELI is controlled in top-down manner, but this top-down control is augmented by better bottom-up control so that more variable input texts are acceptable in CA.

The above parsers were extended and improved in DYPAR (Dyer, 1983) which models expectations by means of processes called demons rather than requests. DYPAR-style demons are less constrained than ELI-style requests. The parser used in this model is an extended version of McDYPAR (Dyer, 1983) with additional features such as noun group processing, morphological analysis, phrase and synonym handling, etc.

### 3.3.2 Data Structures of the Conceptual Parser

During language analysis, the parser in this model uses a memory to hold the structures being processed, which is called the "working memory" (WM). The parser brings up the fragmentary structure of words in the WM one at a time and then integrates them in order to obtain a final representation

of input. Whenever the parser encounters a word or phrase, a new node is created in the WM so that it is appended to the last one. The node thus is a place to hold one structure of a lexical item and its associated demons. The parser traverses every node to test which demons want to fire. In this way an input sentence will be processed by the parser to obtain the meaning representation. The components of the parser will be described below.

3.3.2.1 Lexicon. In this model, the lexicon is a place to hold all the lexical items to be used for language analysis. A lexical item is either a word or a phrase which contains a syntactic category, word senses, demons, and a synonym. The designing of the lexicon is a process to acquire all knowledge around the words being entered.

A systematic way to design the lexicon is suggested by Cullingford (1986). It involves collecting all sentences or sentence fragments which sample as large a fragment as possible of knowledge content of the domain of interest. The lexicon design is based on the observation that many words have a limited number of special meanings in particular contexts. For example, the verbs such as "show, list, or display" probably have a common meaning in database access systems. The word "subject" in the sleep domain, for example, indicates a patient or a sleeping person rather than a topic. In this model, the lexicon is designed in this way, and it is considered to be very useful for a domain-specific approach.

In this model, the lexical item has the following format for a word:

```
(3.1) (WORD      word
      POS      part-of-speech
      DEF      lisp-expression
      SYNONYM  word or phrase
      DEMONS   sequence-of-demons
      M1       lisp-expression
      ...
      Mn       lisp-expression
      PHRASES  list-of-phrase)
```

and for a phrase

```
(3.2) (EXPRESSION phrase
      WORDS    list-of-words
      POS      part-of-speech
      DEF      lisp-expression
      SYNONYM  word or phrase
      DEMONS   sequence-of-demons
      M1       lisp-expression
      ...
      Mn       lisp-expression)
```

A lexical entry is made up of a group of "segments." Each segment begins with a key word (e.g. POS, DEF, DEMONS, etc.). The segment is optional and may appear in any order. POS, PHRASES and SYNONYM segments are proposed to expand the McDYPAR, and the semantics of each segment are described below:

POS: The syntactic category of an entry is used for noun group processing. The noun group processor determines when to start and leave noun grouping mode with this syntactic category.

DEF: The sense of a entry is placed here if it is unambiguous. If it is ambiguous and has n number of senses, this segment should be NIL, and the senses are placed in segments M1 through Mn.

DEMONS: This segment contains the processing or parsing knowledge associated with this entry. The expectation demons of a verb are given in DEF segment, but other demons will be given here.

M1, Mn : The first or n-th sense of an ambiguous word or phrase is placed here. The DEF segment should be NIL, and a disambiguation demon should be given in the DEMONS segment.

PHRASES: A list of all phrases starting with the word is contained here. When this segment is given, the parser tries to match words to see whether a phrase is used in input sentence.

WORDS: A list of words comprising a phrase is placed here.

SYNONYM: If there is a synonym of an entry, it is defined by the lexical entry with the word definition. If this is defined, other segments such as DEF, DEMONS, M1, or Mn are unnecessary.

3.3.2.2 Working memory. In CA, there are two separate lists: one is a short-term memory called CONCEPT-LIST which is a place to hold a list of semantic nodes, the other is called REQUEST-LIST which is a place to hold the active requests.

In this demon-based parser, there is one list (i.e., the WM) which is a place to hold linked semantic nodes while demons are attached under the relevant nodes. In this model, the WM is a short-term memory in the form of queue in which word meanings and associated demons are stored. A node of the queue is called a "semantic node." Since the WM is a doubly-linked queue, it is convenient to search the existing semantic nodes in either direction as well as to insert or delete any semantic nodes. When the parser encounters a word or phrase,



a new semantic node is added in the WM, and its associated demons are added at a place called "demon agenda" under this node. In this model, at the end of a sentence one final independent structure representing the meaning of the sentence will be formed in the WM and then used for the inferential processing.

3.3.2.3 Demons. In this model, the knowledge of how to process words is written in the form of demons. During NL analysis, demons are generally used for the following purposes: 1) to bind a filler with another structure in the WM, 2) to disambiguate word senses, 3) to determine pronoun referents, and 4) to add another demon dynamically in the WM.

A demon, as a test-action pair, is an active process which waits until the condition part is satisfied, whereupon the action part is executed and deactivated. Demons have the following form in this model:

```
(3.3) (DEMON name
      (PARAMS sequence-of-parameters)
      (KILL lisp-expressions)
      (TEST lisp-expressions)
      (+ACT lisp-expressions)
      (-ACT lisp-expressions))
```

A demon consists of a group of segments like the lexicon entry syntax. The semantics of each segment are described below:

PARAMS: This is a place to hold a list of variables which are used in other segments. When a demon is interpreted (i.e., tested, deactivated, or executed), its arguments are bound to its parameters.

KILL: This segment is evaluated before the TEST or the ACT segment. If its result is non-NIL, the demon is "killed" or deactivated (i.e. removed permanently from the demon list of the semantic node and never considered during demon interpretation).

TEST: If its evaluated result is non-NIL, or this segment is missing, then the +ACT segment is fired. Otherwise, the -ACT is fired. If its result is NIL, and the -ACT segment is missing, then the demon remains alive.

+ACT: After this segment is evaluated, the demon is automatically killed.

-ACT: If there is a -ACT, and it is evaluated, then the demon is automatically killed.

### 3.3.3 Control Structure of the Conceptual Parser

The parsing mechanism is based on the integrated approach in which syntax and semantics are jointly applied in order to construct the meaning representation without a separate syntactic phase. In this model, it will be performed via two stages: "word tasks" and "demon tasks."

The overall algorithm of the parser in this model is given below:

```

STEP1    Examine next lexical entry from input - word
          tasks.
STEP2    Interpret demons - demon tasks.
STEP3    If there is no more input, then terminate
          parsing a sentence. Otherwise loop back to
          STEP1.

```

3.3.3.1 Word tasks. The basic control algorithm of the word tasks is described informally as follows:

```

STEP1    Look at next word. If a phrase beginning with
          the word is recognized in the lexicon then go
          to STEP5, otherwise go to STEP2.

```

- STEP2 If the word is recognized in the lexicon then go to STEP5, otherwise go to STEP3.
- STEP3 If the root and suffix of the word are recognized in the lexicon by morphological analysis then go to STEP5, otherwise go to STEP4.
- STEP4 Run the spelling correction routines (described in Section 3.7 so that the word is replaced with a new word and then go back to STEP1.
- STEP5 Create a new semantic node and tokenize the conceptualization and then add it to the WM. If the word or phrase is unambiguous (i.e., the entry has the DEF segment) then bind the node with its conceptualization and spawn its associated demons. Otherwise, set the node to NIL and spawn disambiguation demons under its semantic node.

There are three basic kinds of morphological structure (Winograd, 1983): affixation (e.g., un-, re-, -ify, -ation, etc.), compounds (e.g., lighthouse, hummingbird, etc.), and modified forms (e.g., take, took; goose, geese; etc.). These morphological phenomena show a high degree of irregularity and idiosyncrasy so that only a few of them are handled more systematically by rules and everything else must be stored in the dictionary.

In this model, when a word is not recognized in the lexicon, morphological analysis will try to break up the word according to the spelling rules. For instance, these rules separates the word "running" into its root "run" and its suffix "-ing" so that it is not necessary to have a separate word entry in the lexicon for the inflectional ending of the word.

The morphological analysis in SEEGER is designed to handle a number of inflectional endings: "-n't" for negative, "-'s" and "-'" for possessive, "-s" and its variants for plural nouns and singular third person verbs, "-ing" and "-ed" for verb forms, "-est" for the superlative, and "-er" for comparative. It does not cover all inflectional word endings, but could easily be expanded to manipulate other cases. Exceptional words do not need to be included in the analysis program. The word "has," for example, is directly defined in the lexicon and morphological rules are never applied in trying to find it. If the root is found in the lexicon, its syntactic category is checked to see if the ending is appropriate. There should be negative, "-ing, -ed, and -s" forms with a verb root, plural and possessive forms with a noun root, superlative and comparative forms with an adjective root. The syntactic category of a word is changed when the ending is taken into account. For example, a word with a possessive suffix changes a noun class into an adjective class. This syntactic information is kept in the semantic node when the parser encounters a word.

In this model, the conceptualization in the lexicon is represented as a tokenized form in the WM in which all the fillers as well as a top level are named as a new global symbol. For example, the word "Jones" for a subject name has the following conceptualization:

```
(3.4) (PERSON LASTNAME (JONES)
      GENDER (MALE))
```

When the parser encounters this word a new semantic node (e.g., notated as "CON10") is added in the WM, and it has the following tokenized CD form:

```
(3.5)
CON10: (PERSON LASTNAME LASTNAME11 GENDER GENDER12)
LASTNAME11: (JONES)
GENDER12: (MALE)
```

where PERSON is a head pattern and LASTNAME11, GENDER12 are the tokenized fillers for their slots LASTNAME, GENDER, respectively. The filler named LASTNAME11 binds the CD form "(JONES)" so GENDER12 does "(MALE)." For another example, a verb "show" has the following conceptualization in the lexicon:

```
(3.6) (RETRIEVE ACTOR * <== (EXPECT '*SYS* 'BEFORE)
      TO * <== (EXPECT '*USER* 'AFTER)
      INST * <== (PREPOS 'LOCREL 'POBJ 'AFT)
      OBJ * <== (EXPECT 'ENTITY 'AFT))
```

It will be tokenized as follows in the WM:

```
(3.7)
(RETRIEVE ACTOR ACTOR11 TO TO12 INST INST13 OBJ OBJ14)
```

When a filler is indicated by a symbol "\*" the parser will make up a name based on its preceding slot (e.g., the slot name ACTOR plus a sequentially incrementing number 11) and the demon followed by an arrow ("<==") is spawned under this semantic node. When a conceptualization which satisfies the expectation of the demon is found during demon interpretation,

it will be bound to the tokenized filler. The tokenization of a conceptualization provides a convenient way to change and bind a structure simply. A lot of structure modifications will take place during the inferential analysis after parsing.

3.3.3.2 Demon tasks. The algorithm of the demon tasks is described as follows:

- STEP1      Traverse a node from the last node in the WM.
- STEP2      Get all active demons under the current node and go to STEP3.
- STEP3      Take one demon from the demon agenda and go to STEP4.
- STEP4      If KILL part is true, kill the demon. If TEST part is true then execute +ACT part and then kill the demon. If TEST part is NIL and there is a -ACT part then execute -ACT part and then kill the demon. Otherwise, the demon remains in the demon agenda. Go to STEP5.
- STEP5      If no more demons remain in the demon agenda go to STEP6, otherwise go to STEP3.
- STEP6      If any node to traverse remains, traverse the preceding node and then go to STEP2. Otherwise, go to STEP7.
- STEP7      If demon interpretation becomes quiescent (i.e., there is no firing of any demon during latest STEP1 through STEP6) stop the demon tasks, otherwise go to STEP1.

To spawn a demon is to add an instantiation of the demon to the demon agenda under the semantic node in the WM. The procedure to spawn a demon occurs in one of the following three cases: first, when there are any expectation demons in the conceptualization of the word in a sentence, the demons

are spawned during the word tasks; second, when there are any explicit demon definitions in the lexicon entry, it will be spawned during the word tasks; and finally, demons can be spawned dynamically in the course of executing the action part of a demon during the demon tasks. For example, the instantiation (e.g., DEMON16) of the demon "(EXPECT 'ENTITY 'AFT)" in the conceptualization (3.6) has the following form:

(3.8) (EXPECT CON15 OBJ14 ENTITY AFT)

where CON15 is the semantic node holding the demon agenda and OBJ14 is the filler to be filled by means of the demon. The two arguments (i.e., CON15 and OBJ14) are added to the expression in the demon of the conceptualization. The instance of the demon then knows the semantic node where it is located and the other semantic node where it will be bound. The demon EXPECT expects an entity, i.e., the parser will search the WM for a semantic node with the concept pattern of an entity class in the following direction. The spawning of the above demon is performed during the word tasks and the search will be executed during the demon tasks.

Killing a demon is to remove the demon from the demon agenda. A demon is killed either when the action part of the demon is performed or when the KILL part of the demon is tested as true. Each demon definition has an optional KILL part. Thus, a demon is responsible for its own life. A demon with a KILL part is mainly used to disambiguate word senses.

For instance, a word has three word senses such as M1, M2, M3 and three demons D1, D2, D3 where D1, D2, D3 are defined to disambiguate the word sense M1, M2, M3 respectively. Each demon must have its KILL part which tests whether the semantic node is bound to any sense. When the parser encounters the word, the semantic node is set to NIL, and the demons D1, D2, and D3 will be spawned. If the demon D1 is fired so that the word sense M1 is bound to the node, the demon D1 will be killed immediately since the action part is performed. The demons D2 and D3 will be killed since their own KILL parts are evaluated as true. Therefore, a demon communicates indirectly with other demons through a shared argument.

#### 3.3.4 Noun Group Parsing

The parser described up to now has difficulty in processing a noun group (NG) correctly for more than one noun. The empty filler holding an entity expectation demon might be bound to the incorrect conceptualization. For example, consider a sentence "Show me sleep stage time." The demon of the OBJ slot of "show" conceptualization in (3.6) expects an entity node. The OBJ slot demon searches the WM for an entity class node. It will bind the conceptualization of "sleep" prematurely rather than that of "time."

In this model, in order to prevent the parser from building an incorrect structure, a whole noun group is processed in a different way. When the parser gets into a noun



group, it adds each node for word/phrase in the WM and spawns associated demons without running demon tasks until the noun group ends. When the parser arrives at the end of the noun group, demon tasks will be executed within the noun group pool until it becomes quiescent. At this point, the noun group diagnostic function checks whether there is only one unbound top node which corresponds to a noun group head. Then, it will go back to the normal processing mode. For example, in the following sentence "Show me sleep stage 1 time." the parser will go to the noun group mode when it looks at the phrase "sleep stage." The lexical item for a phrase "sleep stage" has the following entry in the lexicon:

- 1) the syntactic category is noun,
- 2) the conceptualization with the demon of a sleep stage number expectation is (STAGE NUMB \* <== (EXPECT 'NUMBER 'AFT))
- 3) there are two demons:
  - DEMON1: If it is in the NG mode, and the NG end is not a number (e.g., "sleep stage 1 time"), then attach the modifier REL to the conceptualization to the NG end.
  - DEMON2: If it is in the NG mode and either it is the NG end or the NG end is a number (e.g., "sleep stage 1" or "sleep stage"), then do nothing.

and at the end of the NG processing the NG structure will have the following form:

```
(3.9) (TIME REL (STAGE VAL (NUMBER VAL (1))))
```

where the NG head is "time" and the phrase "sleep stage 1" modifies the NG head.

In this model, a NG is processed to decide the boundary and the head of the NG based on Gershman's work (1979). A NG constituent belongs to one of the following syntactic classes: adjective, adverb, noun, name, number, determiner, date, pronoun, unit. The NG is processed from left to right as long as the following conditions are satisfied:

- (1) Each word/phrase which is not specifically expected must belong to one of the classes mentioned above.
- (2) No word can precede a determiner.
- (3) Adjective and adverb cannot be preceded by noun, name pronoun, or number.
- (4) Name and pronoun cannot be preceded by noun, unit, or number.

### 3.4 Inferential Analysis

The inferential process involves transformation of a meaning representation into an adequate one before performing the query translation module in Section 3.5. Four types of domain-specific inferential rules are used in this model: specification, omission, concretion, and substitution inferences.

#### 3.4.1 Specification Inferences

Domain-specific knowledge is crucial to interpret NL questions correctly. For instance, the following ambiguous questions can be given in a sleep database dialogue:

- (3.10) Show me all subjects from 30 to 40.
- (3.11) Show me all subjects from 1988 to 1989.
- (3.12) show me all subjects from 34598 to 34600.

A sleep clinician would infer that (3.10) refers to subject ages, (3.11) refers to recording dates, and (3.12) refers to subject numbers. A domain-specific inferential rule will be applied to the phrase containing two numbers.

The following requests contain an implicit numerical comparison for a sleep parameter.

- (3.13) Show me subjects who have K-complex.
- (3.14) Show me epochs with K-complex in Jones's record.

where the sleep parameter "K-complex" means "K-complex greater than 0." An inference rule adds a numerical condition explicitly if a parameter has no numeric specification when it is said with a possessive type of verbs (e.g., have, contain, include) or prepositions (e.g., with, of).

#### 3.4.2 Omission Inferences

Natural language requests are often entered with redundant information. The redundant information should be removed since it is not necessary for database mapping processing. Consider the following requests:

- (3.15) Show me all subjects in the database.
- (3.16) Show me all subjects you have.
- (3.17) What is sleep stage time in the record of Jones.
- (3.18) What is sleep stage time from the Jones's record.

The phrase "in the database" in (3.15) and the phrase "you have" in (3.16) are redundant expressions and will be removed.

The phrase "the record of Jones" in (3.17) and the phrase "the Jones's record" in (3.18) will be reduced to "Jones."

A quantity request often includes duplicate words as in the following queries:

- (3.19) Show me total recording time of Jones.
- (3.20) How long is the total time in bed for Jones?

A word "total" is verbose in (3.19) and (3.20) since the concept "recording time" or "time in bed" is an aggregate attribute in the database. The concept itself contains the quantity information implicitly. Hence, an omission inference rules will remove the verbose concept "total."

Since these inference rules are applied to the intermediate meaning representation of a sentence, one rule can be applied to cover a wide range of syntactical expressions. For example, one rule is applicable to two cases (3.17) and (3.18).

#### 3.4.3 Concretion Inferences

Some concepts are so vague or ambiguous that it is very difficult to be specific at the parsing time. The actual content of a concept can be determined by these inferential rules. For example, a concept "time" can have several different meanings in a sleep database dialogue:

- (3.21) Show me time when K-complex occurred.
- (3.22) Show me first time of sleep stage 2.
- (3.23) Show me time of each sleep stage.
- (3.24) Show me sleep stage 2 time.

