PROBLEMS IN NATURAL-LANGUAGE INTERFACE TO DBMS WITH EXAMPLES FROM EUFID

Marjorie Templeton
John Burger
System Development Corporation
Santa Monica, California

ABSTRACT

For five years the End-User Friendly Interface to Data management (EUFID) project team at System Development Corporation worked on the design and implementation of a Natural-Language Interface (NLI) system that was to be independent of both the application and the database management system. In this paper we describe application, natural-language and database management problems involved in NLI development, with specific reference to the EUFID system as an example.

I INTRODUCTION

From 1976 to 1981 SDC was involved in the development of the End-User Friendly Interface to Data management (EUFID) system, a natural-language interface (NLI) that is designed to be independent of both the application and the underlying database management system (DBMS). [TEMP79, TEMP80, BURG80, BURG82]. The EUFID system permits users to communicate with database management systems in natural English rather than formal query languages. It is assumed that the application domain is well defined and bounded, that users share a common language to address the application, and that users may have little experience with computers or DBMSs but are competent in the application area.

At least three broad categories of issues had to be addressed during EUFID development, and it is apparent that they are common to any general natural-language interface to database management systems.

The first category involves the application: how to characterize the requirements of the human-machine dialogue and interaction, capture that information efficiently, formalize the information and incorporate that knowledge into a framework that can be used by the system. The major problems in this area are knowledge acquisition and representation. For many NLI systems, bringing up a new application requires extensive effort by system designers with cooperation from a representative set of end-users. Tools that could assist in automating this process are badly needed.

The second set of issues involves language processing techniques: how to assign constituent structure and interpretation to queries using robust and general methods that allow extension to additional lexical items, sentence types and semantic relationships. Some NLI systems distinguish the assignment of syntactic structure, or parsing, from the interpretation. Other systems, including EUFID, combine information about constituent and semantic structure into an integrated semantic grammar.

The third class involves database issues: how to actually perform the intent of the natural-language question by formulating the correct structured query and efficiently navigating through the database to retrieve the right answer. This involves a thorough understanding of the DBMS structure underlying the application, the operations and functions the query language supports, and the nature and volatility of the database.

Obviously issues in these three areas are related, and the knowledge needed to deal with them may be distributed throughout a natural-language interface system. The purpose of this paper is to show how such issues might be addressed in NLI development, with illustrations from EUFID.

The next section includes a brief review of related work, and an overview of the EUFID system. The third section describes the goals that EUFID achieved, and section four discusses in detail some of the major application, language, and database problems that arose. Section five suggests guidelines for determining whether an application is an appropriate target for a natural-language interface.

II BACKGROUND

Over the past two decades a considerable amount of work has gone into the development of natural-language systems. Early developments were in the areas of text processing, syntactic parsing techniques, machine translation, and early attempts at English-language question answering systems. Several early question-answering experiments are reviewed by R. F. Simmons in [SIMM65]. Waltz has edited a collection of short papers on topics related to natural-language and artificial intelligence in a survey of NLI research [WALT77]. A survey of NLIs and evaluation of several systems with respect to their applicability to command and control environments can be found in [OSI79].

A. RELATED WORK

While few NLIs have reached the commercial marketplace, many systems have contributed to advancing the state of the art. Several representative systems and the problems they addressed are described in this section.

1. CONVERSE [KELL71] used formal syntactic analysis to generate surface- and deep-structure parsings together with formal semantic transformation rules to produce queries for a built-in relational DBMS. It was written in SDC LISP and ran on IBM 370 computers. Started in 1968, it was one of the first natural-language processors to be built for the purpose of querying a separate data management system.

2. LADDER [HEND77] was designed to access large distributed databases. It is implemented in INTERLISP, runs on a PDP-10, and can interface to different DBMSs with proper configuration. It uses a semantic grammar and, like EUFID and most NLIs, a different grammar must be defined for each application.

3. The Lunar Rocks system LSNLIS [WOOD72] was the first to use the Augmented Transition Network (ATN) grammar. Written in LISP, it transformed formally parsed questions into representations of the first-order predicate calculus for deductive processing against a built-in DBMS.

4. PHLIQA1 [SCHA77] uses a syntactic parser which runs as a separate pass from the semantic understanding passes. This system is mainly involved with problems of semantics and has three separate layers of semantic understanding. The layers are called "English Formal Language", "World Model Language", and "Data Base Language" and appear to correspond roughly to the "external", "conceptual", and "internal" views of data as described by C. J. Date [DATE77]. PHLIQA1 can interface to a variety of database structures and DBMSs.

5. The Programmed LANguage-based Enquiry System (PLANES) [WALT78] uses an ATN based parser and a semantic case frame analysis to understand questions. Case frames are used to handle pronominal and elliptical reference and to generate responses to clarify partially interpreted questions.