- (3.25) Show me total minute of alpha time.  
(3.26) When did K-complex occur?  
(3.27) When was Jones recorded?

where (3.21), (3.25) and (3.26) refer to "epoch number," (3.22) refers to "sleep latency," (3.23) and (3.24) refer to "sleep stage time," and (3.27) refers to "recording date." The above underlined words are defined as the concept "time" in the lexicon and then will be inferred to be a specific concept based on contextual information. The more specific concepts will be selected based on the contextual information from the ISA-hierarchy of the domain concepts in SEEGER.

The same vague concepts also exist in verbs such as "take" or "use." The following requests show the same problem about a concept "take" in the sleep domain:

- (3.28) Who was taken on 2/5/89?  
(3.29) Who took REM sleep more than 20%?

A sleep clinician would easily infer that the word "take" means to "record" in (3.28) and "possess" in (3.29). In a domain-specific approach words have very restricted meaning in the context of specific application.

When the meaning of words is difficult to decide on the sentence level by the conceptual parser, the concretion inference rules can operate on the hierarchy to determine the more specific meaning of the word.

#### 3.4.4 Substitution Inferences

A concept can be often expressed in a set of concepts with the same meaning. In addition, a set of concepts can be expressed in another set of concepts which is in the canonical form with the same meaning. For instance, consider the following request in a sleep database dialogue:

- (3.30) Show me alpha data.
- (3.31) Show me information about alpha.

where "alpha data" or "information about alpha" means to be either "alpha time" or "alpha count" since its value is stored in the sleep database as an attribute name of either "alpha time" or "alpha count". The substitution inference rules will replace all the identical meaning phrases (e.g., alpha data, alpha activity, alpha information) with "alpha." Then, the concept "alpha" will be clarified as either "alpha time" or "alpha count" by the user in a clarification dialogue.

Some phrases are replaced with more clearly expressed phrases as in the following example:

- (3.32) Show me total arousal time.
- (3.33) Show me total time of arousal.

A phrase "arousal time" or "time of arousal" will be inferred as "epoch number where any arousal occurred" which is in the canonical form with the same meaning. Since this inferential process is performed on a CD representation of a sentence.

### 3.5 Query Translations

Although other KB systems such as EXPLORER and EASYTALK obtain the retrieval query of the underlying database via the DB mapping rules, the proposed model gets the retrieval query by means of a new method used below. When the types of NL request rely on the bottom-up parsing more than top-down parsing, the number of mapping rules are so enormous that it is difficult to manage them. The augmentation of query translation also provides various inferences at the level of the linear expression rather than at the level of CD representation.

#### 3.5.1 Translating to Linear Forms

Since there is a large gap between the meaning structure of NL input and its retrieval query, an intermediate stage of transformation is used in this model. This transformation process is called "translation." A CD representation produced by the conceptual parser is in the form of tree structure and is transformed into a set of finite linear specifications called "virtual query language" (VQL). This process involves grouping the same kind of features into one place called a "field." The VQL is expressed in the six fields which can be easily converted into a target database language. The database operations can be expressed in VQL but do not include all features for every target database language. Rather, VQL contains a set of core features rich enough to cover most of

the low- and medium-complexity queries and ignores infrequently used complex database operations.

In this model, six fields for translation are used as follows:

OUTPUT: a list of concepts to be retrieved

CONDITION: a list of predicate expressions in logical relationship

FUNCTION: a list of expressions in functional relationship

GROUP: a list of concepts to be grouped

ORDER: a list of concepts to be sorted

RELATION: a list of logical relationships between the expressions given in the field

The tree structure representation of a sentence is processed by the translator that applies translation functions to each node from the top concept. For instance, consider the following sentence:

(3.34) Show me the sleep stage 1 time.

The conceptual parser produces the following meaning representation:

```
(3.35) (RETRIEVE ACTOR (NIL)
        TO (*USER*)
        INST (NIL)
        OBJ (TIME REL (STAGE VAL (NUMBER VAL (1))))))
```

The inferential analysis concretes the concept "time" to the another concept "stage time" as follows:



```
(3.36)
(RETRIEVE ACTOR (NIL)
  TO (*USER*)
  INST (NIL)
  OBJ (STAGE-TIME REL (STAGE VAL (NUMBER VAL (1))))))
```

The form in (3.36) is translated into the following form:

```
(3.37) (VQL (OUTPUT (STAGE-TIME))
  (CONDITION ((STAGE = 1))))
```

The meaning representation in (3.36) has the top concept RETRIEVE and it is translated by the function for the RETRIEVE pattern. The function processes the expression in (3.36) in the order of the ACTOR, TO, INST, and OBJ slot-filler pair. The first three slot-pairs are neglected because they do not provide any information for query formulation. Since the OBJ slot is filled with an ENTITY class object, it will be processed by the translation function for the ENTITY pattern. The function inserts the head pattern STAGE-TIME into the OUTPUT field and (STAGE = 1) in the CONDITION field by calling the ENTITY translation function again to interpret the expression (STAGE VAL (NUMBER VAL (1))). The label REL in (3.36) is a marker to switch the field setting from the OUTPUT field to the CONDITION field.

Let's consider another example. For a sentence "Show me the sleep stage 1 and 2 time." the meaning representation is in the following form:

```

(3.38)
(CONJ CON1 (RETRIEVE ACTOR (NIL)
           TO (*USER*)
           INST (NIL)
           OBJ (STAGE-TIME REL
                (STAGE VAL (NUMBER VAL (1))))))
      CON2 (RETRIEVE ACTOR (NIL)
           TO (*USER*)
           INST (NIL)
           OBJ (STAGE-TIME REL
                (STAGE VAL (NUMBER VAL (2))))))

```

This is translated into two sets of VQL as follows:

```

(3.39) (VQL1 (OUTPUT (STAGE-TIME))
        (CONDITION ((STAGE = 1))))

        (VQL2 (OUTPUT (STAGE-TIME))
        (CONDITION ((STAGE = 2))))

```

The coordinate structure generates more than one VQL. They are combined into one final VQL. It will be as follows:

```

(3.40) (VQL (OUTPUT (STAGE-TIME))
        (CONDITION ((STAGE = 1) (STAGE = 2)))
        (RELATION ((AND CONDITION STAGE))))

```

where the RELATION field informs that two elements of the CONDITION field are in the coordinated relationship.

### 3.5.2 Augmenting the Translation

The translation output so far is not so complete as to be mapped into a target database language. It is augmented to form a complete set of VQL. Some patterns in VQL cannot be directly transformed into the target database language since there are no matching attribute names, attribute values, or database key words. It is replaced with other patterns that

can be transformed. The replacement can be done by applying contextual knowledge in a domain. Four types of augmentation are used in this model:

- 1) quantifier replacement
- 2) default unit filling and input value scaling
- 3) question type transformation
- 4) validity check

3.5.2.1 Quantifier replacement. It replaces a quantifier with its equivalent specifications in terms of the domain concepts. For example, a sentence "Show me sleep stage 1 time for all subjects." has the universal quantifier "all." The VQL will be translated in the following form:

```
(3.41) (VQL (OUTPUT (STAGE-TIME))
          (CONDITION ((STAGE = 1)))
          (FUNCTION ((QUANTIFIER SUBJECT ALL))))
```

While the quantifier in the FUNCTION field is removed, all record numbers in the sleep database will be inserted in the CONDITION field and their relationship is specified in the RELATION field as follows:

```
(3.42)
(VQL (OUTPUT (STAGE-TIME))
      (CONDITION ((STAGE = 1) (REC-NO = 34989)
                  (REC-NO = 35998) (REC-NO = 38797) ...))
      (RELATION ((DISJ COND REC-NO))))
```

If a universal quantifier such as "each, all, or every" appears in the query, the expression for the quantifier is replaced with an appropriate expression in terms of the domain concepts.

### 3.5.2.2 Default unit filling and input value scaling.

A user often tends to leave out specifying a unit for a value. He sometimes specifies a unit which is different from the unit given in the database. For example, in a sentence

(3.43) Show me epochs for delta time is greater than 15.

the value "15" means "15 second" as a default unit. Thus, the expression "(DELTA-T > 15 NIL)" in the CONDITION field is converted into "(DELTA-T > 15 SECOND)" by filling in a default unit. Consider another example

(3.44) Show me subjects whose total REM sleep is more than 2 hours.

the value of "total REM sleep" has the unit in minutes in the database. The expression "(SLEEP-TIME > 2 HOUR)" in the CONDITION field is scaled to "(SLEEP-TIME > 120 MINUTE)." Hence, a default unit is specified if it is missing or adjust the scale in order to make the same unit as given in the database.

3.5.2.3 Question type transformation. NL query systems have a single main goal of retrieving information from the underlying database. If a user asks yes/no questions, the question is transformed into an appropriate form to access the database because users are considered to be interested in the data rather than the answer "yes" or "no." For yes/no

questions there are no elements in the OUTPUT field. For example, a yes/no question "Is Kimberley 20 years old?" has the following translation:

```
(3.45)
(VQL (OUTPUT ()))
  (CONDITION ((LASTNAME = KIMBERLEY) (AGE = 20)))
```

This translation needs to be augmented as follows:

```
(3.46)
(VQL (OUTPUT (LASTNAME AGE)))
  (CONDITION ((LASTNAME = KIMBERLEY) (AGE = 20)))
```

That is, the system will access information about "name" and "age" for the given conditions rather than just give an answer "yes" or "no." Hence, a yes/no question is transformed into a question to access the database.

3.5.2.4 Validity check. There may be NL questions for which no corresponding target query language exists. Thus, a VQL is validated before it is mapped into a target DB language. Invalid VQLs are largely due to the user's misconceptions about the domain or the capabilities of the system. The system should remind the user of the limits of the system's knowledge. There are several types of invalid NL input:

- 1) A variety of user goals: NL query systems usually assume that user's requests are limited to the database content. But the following requests can be asked:

- (3.47) What do you know?
- (3.48) What kind of data do you know?
- (3.49) What does sleep time mean?

The NL query systems should have the capabilities to recognize various user goals in order to understand the requests beyond the database content.

2) Numerical quantity of records: If a target database is relational, it cannot handle the number of rows in the records. For example, in the following request,

- (3.50) Show me five subjects who were recorded on Jan. 1988.

only five rows of the records must be given but they cannot be mapped into a relational database language.

3) Multiple DB queries: Some user's requests require the answer to be given in the multiple DB queries due to the DB structure. For example, the following requests,

- (3.51) Show me sleep efficiency and total sleep stage time.

should be divided into two individual queries since "total sleep stage time" is given for each sleep stage, but "sleep efficiency" is a single value regardless of the sleep stage. If the system does not have capability to handle multiple queries it must inform the user of this lack.

4) Problem deduction capability: Although the required information is contained in the database, NL requests require problem deduction capabilities to provide its answer. Consider the following requests:

- (3.52) How many times did Kimberley wake up on the night of 2/5/88?
- (3.53) Give me the longest blocks of sleep stage 2 for record 34689.

In the sleep database the requests cannot be answered directly in a single query and must be deduced to simpler multiple queries and/or the retrieval code. Both requests need the retrieval code to recognize the segmentation of the sleep data.

The items described here are illustration of invalid requests when the facilities are not implemented. It requires a considerable amount of knowledge to recognize a variety of invalid input.

### 3.6 Database Mapping

DB mapping in this model is performed by converting domain concepts in VQL into actual attribute names and attribute values while storing access-path information related to the database search. The access-path information is used to find out involved database files, create joins, and generate an optimal navigation path. The conversion from the

domain concepts to the actual DB fields is specified in the domain-to-database mapping table. The mapping table contains the information about the physical DB file names for access and DB attribute names for the corresponding domain concept.

The expression in VQL is independent of the physical structure of the underlying database. DB mapping procedures convert the domain concepts in VQL into the target DB language. If, for example, the target language is SQL, then the VQL expression is transformed into the SQL query. Hence, the mapping procedures should be designed to generate the appropriate target DB language. For example, consider the following sentence in the sleep database

(3.54) Show me Kimberley's total number of arousals for each sleep stage.

The VQL is given as

```
(3.55)
(VQL (OUTPUT (AROUSAL))
      (CONDITION ((L-NAME = KIMBERLEY))
              (FUNCTION ((SUM AROUSAL) (QUANTIFIER STAGE EACH)))))
```

and is augmented for the sleep stage with the universal quantifier as

```
(3.56)
(VQL (OUTPUT (AROUSAL))
      (CONDITION ((L-NAME = KIMBERLEY) (STAGE = 0)
                  (STAGE = 1) (STAGE = 2) (STAGE = 3)
                  (STAGE = 4) (STAGE = 5)))
      (FUNCTION ((SUM AROUSAL)))
      (RELATION ((DISJ COND STAGE))))
```



The DB mapping procedures traverse the fields of the VQL in (3.56) and convert the domain concepts into the target DB expression via the mapping table. The mapping table for the above concepts is given as

```
(3.57) AROUSAL --> ((EP_SUM) (AROUSAL EPOCH_NO))
      L-NAME   --> ((SUBJ_INFO) (LASTNAME))
      STAGE    --> ((EP_SUM STG_VAR) (SAC_STG EPOCH_NO))
```

and the join table between the DB files is given as their primary attributes such as

```
(3.58) SUBJ_INFO --> SUBJ_INFO.REC_NO
      EP_SUM      --> EP_SUM.REC_NO
      STG_VAR     --> STG_VAR.REC_NO
```

The target DB language is generated by the DB mapping procedures in which the mapping table and the join table are involved. Finally, the SQL language is given as

```
(3.59)
SELECT EP_SUM.REC_NO, SUM(AROUSAL), SAC_STG, LASTNAME
FROM SUBJ_INFO, EP_SUM
WHERE (LASTNAME="KIMBERLEY" AND SAC_STG = 0 OR
      LASTNAME="KIMBERLEY AND SAC_STG = 1 OR
      LASTNAME="KIMBERLEY AND SAC_STG = 2 OR
      LASTNAME="KIMBERLEY AND SAC_STG = 3 OR
      LASTNAME="KIMBERLEY AND SAC_STG = 4 OR
      LASTNAME="KIMBERLEY AND SAC_STG = 5) AND
      EP_SUM.REC_NO = SUBJ_REC.REC_NO
GROUP BY SAC_STG, LASTNAME, EP_SUM.REC_NO,
```

where the two DB files are joined over the attribute REC\_NO.

Finally, the generated DB language is executed in the underlying database management system.

### 3.7 User-Friendly Design

Spelling corrections in this model are based on the method proposed by Carbonell (1985). This capability is a by-product of the conceptual parsing. This model also handles clarification dialogues for clarifying ambiguous terms and semantically incomplete requests. This section also describes how to paraphrase requests and when the system failure occurs in the model.

#### 3.7.1 Spelling Corrections

In this model, when the conceptual parser encounters an unknown word during parsing NL input, the Spelling Corrector first tries to find the most likely correct word from the system dictionary. About 40% of all user errors are known as misspellings (Carbonell, 1985). Most misspellings are caused by one of the following reasons (Damerau, 1964):

- 1) transposition of two adjacent letters
- 2) one letter wrong
- 3) one letter extra
- 4) one letter missing

These context-free spelling rules generate a list of possible candidate words. For example, in the following sentence

(3.60) Show me subjects recorded on sel 1987.

the word "sel" is a typographical error and its candidates given by the above rules are "sex, sem, see, sec, and sep" from the lexicon of the sleep NL DB access system.

After collecting a list of candidates, a context-sensitive spelling check is performed by means of expectations (Carbonell, 1984). It is done by testing active expectation demons to determine whether there is any one whose test part is satisfied. That is, all active demons in the Working Memory are tested in a "recency order" from the candidate list. If there is one to be fired, it is considered as a plausible word. In (3.60) the word "record" has the following tokenized definition:

(3.61) (RECORD ACTOR ACTOR1 TO TO2 CHANNEL CHANNEL3  
PLACE PLACE4 DATE DATE5 INST INST6)

and the demons are simply described as

(3.62)  
 ACTOR1: expect to have heard about the class TECHNICIAN entities before  
 TO2: expect to hear about the class SUBJECT entities next  
 CHANNEL3: expect to hear about the class WAVEFORM or CHAN-NAME next  
 PLACE4: expect to hear about the class PLACE or CHAN-PLACE along with a locational preposition next  
 DATE5: expect to hear about the class NUMBER or DATE along with a temporal preposition next  
 INST6: expect to hear about the class ELECTRODE along with a configurational preposition next

The candidate words will be tested one by one for the pending expectation demons in (3.62). The word "sep" (abbreviation of September) belongs to the class DATE so it will be bound in

the filler DATE5. Therefore, the word is considered as a possible correction and suggested to the user for approval as follows:

(3.63) By SEL, do you mean by SEP? (Y or N)

If the user response as "yes," the misspelling word will be substituted with a possible candidate. Otherwise, the Spelling Corrector gives the user an opportunity to enter a synonym word/phrase or a correct word.

### 3.7.2 Clarification Dialogue

Many requests, especially those generated by casual users, are likely to require an interactive dialogue. Two types of clarification dialogues are used in this model: 1) clarifying terminology and 2) completing a query semantically.

3.7.2.1 Clarifying terminology. It is necessary to recognize and respond to any ambiguous or unknown terms present in a request. Many systems have tried to resolve these problems in several ways.

The LADDER system allowed the user to define a new term or phrase with one that the system knows. It assumes that the user already has enough knowledge about the system.

RENDEZVOUS (Codd, 1978) attempted to extract the meaning from the user by means of a dialogue such as

(3.64)

USER: How many London parts are there?

RENDEZVOUS: The word 'LONDON' is unfamiliar. Is it one of the follows:

- |                |                      |
|----------------|----------------------|
| 1. Part Number | 4. Part Weight       |
| 2. Part Name   | 5. Part City         |
| 3. Part Color  | 6. None of the above |

The system has a list of key words. When the user enters unknown words which are not in the key word list, it goes to a clarification dialogue.

TEAM acquires the DB information and syntactic knowledge through step-by-step interaction. This defining process can be done with a database expert rather than end-users.

When there is a unknown word in a request, INTELLECT engages in the following dialogue (Martin, 1985):

(3.65)

USER: What are the customer in poughkeepsie?

INTELLECT: I'm not familiar with the word "POUGHKEEPSIE." If it is a word you expect to find in the database, hit the RETURN key. Otherwise either fix its spelling or enter a synonym for it.

USER: (press the ENTER key)

INTELLECT: What field should it appear in?

USER: city

As shown in (3.65), INTELLECT assumes that the unknown term is a database value and waits until the user enters a synonym, corrects its spelling, or verifies it through a paraphrase.

As described above, the users in the illustrated systems are required to have prior knowledge about the system or about the physical database structure.

In this model, a user is guided to select one of several choices or just answer with "yes" or "no." For example, consider the following dialogue:

(3.66)

USER: Please show me alpha data.

COMPUTER: By ALPHA, do you mean

- 1) ALPHA TIME
- 2) ALPHA COUNT

Please choose one:

where the system is waiting for the user's response. The system first tries to resolve ambiguous terms in the contextual information by means of the conceptual parsing and the inferential analysis. Then, the meaning structure or its VQL is searched for any ambiguous concepts. Every concept is tested as to whether it matches with the concept in the table which contains all ambiguous concepts in the domain. The concept "alpha data" in (3.66) is ambiguous in the sleep DB access system and must be clarified as one of two concepts. The table informs the system how to clarify it. The ambiguous term will then be replaced with one the user selects. Therefore, this interactive dialogue requires domain-specific knowledge to accommodate naive users.

3.7.2.2 Completing a query semantically. Naive or infrequent users are also likely to miss information that is

necessary to formulate a complete query. The "incomplete" query means that the sentence's structure is complete syntactically but not complete semantically. It is different from the structural ellipsis which constitutes a fragmentary sentence.

PLANES fills in missing constituents if there is any missing one in a semantic grammar. For example, in the following request,

(3.67) What maintenances were performed on the plane 3  
in May 1971?

its semantic constituent is in the form of

(3.68) [QTYPE MAINTYPE MAINACTION PLANETYPE TIMEPERIOD]

The sentence in (3.67) matches all the semantic constituent types in (3.68) as a complete query. But if the following one is given as

(3.69) What maintenances were performed on plane 48?

and the TIMEPERIOD constituent has been missed from the semantic constituent in (3.68). Since it cannot be completely matched with the required semantic constituent, the missing "time period" could be filled in from its discourse context.

EXPLORER (Lehnert & Shwartz, 1983) interprets requests for oil exploration maps. The scriptal knowledge (Schank & Abelson, 1977) about generating maps guides the user through an interactive dialogue in order to specify all requirements

for a complete query. Like PLANES, unspecified information can be drawn from the discourse context by inheriting specifications from the previous request. This type of interaction is critical to untrained users.

In this model, users are guided in the same manner as above. The scriptal knowledge is used to examine the VQL for any missing information to form a complete query. For example, the system interacts with the user with the following response after the disambiguation dialogue in (3.66):

(3.70) Is the subject Jones? (Y or N)

The user is queried to specify one of the subjects in the database.

### 3.7.3 Paraphrasing of Requests

Paraphrasing presents the system's interpretation of NL questions to the user for approval. It should be phrased in a NL string without introducing any ambiguities, and its meaning should be easily recognized by a user.

The different ways of paraphrase in the various systems are described in Finin, Joshi, and Webber (1986). The QTRANS system (Mueckstein, 1983) produces a paraphrase from the SQL language before it is evaluated against the database. It rearranges the SQL expression into another structure without being too repetitive or introducing ambiguities.



In this model the paraphrase utilizes both virtual query language (VQL) and SQL expression selectively. Most of SQL expressions are translated except the WHERE part. Instead of the WHERE part in SQL the CONDITION field in VQL is used. For example, for the following request,

(3.71) Show me Kimberley's total number of arousals for each sleep stage.

the VQL is given as

```
(3.72)
(VQL (OUTPUT (AROUSAL))
      (CONDITION ((LASTNAME = KIMBERLEY) (STAGE = 0)
                  (STAGE = 1) (STAGE = 2) (STAGE = 3)
                  (STAGE = 4) (STAGE = 5)))
      (FUNCTION ((SUM AROUSAL)))
      (RELATION ((DISJ COND STAGE))))
```

and its SQL language is mapped as

```
(3.73)
SELECT EP_SUM.REC_NO, SUM(AROUSAL), SAC_STG, LASTNAME
FROM SUBJ_INFO, EP_SUM
WHERE (LASTNAME="KIMBERLEY" AND SAC_STG=0 OR
       LASTNAME="KIMBERLEY" AND SAC_STG=1 OR
       LASTNAME="KIMBERLEY" AND SAC_STG=2 OR
       LASTNAME="KIMBERLEY" AND SAC_STG=3 OR
       LASTNAME="KIMBERLEY" AND SAC_STG=4 OR
       LASTNAME="KIMBERLEY" AND SAC_STG=5) AND
       EP_SUM.REC=SUBJ_REC.REC_NO
GROUP BY SAC_STG, LASTNAME, EP_SUM.REC_NO.
```

The NL string in the paraphrase is phrased in the order of the output elements from the VQL, the condition parts from the SQL, and sorting information from the SQL. It will be given as

(3.74)

I understand your request to be:

"Find record number, total arousals, sleep stage, and last name with sleep stage = 0, 1, 2, 3, 4, or 5 for last name KIMBERLEY."

Is it right? (Y or N)

where the NL string is stored in the DB mapping table.

After the user has approved the paraphrase, there is little chance that interpretation errors will be introduced later.

#### 3.7.4 Error Handling

An error occurs when the system's interpretation of a NL input diverges from the user's interpretation. It is important to recognize and respond to the failures in a system. If the system is able to respond to what constituent can be interpreted or what constituent cannot be interpreted, it can help the user to rephrase a request.

In this model, when the system fails in reaching an interpretation of the input and is unable to proceed, it assumes that an error has occurred. In the course of transforming NL input into VQL and then into DB language, errors may occur if there exists no corresponding representation for the following stage. How to handle errors at various situations in this model is described below.

3.7.4.1 Lack of vocabulary. No natural language system can be expected to have a complete vocabulary. When there is

no correctly spelled word in the lexicon in this model, the system initiates an interactive dialogue to acquire a synonym. For example, consider the following dialogue:

(3.75)

USER: Give me total amount of arousals for Jones.

SYSTEM: I do not understand the word AMOUNT.

Would you like to

- (1) Enter a correct spelling
- (2) Propose a synonym
- (3) Have me ignore the word
- (4) Re-enter the entire request

Please choose one:

There are four optional selections (Shwartz, 1987) when there is no word "amount" in the lexicon. It is self-explanatory what each selection is meant to be. If the user enters a synonym, the system can capture the synonym of the unknown word and store it in the lexicon for later use. Even though the system has failed due to lack of vocabulary, user's requests can be interpreted correctly through the above dialogue.

3.7.4.2 Failure to parse a noun group. A NG phrase should be represented in one structure which contains its whole meaning at the end of NG processing. In this model, if there is more than one structure or none, it is considered that a NG parsing error occurred. Noun group processing is performed within a NG pool. When the NG pool arrives at the quiescent point, the NG diagnostic function is called in order to count the number of independent structures in the NG pool.

If there is more than one or none, a NG parsing error occurs. This error is due to the missing or wrong piece of knowledge in the lexical items if it is assumed that the user entered a correct request. At this time the system responds to the user that the system is unable to interpret the constituent of the NG phrase.

3.7.4.3 Failure to parse an entire sentence. Similar to NG parsing, if there is more than one structure or none, a conceptual parsing error occurs in this model. After analyzing a whole sentence, the diagnostic function is called to count the number of independent structures of the entire sentence. For example, consider the following request which is missing a word "on" in front of the date expression:

(3.76) USER: Who was recorded 2/5/88?

The meaning representation has the following form:

(3.77) (RECORD ACTOR (NIL)  
           TO (SUBJECT VAL (\*\*))  
           CHANNEL (NIL)  
           PLACE (NIL)  
           DATE (NIL))

(DATE MONTH (5) DAY (2) YEAR (88))

There are two independent structures whose top concepts are RECORD and DATE in (3.77). In the primitive action RECORD, the demon for the slot DATE expects to hear about the class DATE along with a temporal preposition (e.g., on, in, etc.). Thus, the demon remains active in the WM so that the concept DATE

remains in the WM without being linked to any other concept. Finally, The system responds with the constituent that the parser partially interpreted as follows:

(3.78)

SYSTEM: I cannot understand the entire request.

The constituent that I could understand is 'WHO WAS RECORDED' and '2/5/88' of your request. Please try to rephrase it.

3.7.4.4 Failure to translate into a VQL. In this model, an error occurs when there are meaning representations for which no corresponding VQL exists. A meaning representation is translated into a VQL by translation functions. Then, the VQL is augmented as described in Section 3.4. Errors occur during validity check of VQL augmentation. For example, consider the following request:

(3.79) Show me 5 subjects who were recorded on 2/5/88.

If the target DB language is a relational DB, it cannot handle the number of rows in the records. It will inform the user why it cannot work. In this way, errors is explained to users for each situation during the VQL validity check. The message should be understandable to the user.

3.7.4.5 Failure to map into DB language. In this model, an error occurs when there is no VQL for which corresponding DB language exists. Many errors due to the DB structure or DB language are recognized during VQL validity check. An error occurs when there are any concepts which cannot be mapped into

target database attributes or key words. It is checked to see whether a mapped DB language expression is valid or not before executed the underlying DB management system.

### 3.8 Summary

This chapter described a model for a knowledge-based NL query system. In this model, the user's request is transformed into retrieval query via multiple transformation steps. The input sentence is mapped into a Conceptual Dependency representation of its meaning by the Conceptual Parser and then is explicitly expressed by the Inferential Analyzer if the CD representation contains implicit information. The Translator converts the CD representation into a linear expression which is then augmented to a complete form for query formation. Finally, it is mapped into the target DB language of the underlying database. The model also provides an interactive dialogue facility for spelling corrections, clarifying ambiguous concepts or semantically incomplete requests, and paraphrasing user's requests.

This model has some differences from other knowledge-based DB access systems using the conceptual parser. The DB mapping method in this model is different from EXPLORER, EASYTALK, and PEARL (Lehnert & Shwartz, 1982) which are the only systems known in the literature. These systems transform a CD representation of an input sentence directly into a QL via the DB mapping rules to determine what fields and/or

aggregates of fields in the database should be displayed, constrained, and sorted. These rules also contain information to create joins between DB files and to generate an optimal navigation path. This model transforms a CD representation of an input request into a VQL expression as an intermediate stage and then into the target DB language. If the types of NL requests in any application domain are too diverse, the questions are parsed in a bottom-up manner more often than in a top-down manner. This frequently occurring bottom-up parsing produces the various types of meaning structure. To facilitate the transformation from the diverse meaning structures into the target DB expression, the number of the rules is so enormous a VQL expression should be used. The VQL expression is a set of linear specifications of IRL and provides a more convenient method to augment some DB mapping patterns.

There are some other systems that relate to this model but use the different parsing techniques. The EUFID system (Templeton & Burger, 1983) converts the parsing output into a linear string of tokens which are then mapped into the retrieval query similarly to this model. But EUFID used context-free grammars to get the parsing output, and it has no capability for augmenting the linear expression and interactive dialogues. The FRED system (Jakobson, Lafond, Nyberg, & Piatetsky-Shapiro, 1986) maps the case frame parsing output into a linear database query expression, and it does not have the extensive augmentation capability that this model has.

CHAPTER 4  
SEEGER: A SLEEP NATURAL LANGUAGE QUERY SYSTEM

In order to perform at the level of a data processing technician who provides the sleep data to a sleep clinician or researcher, at least the following three capabilities are required:

1. process requests for sleep data retrieval in a wide range of syntactic variations,
2. process requests for sleep data retrieval in a wide range of semantic complexities,
3. engage in an interactive dialogue with the user for ambiguous and incomplete requests.

No attempt has been made to build a sleep NL query system before. SEEGER, as the first sleep NL query system, has been designed and implemented for the purpose of achieving the above capabilities (Kim, Principe, & Smith, 1988).

The overall structure of SEEGER based on the new NL query model of Chapter 3 is discussed in Section 4.1. Section 4.2 describes a proposed sleep representational system to represent the sleep domain concepts. Section 4.3 contains a



proposed sleep database system. The elements to cover the syntactic variations, and semantic complexities for a sleep NL system are described and illustrated with the test requests in Section 4.4 and in Section 4.5. Some of the elements for the systematic evaluation of a NL system proposed by Tennant (1981) are modified for a general knowledge-based NL system. Section 4.6 discusses the implementation issues and system requirements.

#### 4.1 The Overall Structure of SEEGER

The model of Chapter 3 is implemented into SEEGER and shown in Figure 4.1. Since details were described in the previous chapter, some features in SEEGER will be discussed below.

##### 4.1.1 Conceptual Parser

SEEGER has a lexicon of 636 lexical entries (i.e., vocabulary) composed of 455 words and 181 phrases. There are a total of 84 demon definitions used in the lexical entries. Each lexical entry contains a syntactic category for noun group processing, word senses representing the meaning of the entry, demons which keep processing information of the entry, and a synonym. The domain-independent entries refer to constraints (e.g., greater than, less than), sorting and grouping (e.g., oldest, male), statistical terms (e.g., total, average), grammatical indicators (e.g., comma, question mark),

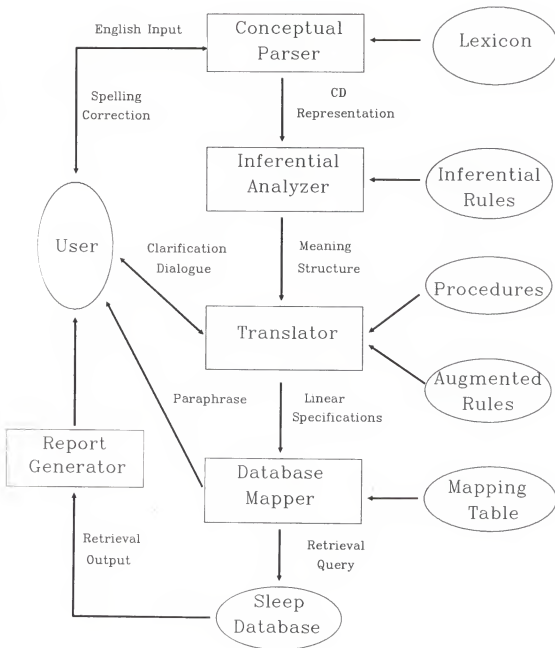


Figure 4.1 The Overall Structure of SEGER

etc. The sleep domain-specific entries refer to temporal references (e.g., recording date, Jan., 1988), unit (hour, minutes, second, %), subject and recording information (e.g., subject#, name, date, montage, EOG), sleep activities (e.g., alpha time), sleep stage (e.g., REM, NREM, stage time), sleep parameters (e.g. sleep time, latency), etc. The meaning definition is in a declarative form, and the demon is in a rule form.

The word senses in SEEGER are designed based on the "model corpus" (Cullingford, 1986) as a bottom-up approach. This is a collection of sentences or sentence fragments that sample as large a fragment as possible of the knowledge content of the domain. A good corpus is the key to the domain coverage. The corpus is annotated as the comment lines in the lexicon module in order to help the sleep NL system developer to design or modify the word meaning definitions.

#### 4.1.2 Inferential Analyzer

The control mechanism of the Inferential Analyzer applies a set of inferential rules to the CD representation of a sentence. There are 47 inferential rules in SEEGER in order to run several types of domain-specific inferences such as specifications, omissions, concretions, and substitutions described in Section 3.3. It specifies implicit numerical or possessive references explicitly; removes redundant references about quantity, possession, or data record; concretes the

temporal data at the various levels of detail; or transforms the fragmentary structure (e.g., relative clause) into the canonical form.

The rules are driven in order from the rule list that holds a set of rules. An inferential rule is defined in the following form:

```
(4.7) (RULE rule-name
        (ISA class-name)
        (PARAMS list-of-parameters)
        (IF lisp-expression)
        (THEN lisp-expression))
```

where "class-name" is used to classify the whole rules into several groups in order to deal with the relevant rules in a control loop. If there is any rule whose IF part is true, its THEN part is executed so the output of the parser is modified. A set of rules are applied repeatedly until it becomes quiescent (i.e., no rule is executed in a loop). Hence, the control mechanism is designed to be executed in the same way as the demon task control mechanism in Section 3.2.

#### 4.1.3 Translator

The meaning structure via the inferential analysis is transformed into a set of linear specifications which is a VQL. The transformation is performed by the set of translation functions. There are 21 translation functions in SEEGER. The meaning structure in the slot-filler format is decomposed by the functions designated in terms of the head patterns. The

translation functions insert the appropriate patterns into the fields while decomposing the meaning structure.

The augmented rules are applied to the VQL. There are 14 augmentation rules in SEEGER. The rules replace the universal quantifier for sleep stage and subject with its corresponding specifications, fill in the missing temporal unit, scale the numeric input to the system's time unit, transform a yes/no question type into a form to access data, or validate a query expression about the numerical quantity of records, multiple DB queries, or DB record segmentation. The rules are driven by a pattern matching mechanism. The patterns in the fields are tested to see whether there exist any matching patterns corresponding to the variables in a rule pattern. A unification algorithm (Charniak, Riesbeck, McDermott, & Meehan, 1987) is used to find the matching patterns. The patterns in the VQL are augmented as described in Section 3.4.

#### 4.1.4 Database Mapper

The augmented VQL is transformed into the retrieval query in the SQL format of dBASE IV as described in Section 3.5. The domain concepts in the fields are mapped into the corresponding DB expressions which are physical DB attributes, DB files, operators, functions, etc. There are 82 domain concepts at the mapping table in SEEGER. The concepts are relevant to subject record, channel recording, epoch summary, stage summary, and night summary. They also contain the

database access-path information of the physical DB files. The SQL expression is stored into a file for the evaluation in dBASE IV.

#### 4.1.5 Interactive Dialogues

There are 20 concepts to be clarified in SEEGER. They are stored in a form of table. As an epoch summary parameter, each concept can be either a time activity or a count activity. It is converted into an unambiguous concept, or it calls a function that allows the user to select one among the candidates. Seven clarification functions lead the user to select a concept of either "time" or "count" for alpha, beta, delta sigma, theta; either REMI (in-phase REM) or REMO (out-of-phase REM) for REM activity; high, medium, or low for EMG level.

To formulate a complete query, SEEGER tries to find whether the user's request includes the subject information such as subject name, record number, etc. If subject information is not specified in the request, SEEGER guides the user to select one of the subjects in the sleep database.

#### 4.1.6 Report Generator

The Report Generator holds the evaluated output from the dBASE IV database management system and shows it to the user one screen at a time. The SQL expression as a source code is compiled to generate an object code and then executed to store

the output in a file automatically without the user's interaction. It also enables the user to quit examining the output at any moment. The whole output can be sent to the printer by selecting the options.

#### 4.2 A Sleep Representational System

A representational system proposed for a sleep EEG domain is shown in Figure 4.2. Many terms used here are based on the manual by Rechtschaffen and Kales (Rechtschaffen & Kales, 1968). The taxonomy of the domain concepts is represented in a semantic network of an ISA-hierarchy. The ISA-hierarchy of a domain reflects the property of the concepts to be classified and the operations to be performed on them. The top concept of the domain is classified into four main categories: entities, actions, states, and relationships. These are subdivided into a number of their subordinate types. There are a total of 237 sleep domain concepts in the ISA-hierarchy. The organization of four main categories will be described.

##### 4.2.1 Entities

The entities or objects of the sleep domain are classified into three subordinate entities: animate entities, physical objects, and abstract objects.

The animate entities (e.g., the system, user, subject or patient, technician) engage in actions. The system and a user engage in a dialogue. For example, if a user enters a request

## Entity:

Animate entity: system, user, subject, technician

## Physical object:

person: subject, technician

man-made object:

polygraph, electrode, printer, screen, computer

place: central, frontal, occipital, eye, chin

## Abstract object:

waveform:

EEG: alpha, beta, delta, theta, sigma,  
artifact, arousal, K-complex

EOG: REM, SEM

EMG

activity:

time activity:

alpha time, beta time, delta time, sigma  
time, theta time, artifact time

count activity:

alpha count, beta count, delta count,  
sigma count, theta count, SEM, REMI, REMO,  
K-complex, arousal

sleep stage:

stage 0, stage 1, stage 2, stage 3, stage  
4, stage 5

EMG level: high, medium, low

number

time:

time point:

recording date, epoch number, latency

time span:

recording time, sleep time, stage time,  
epoch length

unit:

time: hour, minute, second, epoch

proportion: %

frequency: hz

voltage: microvolts

Actions: retrieve, sleep, record, detect, score

## State:

attributional stative

physical configuration

abstract stative

## Relationship:

logical: conjunction, disjunction, junction, negation

numerical: largest, larger, equal, smaller, smallest

configurational: part-of

Figure 4.2 The Representational System of Sleep EEG Domain



"Can you give me total alpha time?" the word "you" designates the system and is defined as "(\*SYS\*)" in the lexicon. The word "me" indicates the user who is defined as "(\*USER\*)." A sleep technician or clinician records or scores the data of sleeping subjects.

The physical objects are the "Picture Producers" (PPs) defined by Schank (1975). The PPs are conceptual entities which tend to produce an image in the mind of a listener. There are three sub-entities: person, man-made object, and place. In the sleep domain, a subject or patient is a "person" whose data are recorded during sleep. Since the class "subject" has more than one super-class such as person and animate entity, it is called the "tangled" ISA-hierarchy. For example, a subject named "Kimberley" will be defined in the lexicon as

```
(4.1) (SUBJECT LASTNAME (KIMBERLEY)
        GENDER (FEMALE))
```

where LASTNAME, GENDER are role names and KIMBERLEY, FEMALE are the fillers of the roles respectively. "Man-made objects" are substantial things which have been manufactured by humans for their own use, such as polygraph, electrode, printer, screen, computer, etc. in the sleep domain. "Places" are the body parts of a subject on which electrodes are located for recording the sleep signals (e.g., frontal, central, and occipital on the scalp for EEG signals, eyes for EOG signal, chin for EMG signal, etc.).

Many entities of the sleep domain belong to abstract objects. They are divided into five groups: waveforms, activities, numbers, time, and units. The electrical signals from electrodes attached to the subject body are recorded and measured. The technical details are available in Smith (1986) and Chang (1987). The EEG signal from the human brain is measured to detect several waveforms, i.e., alpha, beta, delta, theta, sigma, artifact, arousal, and K-complex. The EOG signal captures the eye movements in which REM (Rapid Eye Movement) and SEM (Slow Eye Movement) waveforms are detected. The EMG signal from the chin or muscle areas is used to indicate tonic muscle activity during sleep. The detected waveforms or features are further processed in an epoch-by-epoch approach so that four types of activity are produced: 1) time activity such as alpha time, beta time, delta time, sigma time, theta time, or artifact time, 2) count activity such as alpha count, beta count, delta count, sigma count, theta count, SEM, REMI (in-phase REM), REMO (out-of-phase REM), K-complex, or arousal, 3) sleep stage such as sleep stage 0, stage 1 or stage 5 (or REM sleep), and 4) EMG level such as high, medium, or low level. One of the frequently referred entities is a number because most of the sleep database elements are composed of numbers. The sleep database also contains a collection of temporal events which take place at a point of time or at a certain period of time. The temporal representation used here handles only the limited

types of time specifications. The specifications here are defined to represent the words and the inference rules enough in the sleep DB query system. The system represents a point of time (e.g., recording date, epoch number) and a duration of time (e.g., total sleep time, recording time, stage time). There are several units to represent a time duration (e.g., hour, minute, second, epoch), a proportion (e.g., %), a frequency scale (e.g., Hz), a voltage scale (e.g., microvolt), etc.

#### 4.2.2 Actions

NL questions in a sleep NL system can be represented as either an actional event or a stative event. The actional event is expressed in terms of primitive classes of actions. The verbs, for example, "give, list, show, display, and find" have an identical meaning in the sleep DB query system. This is represented as a primitive act RETRIEVE and defined in the dictionary as the following conceptualization:

```
(4.2) (RETRIEVE ACTOR W
        TO X
        OBJ Y
        INST Z)
```

where W, X, Y, and Z are empty slots to be filled.

Subjects or patients sleep at night and their data are recorded by a sleep clinician or technician. The waveforms are detected and the sleep stages are scored. The primitive acts

such as SLEEP, RECORD, DETECT, SCORE are necessary to cover events that may be entered by the user.

#### 4.2.3 States

The stative event in a sleep NL system is described in terms of attributes of entities, relationships among entities, or changes in these attributes and relationships. To represent a query as a stative event, there are three primitive statives: attributional stative (S-ATTR), physical configuration (P-CONFIG), and abstract configuration (A-CONFIG). The attributional stative represents the attribute value of an entity. An attribute "gender" of subjects, for example, can be asked in a query "Who are female?" and it has the following CD form:

```
(4.3) (S-ATTR CON (PERSON VAL (*?*))
      ATTR (GENDER VAL (FEMALE)))
```

The abstract configuration represents the non-physical relationship among the entities. For example, in a sentence "How many channels does Jones have?" the abstract configuration of possession between two entities can be represented as the following form:

```
(4.4) (A-CONFIG REL (POSSESS)
      CON1 (CHANNEL NUMREL (GRPNUM VAL (*?*))
      CON2 (PERSON L-NAME (JONES)
            GENDER (MALE)))
```

The physical relationship P-CONFIG is used to represent the locational relation among entities.

#### 4.2.4 Relationships

The state in a sleep NL system is expressed in terms of relationships among the entities and/or the actions. There are three subordinate relationships: logical, numerical, and configurational relationships.

As a logical relationship, conjunction for the word "and," disjunction for "or," or the junction for "," are used in English. For instance, a sentence "Show me alpha time and beta time?" is expressed in the following CD form:

```
(4.5) (CONJ CON1 (RETRIEVE ACTOR NIL
                    TO (*USER*)
                    OBJ (ALPHA-T)
                    INST NIL)
        CON2 (RETRIEVE ACTOR NIL
                    TO (*USER*)
                    OBJ (BETA-T)
                    INST NIL))
```

where the logical relation CONJ is used to coordinate two separate events.

Numerical relations are often used to compare two entities. There are several types of comparative relations such as GE (greater than or equal), GT (greater than), LE (less than or equal), LT (less than), and EQ (equal). For example, a query "sigma count is greater than 2" has the following form:

```
(4.6) (A-CONFIG REL (GT)
        CON1 (SIGMA-C)
        CON2 (NUMBER VAL (2)))
```

where two entities are on a comparative relationship of abstract configuration.

The configurational relationship is expressed between two entities such as POSSESS in (4.4) and GT in (4.6).

### 4.3 A Sleep Database System

A new sleep database system is designed as a menu-driven system that is used to edit sleep data. The menu-driven system allows users to add new subject data to the sleep database and examine or update the existing sleep database. The data in the sleep database can also be retrieved by means of retrieval query generated from SEEGER.

The sleep database has been implemented under the dBASE IV database management system (DBMS). The menu-driven system has been written in the dBASE language of dBASE IV, and the retrieval query from SEEGER is generated from SEEGER in the SQL (Structured Query Language) of dBASE IV.

#### 4.3.1 The Characteristics of the Sleep EEG Data

The information about sleep activities is obtained from the sleep waveform analyzing system (Chang, 1987). As a feature extraction system, the sleep waveform system analyzes the sleep EEG/EOG/EMG signals from the human brain during sleep and detects the waveforms of interest. The EEG signal from the central scalp is used to detect the waveform

occurrences of alpha, beta, delta, theta, sigma, artifact, arousal, and K-complex. The two EOG signals from the left and right eyes are used to detect REM (Rapid Eye Movement) and SEM (Slow Eye Movement) occurrences. The EMG signal from the chin is used to measure the amplitude level of muscle activity. The waveform occurrences are further processed and summarized during an epoch which is a 30 or 60 second interval. These epoch-wise summary data are composed of four types of activities: time activities, count activities, EMG level, and sleep stage as described in section 3.1. The processed epoch summary data are stored in a file.

The information about subjects or patients includes record number, recording date, age, sex, first name, middle initial, and last name. The record number is a unique identifier of the record.

The recording information includes channel number, channel place (e.g., A1, A2, C3, C4, chin), and channel description (e.g., EEG, EOG, EMG).

The sleep stage-wise parameters represent the information with respect to the sleep stages. It includes sleep stage time, sleep latency, and percent of sleep for each sleep stage. They are obtained from the deduction of the epoch summary data. For example, the sleep stage 1 time is obtained by counting the number of epochs with sleep stage 1, and then it is scaled to the appropriate time unit. The sleep latency is obtained by selecting the first epoch number of each sleep

stage from the epoch summary data and then scaling to the appropriate time unit.

The night-wise parameters as a single value include sleep time, recording time (i.e, time in bed), sleep efficiency, NREM time, and epoch length. The sleep time, NREM time, and sleep efficiency are obtained from the stage-wise parameters.

#### 4.3.2 The Structure of a Sleep EEG Database

The proposed relational sleep database is organized as shown in Figure 4.3. There are five database files: SUBJ\_INFO, CHAN\_INFO, EP\_SUM, STG\_VAR, NGT\_VAR. Each file has its attributes corresponding to the columns in a table. As relational databases (Date, 1986), all DB files have a field "record number" as a primary key. The "join" operation between the DB files is performed on the primary key. The physical DB structure information is stored in the DB mapping table of SEEGER. The domain concepts are mapped into the corresponding DB attribute names and the DB file names.

#### 4.3.3 A Menu-Driven System

SEEGER retrieves information from the database, but it cannot edit the database. A menu-driven system is developed to add new subject data or edit the existing sleep data. It allows the user to get into each sleep DB file and do the followings:

1. add a record
2. modify a record



<u>DB file</u>	<u>Attribute</u>	<u>Description</u>
SUBJ_INFO	REC_NO	record number
	LASTNAME	last name
	FIRSTNAME	first name
	SEX	sex
	AGE	age
	REC_DATE	recording date
EP_SUM	EP_LEN	epoch length
	REC_NO	record number
	EPOCH_NO	epoch number
	ALPHA_T	alpha time
	ALPHA_C	alpha count
	BETA_T	beta time
	BETA_C	beta count
	DELTA_T	delta time
	DELTA_C	delta count
	SIGMA_T	sigma time
	SIGMA_C	sigma count
	THETA_T	theta time
	THETA_C	theta count
	ARTIF_T	artifact time
	SEM_C	SEM count
	REMI_C	REMI count
	REMO_C	REMO count
EMG_LEVEL	EMG level	
AROUSAL	arousal	
K_CMLPX	K-complex	
SAC_STG	computer-scored sleep stage	
HUM_STG	human-scored sleep stage	
CHAN_INFO	REC_NO	record number
	CHAN_NO	channel number
	CHAN_NAME	channel name
	PLACEMENT	electrode placement
STG_VAR	REC_NO	record number
	SAC_STG	computer-scored sleep stage
	STG_TIME	total stage time
	LATENCY	latency
PERC_STG	percentage sleep stage	
NGT_VAR	REC_NO	record number
	NREM_TIME	total NREM time
	SLP_TIME	total sleep time
	NREM_PERC	NREM percentage
	EFFICIENCY	sleep efficiency

Figure 4.3 The Structure of Sleep EEG Database

3. delete a record
4. examine a record at any place
5. list records
6. group records by setting conditions
7. count the records.

The primary purpose of the menu system is to add new subject data. Data for one subject can be inserted by the following sequential steps:

1. enter subject information
2. enter the name of epoch summary file
3. enter channel information.

In the second step new subject data are automatically entered into the three DB files: EP-SUM file, STG\_VAR file, and NGT\_VAR file.

#### 4.3.4 Interface to SEEGER

SEEGER generates a retrieval query in the SQL form and stores it into a file. The database management system compiles the query file and evaluates it using the sleep database and then stores the result in a file. SEEGER then reads the evaluated output and shows it to the user.

#### 4.4 Conceptual Coverage

The conceptual coverage of the natural language system is the range of concepts that are covered in the system. Three factors were considered in defining the conceptual coverage. First, the concepts are specific to the domain. The concepts

in the sleep domain are related to sleep data, data recording events, data retrieval events, numeric comparison, etc. Second, the concepts should be so broad as to cover all conceivable language phenomena in a domain. Third, the concepts should be represented at a sufficient level of detail. These three are essential criteria for a good representational system.

The taxonomy of the conceptual coverage for a sleep NL system is based on the test results and the elements proposed by Tennant (1981) and listed in Figure 4.4. Content of two elements (i.e., domain concept representation and inference) is modified, and one new element (i.e., conceptualization of word) is added to the Tennant's list for knowledge-based NL systems. Each element will be described below.

#### 4.4.1 Domain Concept Representation

The representational system of the sleep domain is shown in Figure 4.1. The taxonomy of the domain concepts is represented in a semantic network in an ISA-hierarchy. The ISA-hierarchy of a domain reflects the property of the concepts to be classified and the operations performed on them. The top concept of the domain is classified into four main categories: entities, actions, states, and relationships. These are subdivided into a number of their subordinate types. Although Cullingford (1986) suggested three criteria for a good representational system as described in Chapter 3, there

1. Domain concept representation
  - entity
  - action
  - state
  - relationship
2. Conceptualization of word
3. Capability of the system
  - retrieval of sleep data
  - problem deduction
  - data segmentation
  - exceeding the limits of discourse
4. Inference
  - specification inference: number
  - omission inference
  - concretion inference: time
  - substitution inference
5. Knowledge about DB
  - DB values, fields, files
  - DB model
  - DB mapping
    - nested query
    - multiple query
    - expensive query trap
6. Logical propositions
  - negation
  - disjunction
  - conjunction
  - quantification: universal and existential
7. Quantitative propositions
  - numerical quantifiers
  - comparatives
8. Numerical functions
  - ordinality
  - cardinality
9. Discourse objects
  - sentence
  - topic of conversation
10. Extensions
  - new concepts - synonym
  - clarifying of terms
  - altering a formal database
  - updating content

Figure 4.4 Conceptual Taxonomy for SEEGER

is no objective way to evaluate the concept representational system.

#### 4.4.2 The Conceptualization of Word

This is a knowledge encoding process into the words in the lexicon. It is based on the notion that the word in a restricted domain has a limited number of meanings. The wrong or missing word senses was one of the most serious problems of SEEGER. Consider a word with missing word definitions in the following request:

(4.8) Where was the highest sem count for record 34989n?

The word "where" was only encoded as "a place for electrode," but it means "epoch number" here.

#### 4.4.3 Capability of the System

SEEGER is able to retrieve the sleep DB data or their aggregates. The aggregate functions include counting of the number of records, summing the values, average of the values, minimum of the values, and maximum of the values.

Although the required information is contained in the database, it cannot be extracted with a single query. For example, in the following sentences

(4.9) Show me longest stage 2 period from record 34989n.

(4.10) Give me the ending epoch number of the longest contiguous block of stage 2 for 34989n.

the phrase "longest block of sleep stage 2" can be resolved by the problem deduction for data block segmentation. The system must develop a plan for answering the request which deduces the problem to simple multiple sub-problems. SEEGER currently has no problem deduction capability.

It is assumed that the user will confine his discourse to the system capabilities. If it cannot get through the interpretation of the user's requests, SEEGER considers that an error has occurred. If SEEGER encounters an unknown word, it assumes that the user misspelled the word and tries to find a possible candidate. If one is found, it will be suggested to the user for approval. Otherwise, it will give the user an opportunity to enter a correct word or enter a synonym.

#### 4.4.4 Inferences

SEEGER, as a knowledge-based system, uses an intermediate meaning representation that is based on the conceptual content rather than on the literal content of a request. It has the facility for inferential analysis in terms of conceptual content. It shifts the meanings into an appropriate form for the retrieval code via multiple steps of transformations.

One of the most complicated concepts is "time" in the sleep domain due to the data characteristics. Consider, for example, the following requests:

- (4.11) Show me time that Jones was recorded.
- (4.12) When was Jones recorded?
- (4.13) How long was Jones recorded?

The above meanings can be easily obtained via inferential analysis such as "recording date" for (4.11) and (4.12), and "recording time" for (4.13). There are different levels of detail in the time data. For example, in the following sentences

- (4.14) Show me total time that K-complex has occurred.  
 (4.15) Show total time that Jones was at sleep stage 2.

the phrase "total time" means "the number of epochs" in both (4.14) and (4.15). But it will be inferred as "sleep stage time" in (4.15) since the attribute is already in the sleep database and its value can be quickly retrieved.

The following concept association will be implicit in the following requests:

- (4.16) Who has sleep stage 4 more than 20%?  
 (4.17) Who has sleep stage 4 more than 2 hours?

The phrase "sleep stage" means "percent sleep stage" in (4.16) and "sleep stage time" in (4.17). It will be inferred by means of the unit about time or proportion.

Some of the time concepts will be defined clearly in the lexicon. For example, "arousal time" in the lexicon is defined as "epoch number whose arousal occurred."

The implicit meaning of a number can be inferred by domain-specific knowledge (Shwartz, 1982). In the following requests, for example,

- (4.18) Show me all subjects from 20 to 35.
- (4.19) Show me all subjects from 1987 to 1988.
- (4.20) Show me all subjects from 34500 to 34570.

SEEGER will infer that (4.18) refers to subject age, (4.19) refers to recording year, and (4.20) refers to record number.

A missing inference rule caused a request to be misinterpreted in the following:

- (4.21) What was the name of the record on 2/5/89?

The phrase "name of the record" refers to "record number," but it was misinterpreted as "subject name."

#### 4.4.5 Knowledge About the Database

SEEGER's goal is to produce retrieval queries. To transform a meaning representation into a query language, it is necessary for the system to know about the database.

4.4.5.1 Database elements. The knowledge about the data elements in the DB plays an important role to guide NL analysis in NL query systems. In SEEGER, the ISA-hierarchy of the domain concepts contains the classes of database values. There is a limited number of nominal data in the sleep DB because most of them are numbers, and they will be automatically defined. During the system initialization, the subject information such as record number, name, sex, and age is retrieved from the DB and stored in the system's memory. For example, the subject JONES is defined as



```
(4.22) (WORD JONES
        POS NAME
        DEF (PERSON LASTNAME (JONES)
            GENDER (MALE)))
```

and stored in the system's memory like a word in the lexicon. Since the record number can be either a number or a string, all record numbers are stored into the "Record List" in order to be recognized immediately during the parsing. This automatic definition is made for the first names and last names, and it is not necessary to code manually for changing data elements.

4.4.5.2 Database model. It is usually assumed that a user's request is correct. Thus, it is necessary to recognize and respond to the user's misconceptions about the domain or the database. The COOP system (Kaplan, 1979) detects a user's presuppositions in the request if the evaluated query returns an empty response. For example, in the following request

(4.23) Show me Kimberley's total arousals on 2/5/89.

if the result were empty (i.e., the total arousals is zero), COOP would check whether the presupposition (i.e., existence of is any record on 2/5/89 for Kimberley) is valid or not. If there were any misconceptions on the user's part, the system would respond as follows:

(4.24) I don't know any record on 2/5/89 for Kimberley.

SEEGER assumes that the user's request is correct and currently does not check any presuppositions.

4.4.5.3 Database mapping. It is necessary to perform a validity check before the DB language is generated. There may be NL requests for which no corresponding target DB expression exists. Some requests may require nested expressions or multiple queries. For example, the following request

(4.25) Show me sleep efficiency and total sleep stage time.

should be divided into two separate individual queries since "total sleep stage time" is given for each sleep stage, but "sleep efficiency" is a single value regardless of the sleep stage. A query involving a join of two DB files may require a long time to execute. If it takes a long time for a query to execute, the system should inform the user of how long the query will take. SEEGER currently does not support nested expressions, multiple queries, or an expensive query trap.

#### 4.4.6 Logical Propositions

The predicates that were not explicitly specified are assumed to be logically ANDed. Though the logical relationships between predicates are explicitly uttered in the NL sentences, the relationships sometimes depend on the physical DB structure. For example, in the following sentences

- (4.26) Show Kimberley's arousal time in sleep stage 2.  
 (4.27) Show me arousal time in sleep stage 1 and 2.

the two-predicate relationship between "last name is Kimberley" and "sleep stage is 2" is ANDed by default in (4.26), but the logical relationship between "sleep stage is 1" and "sleep stage is 2" in (4.27) is ORed in the DB language expression.

The logical negation of a predicate is handled in limited cases. For instance, in the following sentences

- (4.28) Who is not older than 20?  
 (4.29) Who has no sleep stage 4?

the negative predicate of "age > 20" is modified into a positive predicate "age <= 20" in (4.28). But (4.29) can be handled with a negative EXISTS predicate in the DB language.

#### 4.4.7 Quantitative Propositions

Numerical quantification is uttered as either a noun head or a modifier of a noun phrase. The number is handled as a modifier in the phrase "3 sigma counts" or "sleep stage 3" but as a noun head in the phrase "20 seconds, 20 minutes, or sigma is 3." It cannot handle a specific number of rows in the records such as

- (4.30) Show me 5 subjects recorded on 2/5/88.

since the target SQL language does not support this facility.

#### 4.4.8 Numerical Functions

Counting the number of rows of the records, summation of DB value, and averaging of DB values are handled in SEEGER. The cardinality or ordinality is supported in the SQL language. But one inference rule shifts the meaning in the phrase such as "first epoch number of sleep stage 1" to "sleep latency of sleep stage 1."

#### 4.4.9 Discourse Objects

Discourse objects include objects described in the previous discourse and sentences themselves (e.g., what was my last question?). Like most other systems, each question is considered as a query onto itself. It does not maintain the topic of conversation. For instance, in two consecutive sentences

- (4.31) Show me male subjects between 20 to 30 years old.  
(4.32) Was any of them recorded on 2/5/88?

The word "them" in (4.31) refers to the restricted set of subjects described in (4.32). SEEGER does not store the objects generated from the answer. SEEGER has no capability to recognize discourse objects.

#### 4.4.10 Extensions

Synonyms can be added by users during interaction occurring due to the lack of vocabulary. For example, in the

following interactive dialogue

(4.33)

User: How much alpha did kimberley have in stage 1?

SEEGER: I don't understand the word MUCH.

Would you like to: (1) Enter a correct spelling  
 (2) Propose a synonym  
 (3) Ignore the word  
 (4) Reenter the entire request

Enter a selection --> 2

Enter a word --> many

OK ...

SEEGER allows the user to define the synonym. SEEGER has no capability for users to define a new word or concept. It cannot support requests to clarify the terms. It cannot update or delete the data from database through NL questions.

#### 4.5 Linguistic Coverage

The linguistic coverage of a NL system is a description of the ways in which concepts can be expressed. Three factors were considered in the syntactic facilities for SEEGER. First, the variety of phrases expected from users is deeply related to the domain. In SEEGER, it is expected that many verbs will be "be, have," and a retrieval type (e.g., list, give, show). Many expressions are expected about numerical functions and temporal references due to the sleep data features. It is unlikely to use many adverbs and adverbial phrases. Second, there is a variability among the users. People have their own idiosyncratic ways of expressing themselves. Some of the requests will be ungrammatical due to the user's carelessness

or informal linguistic style. Third, any system at the current state of technology will have limitations. Most systems provide no clues as to whether the problem is due to spelling, an unknown word, a wrong construction, or a reference to an unknown concept.

The elements in Figure 4.5 cover a wide variety of syntactic facilities for a sleep NL system. They are based on the test results and the elements proposed by Tennant (1981).

#### 4.5.1 Reference to Concepts

4.5.1.1 Reference by name. A name constitutes a word or phrase in the lexicon as a lexical item. SEEGER accepts the name as a reference to the concept like other systems.

4.5.1.2 Reference by description. There are many ways to describe concepts such as noun modifiers, prepositional phrases, clauses, etc. Some structure constructed from the description will be processed via inferential analysis in order to synthesize the concepts or to concrete the concepts as described in Section 3.3.

The noun prenominal modifier described here precedes a noun and is processed as a noun group in SEEGER. The meaning of determiners is ignored in SEEGER since it was considered that person-number disagreements are common in informal requests. The identifiers in SEEGER are record number, subject name, and sleep stage. Consider, for example, the noun modifiers in the following request:

1. Reference to concepts
  - by name (alpha, alpha time)
  - by description
    - noun pronominal modifiers
      - determiners (a, an, the)
      - adjective (male, female)
      - identifier (record# 34589, stage REM, name Carter)
      - quantifiers (each, every, all)
      - noun-noun modifier (sleep stage 1 time)
      - genitive (Kimberley's arousal)
    - noun postnominal modifier
      - participle (subject recorded on 2/5/89)
      - infinitive clause
      - relative clauses
      - prepositional phrase
  - anaphoric reference
    - pronouns (he, she, they, it, I, you, one)
    - antecedents and referents (the last subject)
2. Wordng for brevity
  - ellipsis (What is the record# of Jones? age?)
  - non-anaphoric omissions (REM sleep > 10 seconds)
  - relying on the domain knowledge
    - (Show me epochs with K-complex [greater than 0].)
3. Syntactic forms
  - basic types
  - special structures
    - dates (2/5/89, Feb 5, 1989, Feb. 5, 1989)
    - person name (Lisa J. Kimberley)
    - percentage of stage time (% stage 1, stage 1%, percent stage 1)
    - mathematical expressions (total recording time - sleep stage 0 time)
  - sentence
    - fragmentary (how many subjects?, data available?)
4. Negations
  - not, no, never, only, other than, except
5. Conjunction
  - clause
    - prepositional phrase
    - noun phrase
    - pronominal modifier
    - other structures
6. Ill-formed sentence
  - missing a word or preposition
  - misplacement of symbol

Figure 4.5 Linguistic Taxonomy For SEEGER

(4.34) Show me the sleep stage 1 time.

The modifier "the sleep stage 1" is a combination of determiner ("the"), noun-noun ("sleep stage"), and identifier ("1") modifiers. The above noun group will be processed to be represented into one structure such as

(4.35) (TIME REL (STAGE VAL (NUMBER VAL (1))))

The noun postnominal modifier follows a noun and is not processed as a noun group in SEEGER, but its structure is attached to the noun group. Relative clauses in SEEGER are handled based on the individual relative pronouns. For example, in the sentence

(4.36) Show me subject who is older than 50.

the word "who" is recognized as a relative pronoun since it is neither an imbedded sentence (e.g., Tell me who is ...), nor a question pronoun (e.g., Who is ...). The preceding ANIMATE entity (i.e., subject) is copied and modified by the class ACTION or STATE (i.e., is) which comes after "who." The overall structure is

(4.37)  
 (RETRIEVE ACTOR (NIL)  
   TO (\*USER\*)  
   OBJ (SUBJECT REL (S-ATTR CON (SUBJECT)  
     ATTR (AGE > (NUMBER VAL (50))))))



The relative clause is connected to the preceding ANIMATE entity with the slot REL. Other types of relative clauses without pronouns are given as

(4.38) Show me all subjects you have.

If there are two independent head patterns in the class of ACTION or STATE, the latter is linked to the preceding entity in the same way as (4.37).

Prepositional phrase (PP) attachment is a problem encountered in pure syntactic parsers. SEEGER determines the attachment in top-down and bottom-up manners. If there are any expectations that will be fired along with a PP, the PP will be bound to the slot. Otherwise, it will be attached to a preceding noun group. For example, in the sentences

(4.39) Who was recorded EMG waveform on 2/5/88 from the chin?

(4.40) Show me recording time in Jones's record.

both "on 2/5/88" and "from the chin" in (4.39) are attached to the verb "recorded" by means of the expectation demons. The PP "in Jones's record" in (4.40) has no place to be bound by expectation. It will be bound the preceding entity "time."

4.5.1.3 Anaphoric reference. Anaphoric reference in SEEGER is considered only for pronouns. The pronouns can be resolved in the same manner as the word sense disambiguation. The entities uttered in the previous and current sentences are stored in the Pronoun Buffer (PB). When a pronoun is

encountered in a sentence, a function draws the candidate referents matching semantic content from the PB. SEEGER only handles the pronoun referring to the entity class, but it cannot handle the referent to a clause or a sentence.

#### 4.5.2 Wording for Brevity

4.5.2.1 Ellipsis. Ellipsis is resolved at the level of the conceptual structure. If there is a fragmentary sentence, elliptic expansion is tried first. For example, consider the two consecutive questions

- (4.41) Find Kimberley's total alpha time of sleep stage  
1.  
(4.42) How about sleep stage 2?

the meaning structure of (4.41) is given as

```
(4.43)
(RETRIEVE ACTOR (NIL)
  TO (NIL)
  OBJ (ALPHA-T QUANTIFIER (TOTAL)
    POSSBY (PERSON LASTNAME (KIMBERLEY)
      GENDER (MALE))
    PG-REL (STAGE VAL (NUMBER VAL (1))))))
```

Since the top concept of (4.42) is STAGE, the meaning structure of (4.41) is searched for the concept STAGE in the bottom-up manner. Once it is found, conversation exchange (Cullingford, 1986) is performed. That is, all upper and lower level structure is copied to the structure of (4.42). If is not found, SEEGER considers that the sentence is fragmentary without the retrieval type of verb such as "show."

4.5.2.2 Non-anaphoric omission. Since numeric comparisons are expected in the sleep NL system, a symbol is accepted instead of a phrase. For example, the symbol ">" is allowed to replace "greater than" or "larger than." Other symbols such as ">=", "=", "<", and "<=" are accepted as well.

4.5.2.3 Relying on the domain knowledge. People tend to leave out unnecessary information that is required for query formulation. For example, the request such as

(4.44) Show me epochs with sigma spindles at sleep stage REM.

really means

(4.45) Show me epochs with sigma spindle [greater than 0] at sleep stage REM.

rather than

(4.46) Show me epochs and sigma spindle at sleep stage REM.

The phrase "greater than 0" can be inferred from the knowledge about sleep data. Other types of incomplete query have been described in Section 3.6.

### 4.5.3 Syntactic Forms

Three types of sentence structures are expected in a sleep NL system: declarative, interrogative, and imperative. If the sentence is a yes-no type of question, the question is

transformed to supply the data rather than just answer in "yes" or "no" since it is assumed that the users are interested in the data behind the answer. SEEGER also accepts a single sentence.

Dates can be expressed in different forms. To specify a recording date in the sleep database, the following forms, for example, are accepted: "2/5/89, 2/5/1989, February 5, 1989, Feb 5, 1989, Feb. 5, 1989, Feb. 1989, 1989," etc.

A fragmentary sentence in a noun phrase will be processed as an acceptable input, such as

- (4.46) How many subjects?
- (4.47) data available?

It will be inferred as the retrieval verb "show me" is omitted.

#### 4.5.4 Negation

The negative requests may contain explicit negative words such as "not," "no," and "never" or may contain the implicit negatives "only," "except," and "other than." This is a one of the features poorly handled in NL systems since it is a very difficult problem. The relational DB language also does not support a complement of a set of data. The current implementation of SEEGER only interprets explicit negation and generates a DB language for a valid request.

4.5.5 Conjunction

The scope of the conjunction is a difficult problem for any parser. The conjunction is considered as a coordination of events at the conceptual level in SEEGER. Consider the following sentences:

- (4.48) Show me subject# and age of John.
- (4.49) Show me subject# of John and Lisa
- (4.50) Show me subject# of John and age of Lisa.
- (4.51) Show me subject# of John and show me age of Lisa.

The preceding concept and following concept of "and" are examined. If they are at the same semantic categories in the ISA-hierarchy, a conversation exchange is performed on both the preceding concept and the following concept in the same manner as the elliptic expansion described above. Otherwise, the bottom-up search at the preceding concept is performed in the same manner as ellipsis. For example, (4.50) and (4.51) will have the following final structure as

```
(4.52)
(CONJ CON1 (RETRIEVE ACTOR (NIL)
           TO (*USER*)
           OBJ (SUBJ-NO PG-REL
                (PERSON FIRSTNAME (JOHN))
                GENDER (MALE))))
      CON2 (RETRIEVE ACTOR (NIL)
           TO (*USER*)
           OBJ (AGE PG-REL (PERSON FIRSTNAME (LISA))
                GENDER (FEMALE))))
```

Two events are joined as shown above. Other words "or" or ", " are handled in this way. The logical proportion of "and" or "or" in the request does not necessarily correspond to the

logical meaning in the database. For instance, in the following sentences

(4.53) Find Kimberley's total alpha time of sleep stage 1 and sleep stage 2.

the conjunction "and" in (4.53) corresponds to the disjunction in the conditional relation of the sleep database.

This conjunction handling in SEEGER is limited but many questions can be analyzed. But in the following sentence "Show me subject# and age of John and Lisa." two separate coordinations cannot work at the current implementation.

#### 4.5.6 Ill-Formed Input

Some requests were ill-formed because the users missed a preposition or misplaced an apostrophe such as

(4.54) Who was recorded 2/5/89?

(4.55) What is Hykomens' record number for 2/5/89?

(4.56) Show me Kimberleys record number.

The user missed the preposition "on" in (4.54), misplaced the apostrophe as "Hykomen's" in (4.55), and missed the apostrophe as "Kimberley's" in (4.56). SEEGER cannot handle any of them currently.

#### 4.6 Implementation

SEEGER was written in Golden Common LISP and runs on a 286-based AT personal computer under DOS 3.2. I believe that

LISP is more appropriate than any other programming language such as PROLOG or C. PROLOG (Clocksin & Mellish, 1984) is designed specifically to support logic formalisms so it provides certain classes of inference as a natural by-product of a logic representation. Thus, it is easier for simple theorem provers to be developed and written. But in order to build a SEEGER-like system requiring various types of knowledge structures and complex control mechanism, LISP is more flexible than PROLOG. LISP is also more convenient for symbol manipulation than C.

The sleep EEG database has been built under dBASE IV Database Management System (DBMS) on the same computer. The dBASE IV DBMS provides not only the dBASE language used to create and update the sleep database, but the SQL (Structured Query Language) used to retrieve the information from the sleep database through SEEGER. There are five DB files as discussed in Section 4.3. The epoch summary file occupies about 200 kilobytes of storage for five subjects. But other DB files (i.e. subject record, channel recording, sleep stage summary, or night summary DB file) take close to 2 kilobytes of storage for five subjects. Although any number of subject records can be added to the sleep DB, it will take a longer time for a query to search the epoch summary file. The SQL language of dBASE IV is known as slow and awkward because of internal translation from SQL to dBASE language equivalents and handling of special SQL catalog files. This causes the SQL

expression from SEEGER to be executed slowly. A SQL-based DBMS is desired to be used in order to reduce the execution time.

The entire program has been written in about 10,000 lines of LISP. The system requires 6 megabytes of system memory for the SEEGER program and the LISP environment. The system also requires 15 megabytes of hard disk storage for the Golden Common LISP environment, dBASE IV, the SEEGER program, and the sleep database.



## CHAPTER 5 EVALUATION OF SEEGER

SEEGER is an experimental natural language interface to a sleep EEG/EOG/EMG database. It has been designed and implemented based on the NL query model as described in Chapter 3. SEEGER has been tested on 79 requests by four novice users who are sleep clinicians or researchers familiar with sleep data analysis. Since there is no existing sleep NL query system to compare with SEEGER, SEEGER's performance is discussed regarding the interactive dialogues, system failures, sleep database, and timing. The results show that the preliminary design goal is met, and they provide information on what is needed to improve the system.

### 5.1 Test Descriptions

Four users tested SEEGER. The first user was a hospital sleep technician, the second user was a sleep researcher interested in drug efficacy, the third user was a sleep system developer, and the fourth was a sleep researcher. All of them are familiar with the sleep EEG area and have experience with sleep data analysis. In other words, they are familiar with

the domain of discourse. They had never used SEEGER before. Before the test, they were given the written user's guide to SEEGER described in Appendix A. It briefly describes the introduction of the system, a sample dialogue, and the terms related to sleep EEG data available in the database. They were not required to read through the user's guide.

The users were not given any specific problems to solve. Instead, they were told to solve any problems they had in mind since they were expected to have some problems they would like to solve from their own perspective. This was intentional in order to make the test more "realistic."

There were five records from three subjects in the sleep database for test purposes. Five records consist of one night's data from one subject and two night's data from the other two subjects.

## 5.2 Performance Summary

The test was conducted with a total of 79 questions from four users. Some of the questions were ill-formed due to a missing preposition or a misplaced apostrophe indicating the possessive case. Although an ill-formed input caused the system to fail, the result was classified as an improper response since a natural language system should be able to respond to ill-formed questions appropriately. The results are summarized in Table 5.1.

Table 5.1 Overall Test Results

Total number of requests	79
Interpreted correctly	63.2%
Not interpreted properly	36.8%

### 5.2.1 Performance Categories

The performance criteria are classified into three categories suggested by Lehnert & Shwartz (1983) as follows:

- IF: The request is interpreted correctly at the first try.
- IW: The request is interpreted correctly after one or more interactions.
- NI: The request is never interpreted correctly.

In the category IF, the requests were interpreted correctly and the correct paraphrase was presented to the user without any interactions. The category IW includes requests which were interpreted correctly by means of interactive dialogues. In the category NI, a system error is fatal in the sense that the user does not or cannot recover from the error. It happens when the requests failed in the course of interpretation, or the requests were interpreted incorrectly so the paraphrase was different from the user's intention. A list of all the requests from the users in the test of SEEGER is given in Appendix B.

The results for the three categories are given in Table 5.2. In terms of performance categories, 37% of all requests

Table 5.2 Test Results for Each User

	Number of Requests	IF	IW	NI	Success Rate
User 1	12	4	6	2	83.3%
User 2	38	15	9	14	63.2%
User 3	18	5	4	9	50.0%
User 4	11	5	2	4	63.6%
Total No.	79	29	21	29	
Total %		36.7%	26.6%	36.7%	63.3%

were IF requests, and 27% of all requests were IW requests. Both IF and IW requests are considered as fully functional. The 63% success rate achieves the preliminary design goal of performance level of 50 to 75 %. There are variations of success rate among the users of 83%, 63%, 50%, and 64%. Every user had his own behavior toward SEEGER. One was so curious to know the system's capability that he often typed the same question repeatedly. He continued to rephrase the question until he found the correct answer or became discouraged due to the system's repeated failures. Another user tried to understand the system's capability from the help facility (i.e., similar to the user's guide but contained in the system), so he entered requests that could be easily interpreted. Most of the users rephrased the questions more than once when SEEGER could not interpret the request. Therefore, the success rate in this test varied because of the users' disparate behavior and the system's spotty coverage of the various types of questions.

### 5.2.2 Interactive Dialogues

The elements of interactive dialogue in the test are classified in Table 5.3. Spelling corrections were made at least once for every user. All ten misspelled words were corrected since the correctly spelled words existed in the lexicon. There were four words with one letter missing (e.g., "availabe, kimberly, recrd, gree," where "kimberley" was

Table 5.3 The Elements of Interactive Dialogues and the Number of Occurrences for Each User

Elements	User1	User2	User3	User4	Total
Spelling error	1	2	6	1	10
Ambiguous concept				1	1
Incomplete request	5	9			14
Synonym definition			3	1	4

misspelled twice as "kimberly," and "gree" is the misspelled word of a subject name "greer"), four words with one letter extra (e.g., "reccord, mwe, fopr, withj"), and one word with one letter wrong (e.g., "artefacts," where it was corrected as "artifacts" although "artefacts" is a correctly spelled word). There is an undesirable failure in the spelling correction program of SEEGER. When there is any unknown word, SEEGER considers the word as a misspelled one and first tries to find a possible candidate. If one is found, SEEGER suggests it although it is a correctly spelled word to the user. It happened twice in the test (e.g. "take" was suggested as "wake" since "take" was not in the lexicon). This is unavoidable if a spelling program is to facilitate the friendly spelling correction.

Two users often entered requests without specifying subject information that is necessary to formulate a complete query. But two other users always entered the requests including the subject information such as the subject name, record number, etc. That is, some users will try to maximize efficient communication by minimizing the interactions with the system. At the same time, other users will try to enter a minimal request and let the system ask for more information as needed.

The synonym definition and the ambiguous concept clarification were rarely used. One user entered three synonyms for one unknown word (e.g. "continuous" for

"contiguous," "uninterrupted" for "continuous," and "segment" for "block"), but SEEGER could not recover from it since none of them were known. Another user entered one known synonym, and SEEGER recovered from its own recognition error (e.g., "many" was entered as a synonym of "much" in the request "how much alpha did kimberley have in stage 1?"). The users mostly wanted to enter an entirely new request when a correctly spelled word was not recognized by SEEGER. I think the synonym definition would work as a good facility provided the system adequately covers the vocabulary in the user's request.

### 5.2.3 System Failures

The failed cases are classified by their causes in Table 5.4. The elements in Table 5.4 were described in detail in the conceptual and linguistic coverage of SEEGER in Section 4.4 and in Section 4.5. Most of the failures occur during one of the following processes: parsing a noun group, parsing an entire request, translating into a VQL, and mapping into a SQL. The system developer can find where the system failed by either analyzing the intermediate results of each module or setting a flag which enables more detailed messages to be displayed. Misinterpretation of a request can be detected by examining the paraphrase presented to the user.

One of the major causes of failures in SEEGER is missing or incomplete word definitions in the lexicon. There were six cases for the known words in the lexicon:



Table 5.4 The Elements of System Failures  
and the Numbers of Occurrences

Elements of Failures	Occurrences
Exceeding domain discourse	
problem deduction	2
irrelevant to DB query	4
unknown concept	3
Word Meaning Definition	6
Inference rule	4
Ill-formed sentence	4
Conjunction	4
Query translation	1
Sentence form	
PP+S+V+O	1

- 1) In the request "where are the highest sem count for record 34989n" the word "where" asks about "epoch number," but it was only encoded as a question about "a place for electrode to be located" and misinterpreted.
- 2) In the request "how many nights are there" the word "night" refers to "recording date," but it was encoded as a concept "night." The request was failed during the DB mapping because the concept did not exist in the mapping table.
- 3) In the request "when was hykomen recorded" the word "when" asks about "recording date" as well as "epoch number." It was interpreted incorrectly because the word has one definition of "epoch number."
- 4) In the request "how long did hykomen sleep on the night of 02/05/89?" the word "sleep" was expected to be used as a noun, but it is used as an action as well. A noun group parsing error was given for the phrase "hykomen sleep on the night of 02/05/89" because of the two independent structures (i.e., "hykomen" and "sleep") in the Working Memory.
- 5) In the request "go to theta" followed by the request "show me delta time" the phrase "go to" is used to continue the same request for "theta time" instead of "delta time." The word "go" was just used to define the phrase "go to bed" in the lexicon and had no other definition for "go to" so the request failed during the

query translation.

6) In the request "how long before kimberley to fall asleep?" the word "before" was only expected to hear about a number or a value, but another meaning should be added to handle a request in which "before" is used along with a concept representing the time duration (e.g., "how long"). The request failed during parsing an entire request.

There are also many unknown words (e.g., "now, take, taken, as, hypnogram, leg, ending, contiguous, continuous, uninterrupted, block, segment, much"). When there was any unknown word, two different attempts were made: entering a new request or trying a synonym definition.

To define the word meanings is one of the most difficult tasks in developing SEEGER. With a bottom-up manner of defining word meaning, the model corpus is built up by collecting a set of representative sentences which contain different knowledge items. This is the knowledge extraction process undertaken by the SEEGER developer. When the corpus has been collected, the word meaning is encoded into a set of the primitive domain concepts through a "cut-and-try" process. Primitive item selection is a matter of judgement, and many different selections are possible. The basis of the choice is the level of detail that SEEGER communicates with the user. It is necessary to select appropriate items covering the collected model corpus. The main reason for missing or

incomplete word definitions is that I did not realize what the important knowledge items were, or I did not have enough knowledge of what the users wanted to say. Practically, it seems very difficult for the developer to know and codify all the knowledge of what the users express even in a narrow domain. The missing or incomplete knowledge items can be obtained through hands-on interactions with the users. Therefore, it is essential to have a testing phase for feedback purposes with a set of potential users in order to improve the performance.

Missing or incomplete inferential rules also led to wrong interpretation. A missing rule caused a request to be misinterpreted in the following requests:

- 1) In the request "what was the name of the record on 2/5/89?" was misinterpreted as "subject name" since the word "name" was encoded as "subject name," and the phrase "of the record" was removed by the omission rule. The user intended the underlined phrase to be "record number."
- 2) In the request "show me the lowest emg level on kimberley" the noun group "lowest emg level" was misinterpreted as "minimum emg level" instead of "emg level is low." It requires a rule saying that "lowest" means "low" when it is used along with "emg level."
- 3) In the request "list a summary of sleep stages for greer" there was no rule to handle a concept "summary"

about a concept "sleep stage." The request failed during the DB mapping.

Most of the inferential rules are specific to the sleep domain. One of the modules requiring rich domain-specific knowledge in SEEGER is the inferential rule module. Some knowledge is more conveniently incorporated in the inferential rules than in the individual word definitions. Many inferential rules are integrated with the word definitions when the model corpus is collected. A new rule should be inserted in an appropriate order in order to prevent misinterpretation of a request. But it is difficult for the developer to anticipate all inferences involved in the user's expression.

The user often entered the same question more than once because SEEGER did not provide an appropriate message to identify what was wrong. There was a sentence in which the user missed a preposition (i.e., he missed a preposition "on" in front of the date expression in the following sentence "Who was recorded 2/5/89?"). SEEGER responded with inappropriate message (i.e., "I cannot understand '2/5/89'.") since the word "record" expected a date expression to be used along with a preposition, so the data expression was left as an independent structure in the Working Memory. This led him to enter another ill-formed sentence (i.e., "Who was recorded 2/5/1989?"). Therefore, it is desirable for SEEGER to respond with all partial constituents that were successfully interpreted (e.g.,

"I understand 'Who was recorded' and '2/5/89'."). It will help the user recognize what is wrong in his request. One message (i.e., "The DB language cannot handle a query with two quantities: activity and night summary parameter.") was given to the user but could not be understood because it was in technical terms that he did not know. It is necessary to respond with the terms easily understandable by the user. The message often did not provide any relevant information (e.g., "I cannot interpret your request.") about the failed causes due to the lack of internal knowledge. In general, artificial intelligence (AI) programs do not know what they know, nor are they aware of what they can do. This lack of internal knowledge is one of the critical limitations in current AI programs. I believe it will not be solved in the near future. A good help system that offers guidance on rephrasing can inform the user what it knows and reduce the main source of frustration. The user also naturally adapts himself to the system's capabilities and operations quickly.

Many facilities were not implemented in SEEGER because they either required too much effort or were not expected to be used. One is the simple problem deduction capability for segmenting the data DB records. There were two cases (e.g. "give me the ending epoch number of the longest contiguous block of stage 2 for 34989n" and "show me the longest stage 2 period from record 34989n"). The data for the answer are in the database, but the SQL language does not support extracting

the answer. This facility could be implemented by creating a new database that provides the same information. It will require a set of new concepts in the lexicon and in the DB mapping module. Another problem was interpreting a sentence whose structure started with a prepositional phrase followed by a subject, a verb phrase (e.g. "for record 34989n show me the number of arousals and the total wake time and total artifact time"). This syntactic facility can be added without difficulty.

There were four ill-formed requests due to the user's carelessness:

- 1) In the request "what is hykomens' record number for 02/05/89" the apostrophe in the possessive form "hykomens'" was misplaced.
- 2) In the request "show me kimberleys record number" an apostrophe was missing in the word "kimberleys."
- 3) In two requests "who was recorded 02/05/89" and "who was recorded 02/05/1989" a preposition was missing in front of the date expression.

It is difficult to completely solve this problem because all possible ill-formed requests cannot be anticipated. There were three requests irrelevant to database queries (e.g. "which subject has the best looking legs, give tst, give sws, and what can I do now" where the last request was failed since "now" was unknown--but it would be failed even if the word "now" were known).

There were four failed requests:

- 1) what was the sleep efficiency and total arousals and k-complex?
- 2) what was the sleep efficiency and the total arousals?
- 3) give sleep efficiency and give total sleep time and give arousals.
- 4) list number of arousals and wake time and number of artifact for record 34989n.

They failed because of the limited capability of the coordinate "and." SEEGER was restricted to the handling of the coordinate between the attributes in the same DB file. The coordinate program should be modified to handle the above requests. However, the request "show sleep efficiency and total stage time" should be handled with two separate queries because the stage time will be given for each sleep stage, but the sleep efficiency will be given a single value.

There was one failed query translation since it was an unexpected type of request in the translation procedure (e.g., "is hykomen in the database"). This failure can be easily corrected.

### 5.3 The Sleep Database

There are five sleep DB files about subject record, epoch summary, channel recording, stage summary, and night summary. The data in the DB are retrieved by executing the SQL expression generated from SEEGER. There are three different



levels of detail in the sleep data. First, the epoch summary data are summarized from the waveform occurrences during an epoch which is a 30 or 60 second interval. Second, the stage summary data are time per sleep stage, sleep latency, and percent of sleep. Third, the night summary data include sleep time, recording time (or time in bed), sleep efficiency, NREM time, and epoch length.

Table 5.5 The Number of Requests for the Sleep Database Access

Sleep Database	Subject Records	Epoch Summary	Channel Recording	Stage Summary	Night Summary
Total #	29	16	3	3	14

Out of a total of 79 requests, 65 requests are classified by the DB access types in Table 5.5. The users requested the data from a single DB file or a joined DB file. The subject record file was joined with another file when the request included a subject name (e.g., "how many arousals did kimberley have"). As shown in Table 5.5, the subject information was requested more often than any other information. The night summary data were often asked as well.

The other five questions requested a join operation between the epoch summary file and night summary file (e.g., "what is the sleep efficiency and the total arousals"). The answer of the requests could be obtained by summing the values of a column in the epoch summary file.

There were another seven requests which could not be answered: one request could not be answered directly from the DB (e.g., "show me a sleep stage hypnogram of record 35009"); three requests were irrelevant to the sleep DB (e.g., "show me tst" and "which subject has the best looking legs"); and three requests were too general (e.g., "show me, what can I do now, data available").

Another two requests could not be answered since they required a new type of database (e.g., "show me the longest stage 2 period from record 34989n" and "give me the ending epoch number of the longest contiguous block of stage 2 for 34989n"). To accommodate these requests, the new DB should contain the information about the starting epoch number, the contiguous sleep stage period, and sleep stage number.

#### 5.4 Timing

The response time is divided into two elements: the time of request interpretation and the time of DB execution. The request interpretation time is the time taken to obtain the interpretation result since NL input was entered. The response is either the paraphrase or error message. The DB execution time is the time until the result is displayed after the paraphrase had been approved. It involves the time to load the DBASE IV environment in the system, to compile the SQL expression generated from SEEGER, to execute the compiled object file using the sleep database, to store the executed

result into a file, and finally to show the result to the user. The timing is summarized in Table 5.6.

Table 5.6 Average Timing for Request Interpretation and DB Execution

Interpretation Time	DB Execution Time
10.4 seconds	47.1 second

There was a variation in the interpretation time from 2 to 38 seconds; the average interpretation time was 10 seconds. The interpretation time was proportional to the number of active demons in the Working Memory during parsing a request. All active demons in the Working Memory are tested, executed if the test is true, and then immediately deactivated after it is executed. The testing of a demon mainly involves searching the Working Memory and takes a large portion of the interpretation time. The interpretation time is considered to be acceptable since no user complained about the response time.

The DB execution time varies from 26 seconds to 3 minutes. DB execution time mainly depends on the time taken to search the DB files. The epoch summary file (approximately 40 kilobytes per subject) is so large that it requires a long search time. If a SQL query is executed using a single DB file, it takes about 28 to 50 seconds. But a join operation

in the epoch summary file usually takes close to 3 minutes. It will be a serious problem for a large set of subject records when a join operation is involved in the epoch summary file. It also unnecessarily searches the whole file even though a query retrieves the data for any specific subject. The above problem would be resolved with a hierarchical type of DB linked through a file name. Instead of placing the epoch summary data for all subjects in one file, one file for one subject should be assigned to store the epoch summary data. Each subject epoch summary file would be connected through the subject record file which contains the file name for each epoch summary data. But this hierarchical DB will require a complex SQL language when a query requires a search of all subjects' epoch summary data. Therefore, it is desirable to maintain two different types of epoch summary DB files for a large number of subject records. It will reduce the DB execution time drastically although SEEGER will need to determine which DB file should be used for each type of query.

CHAPTER 6  
SUMMARY, CONCLUSIONS AND GUIDELINES  
FOR FUTURE DEVELOPMENT

Section 6.1 describes the summary of this dissertation. Section 6.2 discusses the conclusions, and in the last section the guidelines for future development are suggested.

6.1 Summary

This dissertation built and tested a sleep NL query system whose goal is to perform at the level of a data processing technician. A model for building a knowledge-based NL query system has been developed and designed into SEEGER. This model provides a more convenient method for incorporating domain-specific knowledge than has been used in the previous NL approaches because domain knowledge is crucial to interpretation of NL requests. The proposed model also provides an efficient database mapping method for those NL requests that rely on bottom-up parsing more often than top-down parsing. The domain concepts are represented in a hierarchical manner, and they are used to encode the word senses and the system knowledge. NL input is mapped into an

intermediate representation of its meaning, which is then translated into the retrieval query of the underlying database.

The proposed model is composed of the following components: the Conceptual Parser that transforms NL input directly into a Conceptual Dependency representation of its meaning without using an independent syntactic phase; the Inferential Analyzer that specifies implicit information explicitly, removes redundant information, and synthesizes the concepts, etc.; the Translator that transforms the meaning representation into a set of linear specifications; and the DB Mapper that transforms the linear forms into the underlying DB retrieval query.

The Conceptual Parser handles ungrammatical input, performs complicated word sense disambiguation, expands elliptic or fragmentary input, corrects misspellings quickly and easily, etc. However, since many demons are allowed to be operating in parallel in the Working Memory, it leads to complex unexpected interactions which may cause an incorrect assessment of the meaning structure of a sentence. One major reason for the system failures was missing or incomplete word meaning definitions and demons for existing words in the lexicon. Thus, it is necessary to carefully implement demons and word meaning definitions; moreover, it is essential to have a feedback phase from a set of potential users in order to collect a wide range of a model corpus.

The Inferential Analyzer provides a convenient method to incorporate contextual inference rules specific to a domain. Grouping of the rules invokes only relevant rules and avoids irrelevant rule interactions. But the forward-chaining rule interaction requires careful rule ordering when more rules are added in order to avoid ordering conflict. Premature rule triggering might lead to an incorrect interpretation by modifying the meaning structure of a sentence.

The query translation from the meaning structure to a VQL expression in the Translator provides an efficient intermediate stage for database mapping. The augmentation of query translation provides another type of inferences to VQL expression. However, it will be difficult to accommodate complicated queries or complex database operations into a VQL because of the constraints given in the VQL schema.

The sleep EEG DB consists of the information about subjects, recordings, epoch-wise summaries of waveform occurrences in EEG/EOG/EMG, stage parameters, and night parameters. The sleep DB is retrieved by retrieval query from SEEGER. New subject data are inserted into the sleep DB by a menu-driven system.

SEEGER was evaluated using four casual users who are familiar with sleep data analysis and had never used SEEGER before. Since SEEGER was designed to take a position that casual users should be accommodated, it was evaluated using casual users. The test was conducted for a total of 79

questions, and it gave correct answers at the performance level of 63%, achieving the preliminary design goal. Since there is a large amount of rules or knowledge involved in language understanding, it is difficult for novice users to evaluate a NL system extensively. The performance range is dependent on several factors: the user's knowledge about the system capabilities and domain, the quality of implementation, the nature of problems that users try to solve, and the formalism on the domain of discourse.

## 6.2 Conclusions

SEEGER was written in LISP. LISP is the primary language used in artificial intelligence programs. It was adopted in SEEGER since it is more convenient than C in order to manipulate symbols used to represent the system or domain knowledge, to define vocabulary, to communicate as input/output language, etc. It also provides efficient and flexible features such as recursion and functional language. The test results showed that the speed of SEEGER written in LISP on a 286-based personal computer is acceptable. One shortcoming is that SEEGER requires a lot of system memory (at least 6 megabytes). I believe LISP is the most appropriate language to date for a sleep NL query system.

dBASE IV is a dBASE-language-based DBMS rather than a SQL-based one. It is convenient to build the menu-based system that is used to add a new subject record. But the SQL language



of dBASE IV is known to be slow and awkward. Since SEEGER performs the interface to dBASE IV through the SQL language, I felt that the DB execution time was slow in the test. Thus, dBASE IV is not appropriate for the purposes of SEEGER. It would be better to select a SQL-based DBMS for a NL interface.

The proposed knowledge-based query model is designed to incorporate more domain-specific knowledge and to provide an efficient DB mapping. The conceptual parsing technique enables more domain knowledge to be embodied than the previous ones because it enables well-engineered programs to be built in a specific domain. The inferential analysis and the VQL augmentation provide a convenient mechanism to incorporate domain-specific knowledge at the different meaning representations. The VQL translation and the DB mapping also efficiently transform the meaning representation into a target DB language. This model is applied to building SEEGER and achieved the preliminary design goal.

It is necessary to add additional knowledge in order to improve the performance. A lexicon containing word definitions is the major source of domain knowledge. Adding knowledge usually starts with collecting a model corpus. Collecting a model corpus is the knowledge extraction process, and a good corpus is the major factor of determining how well the system covers a user's request. Knowledge addition involves adding or modifying word definitions, inferential rules, VQL translation and augmentation, or concepts in the DB mapping

table, etc. The whole process should be integrated through a "cut-and-try" process.

Understanding natural language requires considerable knowledge about the subject. A system which truly understands natural language should have a level of knowledge equivalent to that of an expert system. Therefore, a NL developer must be familiar with the application area because the process of embodying knowledge requires the same level of domain knowledge that the user employs.

Based on the test result as well as the elements of syntactic coverage, semantic coverage, and interactive dialogues, I feel it would be possible to build a sleep NL system performing at the level of a data processing technician, but it would take at least two man-years of effort, assuming the developers already possess some background in artificial intelligence programming. It would be desirable to assign half of this development time to a "test-and-tuning" cycle since it is essential to have a testing phase with a set of potential users.

Although SEEGER focuses on the sleep EEG/EOG/EMG data, it can be expanded to include other databases such as respiration, heart rate, and other sleep variables. Those databases will contain temporally occurring events and/or their summaries. It would require a more complex temporal reasoning. Furthermore, future research can incorporate more knowledge so that the system can provide not only information

in the database, but also terminology explanation, sleep stage scoring, modification of sleep data, etc. The proposed model can also be utilized to build a NL system in any specific domain.

### 6.3 Guidelines for Future Development

Based on the development experience and the test results of the sleep NL query system, the following guidelines are recommended for future development:

1. Expand the vocabulary. Modify the dictionary items which are word meaning definitions and associated demons. It includes modification of the existing dictionary items or addition of new ones. Collect a good model corpus. Cooperate with the potential user to cover a wide range of corpus, and always annotate the reasons for modifying the corpus. Consider what facilities or elements would be implemented or modified and in what manner while collecting the corpus. This task will often involve modifying other parts of SEEGER such as the inferential rules, the query translation, or the DB mapping. If a new concept is added, it should be incorporated into the ISA-hierarchy of the domain concepts, the inferential rules, the DB mapping tables, etc. Therefore, the whole process should be integrated into the system through a "cut-and-try" process.

2. Elaborate the error handling. The system should be able to provide helpful information when an error occurs. The current error handling needs improvement. Most of the system failures occur at one of the following processes: parsing a noun group (NG), parsing an entire sentence, translating into a VQL and augmenting it, mapping into a SQL query. If a parsing error occurs (detected by the number of independent structures in the Working Memory or in the NG pool), explain what constituents have been interpreted and/or what constituents have not been interpreted. Expand the validity check routine at the VQL augmentation while the model corpus is collected. Present an error message to be understood easily by the user.

3. Maintain both the relational DB and the hierarchical DB for the epoch summary data. The epoch summary data should be handled in a special way for a large set of subject records because of the long DB search time. Two DB types should be maintained for the epoch summary data: relational and hierarchical. SEEGER should be modified in order to select either of two DBs, depending on the type of SQL query. A separate DB mapping routine should be added for the hierarchical DB.

4. Handle extensive query language. The current DB mapping only handles a single query. Expand the DB mapping capabilities to handle nested SQL queries and multiple SQL

queries. The nested query enables some requests with negative expression to be handled. The multiple query enables some currently invalid requests to be handled. The multiple query requires the execution of the sleep DB multiple times.

5. Enhance the VQL schema. The VQL schema is loosely designed. Although low- and medium-complexity requests can be handled with the current VQL schema, it could be enhanced in order to handle queries requiring complex DB operation.

6. Use more uniform knowledge representation. It is difficult to trace and debug the program since it was written in a combination of several different forms such as procedures, rules, tables, and declarative forms. It would be desirable to represent the knowledge in the declarative forms and rules if possible. It could be easier to maintain the program and modify the knowledge.

APPENDIX A  
USER'S GUIDE TO SEEGER

The database (DB) management system provides efficient storage and management of data. But it requires the user to learn a query language syntax and the DB's structure. SEEGER (Sleep EEGer) is a DB front-end system to access a sleep EEG DB using ordinary English queries. It allows sleep clinicians to retrieve information without the need for any database expertise. The user can enter queries about sleep information in common English. The user will be guided by interactive dialogue for an inadequate or ambiguous sentence. Once the request has been analyzed, the system presents the paraphrase of the request. Finally it generates the query language, runs it in the database, and shows the result of the request.

Be aware that SEEGER is able to understand queries about information in the database. Additionally, SEEGER allows the user only to retrieve the data, not to modify or delete the data in the database. To enter a query, just type a sentence at the prompt and press the return key. Then read through the system's response to aid in forming your question.

To help the user to know about the sleep DB content and the way to interact with SEEGER, the following information is provided on the next pages, 1) a sample dialogue, 2) the content of sleep data, 3) the request types of sleep data, 4) sleep terms understood by SEEGER and sample queries, 5) getting started, and 6) when the system has failed.

### 1. A Sample Dialogue

A sample dialogue with SEEGER is show below (user input in italics).

-----

Ready  
 -->*Show me Kimberley's total number of aruosals for each stage.*

OK ...  
 By ARUOSALS, do you mean AROUSALS (Y or N)? y

I understand your request to be:

"Display record number, total arousal, stage number, last name with stage number = 0 and stage number = 1 and stage number = 2 and stage number = 3 and stage number = 4 and stage number = 5 for last name = KIMBERLEY."

Is it right (Y or N) y

..... running database, please wait .....

-----

The initial user input was a request for the sum of a sleep variable (i.e., arousal) for different sleep stages. The user also specified a subject name (i.e., Kimberley). The user

misspelled "arousals" in this example, and the system responded with a correct spelling. The system then responded with a paraphrase which is a system's interpretation of the user's request. After displaying a paraphrase the system goes into a verification phase, offering the user the opportunity to verify the request before the output is actually produced. If you agree with the system's interpretation, it runs the database. Otherwise, it starts again.

After running the database, the output is shown as below. The arousal activity is summed for each sleep stage number.

-----

Q U E R Y   P A R A P H R A S E

"Display record number, total arousal, stage number, last name with stage number = 0 and stage number = 1 and stage number = 2 and stage number = 3 and stage number = 4 and stage number = 5 for last name = KIMBERLEY."

RECORD NUMBER -----	TOTAL AROUSAL -----	STAGE NUMBER -----	LAST NAME -----
34989N	68	0	KIMBERLEY
35008	16	0	KIMBERLEY
34989N	9	1	KIMBERLEY
35008	4	1	KIMBERLEY
34989N	6	2	KIMBERLEY
35008	4	2	KIMBERLEY
34989N	0	3	KIMBERLEY
35008	9	3	KIMBERLEY
34989N	0	4	KIMBERLEY
35008	0	4	KIMBERLEY
34989N	0	5	KIMBERLEY
35008	0	5	KIMBERLEY

!!!    End of Record    !!!

-----



At this point you can send the output to the printer or try another request. The above request was sufficiently specified for the system to interpret it without further user interaction but many requests are likely to require an interactive dialogue. For example, the following interaction might take place with a new user.

---

Ready  
--> *Give me delta.*  
OK ...  
  
By DELTA, do you mean  
    (1) delta time  
    (2) delta count  
Enter selection = 1  
  
Is the record number 35015N (Y or N) y  
OK ...  
  
I understand your request to be:  
  
"Display record number, epoch number, delta time in second for  
record number = 35015N."  
  
Is it right (Y or N)

---

Since the activity "delta" is ambiguous to the system, the clarification dialogue is initiated to have the user choose either "delta time" or "delta count." The user is given a menu to select one. The system also offers the user an option of inheriting a subject specification from a previous request.

The user can start with a request using a verb (e.g., list, display, give, show, want, etc), wh-question (what, how, who, which, etc), yes-no question (be, do, etc), etc.

## 2. The Content of Sleep Data

The SAC (Sleep Analyzing Computer) system detects the waveforms normally occurring in sleep EEG, EOG, EMG data. The detected features are further processed and stored in the system memory. The data in the database is supplied by the SAC system. The minimum time resolution of activity data (i.e., alpha, beta, delta, sigma, theta, artifact, SEM, REMO, REMI, arousal, K-complex, sleep stage) provided by SEEGER is the duration of an epoch. The epoch duration of 30 seconds or 60 seconds is selected by the user of the SAC system. The sleep stage is scored automatically by an epoch-by-epoch approach in the SAC. Each activity occurring is added during an epoch duration in the manner of both "time" and "count" summations by the SAC. The time summation means the summation of the period of time that an activity occurred. The unit of "time" activity is in seconds. The "count" means the number of occurrences in which a burst of an activity separated from another burst. For instance, "alpha time at epoch 10" means the summation of the period that alpha waveform occurred during an epoch immediately after the 9th epoch. "Alpha count at epoch 10" means the number of occurrences in which isolated alpha activities took place. Hence, each epoch is assigned a

single stage score and several summed activities. Note that the temporal resolution of activity data available to the user is an epoch duration.

### 3. The Request Types of Sleep Data

The quantity questions of activity data can be done by one of the following functions:

- 1) count, e.g., how many subjects, the number of subjects,
- 2) sum, e.g., total alpha time,
- 3) average, e.g., average delta time,
- 4) maximum, e.g., maximum artifact time,
- 5) minimum, e.g., minimum delta time.

The user can specify the temporal relationship for quantity in two ways:

- 1) use of "epoch number," e.g., "Show total arousals from epoch 1 to epoch 120."
- 2) use of "hour" or "minute,"  
e.g. "Show total arousals for 2 hours from epoch 1."  
or "Show total arousals for first hour."

The user can use a phrase (e.g., greater than, bigger than, larger than, more than, smaller than, less than, etc) as well as a symbol (e.g., >=, <=, >, <, =) for the comparison of data.

The quantifier "each," "every," "any," or "all" can be interpreted by SEEGER as "each stage," "every subject," etc.

### 4. Sleep-Related Terms Understood by SEEGER and Sample Queries

The terms mentioned here are based on the Rechtschaffen & Kales Manual, Sleep Journal (Vol 2, Number 1, 1979), and the

SAC Manual (Microtronics). Refer to the SAC Manual for the definition of waveforms (i.e., waveform detection criteria). Some sample queries associated with the terms are illustrated below. The terms are classified into five categories:

### Subject Record

Record number, Name (first, last name), Sex, age.

1. Show me all subjects you have.
2. What is the record number of Jones?
3. Show the record number and recording date of Jones.
4. What is the subject name who is older than 30?
5. How many subjects are over 30?
6. Who is female subjects?
7. Show Jones's age.
8. How many subjects are in the database?
9. Tell me about subject names.
10. Give me all subject names.
11. Display subjects recorded on Jan, 1988.
12. List all subjects whose epoch length is 30 sec.
13. How long is the epoch in Jones's record?
14. What is the epoch duration of all subjects?

### Channel Recording

Recording date, Recording time, Montage, Channel name, Channel place, Channel number, EEG Channel, EOG Channel, EMG Channel.

1. Show me the montage of Jones.
2. How many channels does Jones have?
3. What kind of channel does he have?
4. Who has EMG channel ?
5. Show electrode position of Jones.
6. How many channels are used in the recording?

### Sleep Activities

Epoch number or Epoch#, Alpha time, Alpha count, Beta time, Beta count, Delta time, Delta count, Sigma time, Sigma count (Spindle), Theta time, Theta count, Artifact or Artifact time,

SEM (Slow Eye Movement) or SEM count, REMO (out-of-phase REM) or REMO count, REMI (in-phase REM) or REMI count, Arousal, K-complex.

1. Show alpha time, delta time, stage in Jones's record.
2. What is alpha count, beta count, and stage for Lisa.
3. Show epoch number when REMO count occurred.
4. Show epoch number, REMO, beta time for stage REM.
5. What is alpha time, stage from epoch 1 to epoch 20.
6. Show epoch number for spindle  $\geq 4$ .
7. Show total (or average) alpha time, beta time, delta time.
8. What is minimum (or maximum) alpha time?
9. What is total minutes of alpha time?
10. Show epochs which has more than 5 spindles.
11. Show me total alpha time, beta time for each sleep stage.

#### Sleep Stage

Sleep stage or stage or stage number, sleep stage 0 or stage 0 or stage awake, stage 1, stage 2, stage 3, stage 4, stage 5 or stage REM, REM sleep, NREM sleep, stage time, stage NREM.

1. Show total sleep stage time.
2. Show total sleep stage time for each sleep stage.
3. What is the sleep stage 1 time?
4. What is the time of sleep stage 1?
5. Show me epoch number, beta time for stage REM.

#### Sleep Parameters

Sleep time, Recording time or Time in bed, Sleep efficiency, Percent stage 0, Percent stage 1, Percent stage 2, Percent stage 3, Percent stage 4, Percent stage 5, Latency or Onset.

1. Who has stage REM time greater than 1 hour?
2. Show percent stage 1
3. Show sleep time of all subject you have.
4. Show me sleep latency of sleep stage 2.
5. Who has sleep stage 3 or 4?
6. What is the first sleep stage 2 minute for kimberley?

### 5. Getting Started

The GCLISP Developer software and the DBASE IV package have been installed on the system's hard disk. The SEEGER program (written in LISP) and the Sleep Database (running under DBASE IV) have also been installed in the hard disk. The dialogue between the user and SEEGER will be recorded in a log file.

To start the SEEGER program, follow these steps:

1. Turn on your computer, and the screen will display the prompt "C:\>."
2. Change to the directory where the GCLISP programs are located by entering the following command:

cd gclisp2 <ENTER>

3. Start the GCLISP environment by entering the following command:

gclisplm <ENTER>

At this point, you should see the Lisp prompt "\*."

4. Load SEEGER program by entering the following command:

(load 'main)

Note the parentheses in the input to the GCLISP environment above. This will take about 10 minutes.

5. Run the SEEGER program by typing the following command:

(seeger)

It will respond with the prompt "-->." The system is now ready to accept your request.

To exit from the SEEGER program, follow these instructions:

1. Exit from SEEGER and return to the GCLISP environment by typing the following command:

Ctrl-C (hold down the Ctrl key and hit C key)

2. Exit from the GCLISP environment and return to the DOS by typing the following command:

(exit)

#### 6. When the System Has Failed

Since SEEGER is in the experimental stage, it might fail at some point due to system errors. Two different types of failure might happen. One occurs during interpretation of the user's request due to LISP program errors. It happens before the system displays the paraphrase to the user. The other occurs during execution in the database due to database syntax errors. This happens after the paraphrase.

To exit from the system failure, follow one of two steps:

- 1) when the failure occurs during interpretation, type the following command:

Ctrl-C and then (seeger)

- 2) when the failure occurs while running the database, type the following command:

quit <ENTER>

APPENDIX B  
REQUESTS FROM USERS IN THE TEST OF SEEGER

IF: interpreted correctly at the first try  
IW: interpreted correctly with interaction  
II: interpreted incorrectly  
NI: could not be interpreted

Requests from the First User

IF -> show me

IW -> show me channel info

NI -> what can I do now

IF -> data available

IW -> show me delta time

NI -> go to theta

IW -> show me all theta times

IF -> tell me about the subjects name

IW -> what is the time in stage 2

IF -> tell me about subject names

IW -> show me all records available

IW -> show me sleep stages



Requests from the Second User

IF -> how many subjects are in the database  
NI -> how many nights are there  
NI -> is hykomen in the database  
IF -> give me names  
IF -> is the name hykomen present  
II -> when was hykomen recorded  
IF -> what date was hykomen recorded  
NI -> who was recorded 02/05/89  
NI -> who was recorded 02/05/1989  
IF -> who was recorded on 02/05/88  
IF -> what records are there for 02/05/89  
NI -> what was the name of the record taken on 02/05/89  
IF -> who was recorded on 02/05/89  
II -> give me the name of the record for 02/05/89  
IW -> how many arousals  
IF -> how many arousals did hykomen have on 02/05/89  
NI -> how long did hykomen sleep on the night of 02/05/89  
IF -> show me the montage of hykomen  
IF -> show me hykomen's arousals  
NI -> what is hykomens' record number for 02/05/89  
IF -> what is hykomen's record number for 02/05/89  
IF -> what is the total number of arousals for 35009n  
IW -> show total arousals  
IW -> how many artefacts were there  
IW -> what was the sleep efficiency

NI -> what was the sleep efficiency and total arousals and  
k-complex

NI -> what was the sleep efficiency and the total arousals

NI -> give sleep efficiency and give total sleep time and give  
arousals

IF -> give me sleep efficiency for record 35009n

IF -> give me record

NI -> give tst

NI -> give sws

IW -> give total sleep time

IW -> give total sleep time

IW -> give sleep efficiency and total sleep time

IW -> give total arousals

IF -> display sleep efficiency for record number 35009n

IW -> give sleep efficiency

Requests from the Third User

IF -> Give me a list of all female subjects please

IF -> I want to see a list of all subjects with rem latency  
greater than 10 minutes

IW -> show mwe a list of subjects

NI -> show me a sleep stage hypnogram of record 35009

NI -> list a summary of sleep stages fopr gree

NI -> show me the lowest emg level on kimberley

IW -> show me the greatest sigma count on kimberly

NI -> show me kimberleys record numbers

IW -> show me the record number for kimberley  
NI -> give me the ending epoch number of the longest  
contiguous block of stage 2 for 34989n  
II -> show me the longest stage 2 period from record 34989n  
NI -> for record 34989n show me the number of arousals and the  
total wake time and total artifact time  
NI -> list numbers of arousals and wake time and number of  
artifact for record 34989n  
IF -> list the average rem latency for all subjects  
IF -> list the average rem latency for all female subjects  
IF -> show me the channel names for record 34989n  
II -> where was the highest sem count for record 34989n  
IW -> list the epoch withj the greatest sem count for recrd  
34989n

#### Requests from the Fourth User

IF -> show me how many subjects  
NI -> which subject has the best looking legs  
IF -> who is the oldest subject  
NI -> list subjects as male or female  
IF -> which subjects are male  
IF -> how long was kimberley's longest rem period  
NI -> how long did it take kimberley to fall asleep  
NI -> how long before kimberley fell asleep  
IW -> what was sleep onset for kimberly  
IF -> how old is kimberley  
IW -> how much alpha did kimberley have in stage 1

APPENDIX C  
DEVELOPER'S GUIDE TO SEEGER

This guide introduces briefly the development environment for SEEGER and the sleep database (DB) system. SEEGER was developed under the Golden Common LISP (GCLISP) 286 Developer. The sleep DB and the DB menu-driven system were developed under the dBASE IV environment.

1. The SEEGER Program Development

The GCLISP 286 Developer provides software tools in order to support the LISP programming needs. The following two features will be used often:

- 1) The GCLISP interpreter: LISP code can be interpreted and the developer will be able to see the results of executing a portion of this code immediately, without having to go through a tedious compilation process. An error in a function or data objects could then be corrected, and the correction tested. The GMACS editor can be invoked here.
- 2) The GMACS editor: This is a full-screen display editor modeled after EMACS. The developer can create, load,

delete, evaluate, and save LISP code in a file or buffer. He can manipulate several files by saving in different buffers and can exit to the GCLISP interpreter and re-enter GMACS without disturbing the current environment.

The entire SEEGER program can be divided into the six main groups:

#### 1) Parsing

PARSE1.LSP: main function for SEEGER, initialization for parsing, printing for tracing  
 PARSE2.LSP: word tasks (loading a word into the working memory)  
 PARSE3.LSP: demon tasks (test, execute, or deactivate demons)  
 PARSE4.LSP: utilities used by demons, macro-definitions for word, phrase, and demons.  
 PARSE5.LSP: searching the working memory, pretty-printing the parsing output.  
 MORPH.LSP: analyzing word ending (morphological analysis)  
 DEMONS.LSP: demon definitions used in the LEX.LSP  
 DEMFUN.LSP: utilities called in the DEMONS.LSP  
 LEX.LSP: vocabulary definitions for SEEGER (words or phrases)  
 NG.LSP: parsing a noun group  
 PRONOUN.LSP: finding referent.

#### 2) Inferential Analysis

INFER.LSP: interpreting the meaning structure  
 RULE.LSP: rule control mechanism, rule definitions, rule sets

#### 3) Query Translation

TRANS.LSP: translating into VQL  
 AUGTR.LSP: augmenting VQL  
 DEDUCT.LSP: unification functions for pattern matching

#### 4) DB Mapping

MAPPING.LSP: mapping into SQL  
 DBMAP.LSP: DB mapping table  
 DBINTERF.LSP: utilities for creating SQL files, running DBASE IV, displaying DB execution results.

## 5) Interactive Dialogue

PARA.LSP: paraphrasing  
 CLARIFY.LSP: clarifying subject information, ambiguous  
                   concepts  
 KB.LSP: ambiguous concept table  
 SPELL.LSP: spelling correction

## 6) Other Utilities

MACRO.LSP: macro definitions  
 WINDOW.LSP: window display for input/output communication  
 ERROR.LSP: error handling  
 COORD.LSP: coordinate handling  
 ELLIPSE.LSP: ellipsis handling

2. The Sleep Database Development

Although dBASE IV provides many convenient features for managing data in DB files, only two language (i.e., SQL and dBASE) features are used for the sleep DB development. There are five files in the sleep DB: SUBJ\_REC.DBF (subject record), CHAN\_INF.DBF (channel recording), EP\_SUM.DBF (epoch summary), STG\_VAR.DBF (sleep stage summary), and PARAM.DBF (night summary). These files are created and edited by the menu-driven system.

The menu-driven system was written in dBASE language. There are five files:

SLEEPDB.PRG: main menu for selecting submenu.  
 SUBJ\_REC.PRG: subject record menu for editing data  
 CHAN\_INF.PRG: channel recording information menu  
 EP\_SUM.PRG: epoch summary menu  
 LIBRARY.PRG: utilities for examining, adding, deleting,  
                   counting, grouping, and searching records

The epoch summary data are provided with a file from the SAC (Sleep Analyzing Computer) system. This file is processed

into an input file form which can be added to the existing DB file. This is done by the SUMMARY.COM command written in C. The epoch summary data are further reduced into an appropriate form for a sleep stage summary and a night summary by the PARAM.COM command written in C. These input files are automatically added to the existing DB files when the user enters a file name containing a new subject epoch summary data. The input data of a subject record and channel recording should be entered by the user through the menu-driven screen.

To bring the subject information in the DB into the system, SUBJECT.PRS in SQL is executed under dBASE IV during the SEEGER initialization. This result is stored in SUBJECT.TXT and then read into the system.

The SQL expression generated from SEEGER is stored in SQL-F.PRS and then executed under dBASE IV. This result is stored in SQL-F.TXT and then presented to the user.

## REFERENCES

- Birnbaum, L. A. (1986). Integrated processing in planning and understanding, Ph.D. Dissertation, Yale University.
- Birnbaum, L. A. and M. Selfridge (1981). Conceptual analysis of natural language, In R. Schank and C. Riesbeck, (Eds.), Inside computer understanding: Five programs plus miniatures, Lawrence Erlbaum, Hillsdale, NJ, 318-353.
- Burton, R. R. (1976). Semantic grammar: An engineering technique for constructing natural language understanding systems, Report 3453, ICAI Report 3, Bolt, Beranek and Newman, Cambridge, MA.
- Carbonell, J. G. (1985). The role of user modeling in natural language interface design, In S. J. Andriole, (Ed.), Application in artificial intelligence, Petrocelli Books, Princeton, NJ, 213-226.
- Chang, T. G. (1987). Development of an expert system for multichannel EEG signal analysis, Ph.D. Dissertation, University of Florida.
- Charniak, E., C. K. Riesbeck, D. V. McDermott, and J. R. Meehan (1987). Artificial intelligence programming, Lawrence Erlbaum, Hillsdale, NJ.
- Chomsky, N. (1965). Aspects of the theory of syntax, MIT Press, Cambridge, MA.
- Clocksin, W.F. and C. S. Mellish (1984). Programming in Prolog, Second edition, Spinger-Verlag, New York.
- Codd, E. F. (1978). How about recently? (English dialog with relational data bases using RENDEZVOUS version 1), In B. Shneiderman, Databases: Improving usability and responsiveness, Academic Press, New York, 3-28.
- Cullingford, R. E. (1986). Natural language processing: A knowledge-engineering approach, Rowman & Littlefield, Totowa, NJ.



Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors, Comm. ACM, Vol. 7, No. 3, March, 171-176.

Date, C. J. (1896). An introduction to database systems, Vol. 1, Fourth edition, Addison-Wesley, Reading, MA.

Dyer, M. G. (1983). In-depth understanding: A computer model of integrated processing for narrative comprehension, MIT Press, Cambridge, MA.

Finin, T. W., A. K. Joshi, and B. L. Webber (1986). Natural language interactions with artificial experts, Proceedings of the IEEE, Vol. 74, No. 7, July, 921-938.

Gershman, A. V. (1979). Knowledge-based parsing, Ph.D. Dissertation, Yale University.

Ginsparg, J. (1983). A robust portable natural language database interface, Proceedings of the Conference on Applied Natural Language Processing, ACL, Santa Monica, CA, 25-30.

Grosz, B. J. (1982). Transportable natural-language interfaces: problems and techniques, Proceedings of the Conference in the 20th Annual Meeting of the Association for Computational Linguistics, Toronto, Ontario, 46-50.

Grosz, B. J. (1983). TEAM: a transportable natural language interface system, Proceedings of the Conference on Applied Natural Language Processing, ACL, Santa Monica, CA, 39-45.

Harris, L. R. (1977). User oriented data base query with the Robot natural language query system, International Journal of Man-machine Studies, Vol. 9, 697-713.

Hayes, P. J. and J. G. Carbonell (1981). Multi-strategy construction-specific parsing for flexible data base query and update, Proceedings of the 7th International Joint Conference on Artificial Intelligence, Vancouver, British Columbia, Canada, August, 432-439.

Hendrix, G. G. (1977). The LIFER manual: A guide to building practical natural language interfaces, AI Center Technical Note 138, SRI International, Menlo Park, CA.

Hendrix, G. G., E. D. Sacerdoti, D. Sagalowicz, and J. Slocum (1978). Developing a natural language interface to complex data, ACM Trans. on Database Systems, Vol. 3, No. 2, June, 105-147.

Jakobson, G., C. Lafond, E. Nyberg, and G. Piatetsky-Shapiro (1986). An intelligent database assistant, IEEE Expert, Vol. 1, No. 2, Summer, 65-79.

Kaplan, S. J. (1979). Cooperative responses from a portable natural language data base query system, Ph.D. Thesis, University of Pennsylvania.

Kim, C., J. C. Principe, and J. R. Smith (1988). SEEGER: A natural language interface to a sleep EEG/EOG database, Proceedings of the ISMM International Symposium Mini and Microcomputers, Miami Beach, Florida, Dec., 104-107.

Lehnert, W. G. and S. P. Shwartz (1982). Natural language data base access with Pearl, Proceedings of the 9th International Conference on Computational Linguistics, Prague, Czechoslovakia, 121-128.

Lehnert, W. G. and S. P. Shwartz (1983). EXPLORER: a natural language processing system for oil exploration, Proceedings of the Conference on Applied Natural Language Processing, ACL, Santa Monica, CA, 69-72.

Marcus, M. (1980). A theory of syntactic recognition for natural language, MIT Press, Cambridge, MA.

Martin, J. (1985). Fourth-generation languages, Prentice-Hall, Inc., Englewood Cliffs, NJ., 213-225.

Mueckstein, E. M. (1983). Q-trans: query translation into English, Proceedings of the 8th International Joint Conference on Artificial Intelligence, Karlsruhe, West Germany, Aug., 660-662.

Perrault, C. R. and B. J. Grosz (1986). Natural-language interfaces, Annual Review of Computer Science, Vol 1, 47-82.

Principe, J. C. and J. R. Smith (1986). SAMICOS: A sleep analyzing microcomputer system, IEEE Trans. Biomed. Eng., Vol. 33, No. 10, Oct., 550-559.

Rechtschaffen, A. and A. Kales (1968). A manual of standardized terminology, techniques and scoring system for sleep stages of human subjects, Public Health Service, U.S. Government Printing Office, Washington, D.C.

Riesbeck, C. K. (1975). Conceptual analysis, In R. Schank, (Ed.), Conceptual information processing, North-Holland, Amsterdam, 83-156.

Riesbeck, C. K. (1978). An expectation-driven production system for natural language understanding, In D.A. Waterman and F. Hayes-Roth, (Eds.), Pattern-directed inference systems, Academic Press, New York, 399-413.

- Riesbeck, C. K. and Martin, C. (1986). Direct memory access parsing, In J. Kolodner and C. Riesbeck, (Eds.), Experience, memory, and reasoning, Lawrence Erlbaum, Hillsdale, NJ, 209-226.
- Scha, R. J. H. (1982). English words and data bases: how to bridge gap, Proceedings of the Conference in the 20th Annual Meeting of the Association for Computational Linguistics, Toronto, Ontario, 57-59.
- Schank, R. (1973). Identification of conceptualizations underlying natural language, In R. Schank and K. Colby, (Eds.), Computer models of thought and language, W. H. Freeman, San Francisco, 187-247.
- Schank, R. (1975). Conceptual information processing, R. Schank, (Ed.), Elsevier, North-Holland, New York.
- Schank, R. and R. Abelson (1977). Scripts, plans, goals, and understanding, Lawrence Erlbaum, Hillsdale, NJ.
- Schank, R. and S. P. Schwartz (1985). The role of knowledge-engineering in natural language systems, In S. J. Andriole, (Ed.), Application in artificial intelligence, Petrolli Books, Princeton, NJ, 193-211.
- Shwartz, S. P. (1982). Problems with domain-independent natural language database access systems, Proceedings of the Conference in the 20th Annual Meeting of the Association for Computational Linguistics, Toronto, Ontario, 60-62.
- Shwartz, S. P. (1987). Applied natural language processing, Petrocelli Books Inc., Princeton, NJ.
- Smith, J. R. (1986). Automated analysis of sleep EEG data, In L. D. Silva, S. V. Leeuwen, and A. Remond, (Eds.), Handbook of electroencephalography and clinical neurophysiology, Vol. 2, Elsevier Science Publishing Co., Inc., New York, 131-147.
- Templeton, M. and J. Burger (1983). Problems in natural language interface to DBMS with examples from EUFID, Proceedings of the conference on applied natural language processing, ACL, Santa Monica, CA, 3-16.
- Tennant, H. R. (1981). Evaluation of natural language processors, Ph.D. Dissertation, University of Illinois at Urbana-Champaign.
- Waltz, D. L. (1978). An English language question answering system for a large relational database, Comm. ACM, Vol. 21, No. 7, July, 526-539.

Warren, D. H. D. (1982). Issues in natural language access to databases from a logic programming perspective, Proceedings of the Conference in the 20th Annual Meeting of the Association for Computational Linguistics, Toronto, Ontario, 63-66.

Warren, D. H. D. and F. C. N. Pereira (1982). An efficient easily adaptable system for interpreting natural language queries, American journal of computational linguistics, Vol. 8, No. 3, 110-122.

Winograd, T. (1972). Understanding natural language, Academic Press, New York.

Winograd, T. (1983). Language as a cognitive process: Syntax, Vol. 1, Addison-Wesley, Reading, MA.

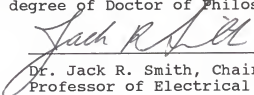
Woods, W. A. (1970). Transition network grammars for natural language analysis, Comm. ACM, Vol. 3, No. 10, Oct., 591-606.

Woods, W. A., R. M. Kaplan, and B. Nash-Webber (1972). The Lunar sciences natural language information system: final report, Report 2378, Bolt, Beranek and Newman, Cambridge, MA.

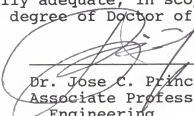
#### BIOGRAPHICAL SKETCH

Chongtai Kim was born on September 19, 1954, in Daejon, Korea. He received his B.S degree in electrical engineering from Seoul National University in 1977. After graduation, he worked for Agency for Defense Development for six years. Since August 1983, he has been studying electrical engineering at the University of Florida. He received the degree of Master of Science in August 1985. Since then, he has been pursuing the degree of Doctor of Philosophy at the University of Florida.

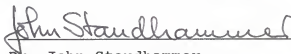
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

  
\_\_\_\_\_  
Dr. Jack R. Smith, Chairman  
Professor of Electrical Engineering

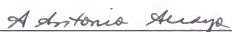
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

  
\_\_\_\_\_  
Dr. Jose C. Principe, Cochairman  
Associate Professor of Electrical  
Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

  
\_\_\_\_\_  
Dr. John Staudhammer  
Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

  
\_\_\_\_\_  
Dr. A. Antonio Arroyo  
Associate Professor of Electrical  
Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



---

Dr. Sharma Chakravarthi  
Associate Professor of Computer and  
Information Science

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

May 1990



---

Winfred M. Phillips  
Dean, College of Engineering

---

Madelyn M. Lockhart  
Dean, Graduate School