6. REL [THOM69], initially written entirely in assembler code for an IBM360, has been in continuous development since 1967. REL allows a user to make interactive extensions to the grammar and semantics of the system. It uses a formal grammar expressed as a set of general re-write rules with semantic transformations attached to each rule. Answers are obtained from a built-in database.

7. RENDEZVOUS [CODD74] addresses the problem of certainty regarding the machine's understanding of the user's question. It engages the user in dialogue to specify and disambiguate the question and will not route the formal query to the relational DBMS until the user is satisfied with the machine's interpretation.

8. ROBOT [HARR78] is one of the few NLI systems currently available on the commercial market. It is the basis for Cullinane's OnLine English [CULL80] and Artificial Intelligence Corporation's Intellect [EDP82]. It uses an extracted version of the database for lexical data to assist the ATN parser.

9. TORUS [MYLO76], like RENDEZVOUS, engages the user in a dialogue to specify and disambiguate the user's question. It is a research oriented system looking at the problems of knowledge representation, and some effort has been spent on the understanding of text as well as questions.

B. OVERVIEW OF EUFID

EUFID is a general purpose natural-language front-end for database management. The original design goals for EUFID were:

- to be application independent. This means that the program must be table driven. The tables contain the dictionary and semantic information and are loaded with application-specific data. It was desired that the tables could be constructed by someone other than the EUFID staff, so that users could build new applications on their own.

- to be database independent. This means that the organization of the data in the database must be representable in tables that drive the query generator.* A database reorganization that does not change the semantics of the application should be transparent to the user.

- to be DBMS independent. This means that it must be able to generate requests to different DBMSs in the DBMS's query language and that the interface of EUFID to a different DBMS should not require changes to the NLI modules. Transferring the same database with the same semantic content to another DBMS should be transparent to the natural-language users.

- to run on a mini-computer that might possibly be different from the computer with the DBMS.

- to have a fast response time, even when the question cannot be interpreted. This means it must be able quickly to recognize unanalyzable constructs.

- to handle nonstandard or poorly-formed (but, nevertheless, meaningful) questions.

- to be portable to various machines. This means that the system had to be written in a high level language; initially a customer required code to be written in FORTRAN, later we were able to use the "C" programming language.

- to support different views of the data for security purposes.

The design which met these requirements is a modular system which uses an Intermediate Language (IL) as the output of the natural-language analysis system [BURG82]. This language represents, in many ways, the union of the capabilities of many "target" DBMS query languages.

The EUFID system consists of three major modules, not counting the DBMS (see Figure 1). The analyzer (parser) module is table driven. It is necessary only to properly build and load the tables to interface EUFID to a new application. Mapping a question from its dictionary (user) representation to DBMS representation is handled by mapping functions contained in a table and applied by a separate module, the "mapper". Each content (application dependent) word in the dictionary has one or more mapping functions defined for it. A final stage of the mapper is a query-language generator containing the syntax of IL. This stage writes a query in IL using the group/field names found by the mapper to represent the user's concepts and the structural relationships between them. This design satisfies the requirement of application independence.

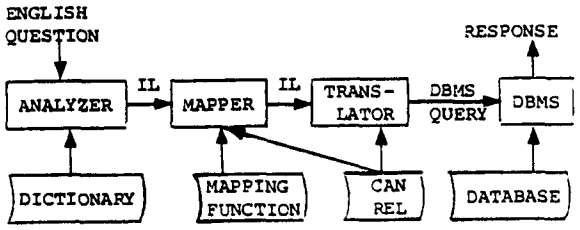

Figure 1: EUFID Block Diagram

For each different DBMS used by a EUFID application, a "translator" module needs to be written to convert a query in IL to the equivalent in the DBMS query language. This design satisfies the requirement of DBMS independence.

Other modules are the system controller, a "help" module, and a "synonym editor". An "Application Definition Module" is used off-line to assist in the creation of the run-time application description tables.

* We make a technical distinction between the words "question" and "query". A question is any string entered by the user to the EUFID analyzer, regardless of the terminating punctuation. This is consistent with the design since EUFID treats all input as a request for information. A query is a formal representation of a question in either the EUFID intermediate language IL, or in the formal query language of a DBMS.

The following subsections describe each of the modules of the EUFID system, and give our motivation for design.

1. Application Definitions

Bringing up a new application is a long and complex process. The database definition must be transmitted to EUFID. A large corpus of "typical" user questions must be collected from a representative set of users and from these the dictionary and mapping tables are designed. A "semantic graph" is defined for the application. This graph is implicitly realized in the dictionary where the nodes of the graph are the definitions of English content words and the connectivity of the graph is implied by the case-structure relationships defined for the nodes.

All dictionary and mapping-function data are then entered into computer files which are processed by the Application Definition Module (ADM) to produce the run-time tables. These final tables are complex structures of pointers, character strings, and index tables, designed to decrease access time to the information required by the analyzer and mapper modules.

The ADM, typically, needs to be run several times to "debug" the tables. EUFID interfaces to three applications currently exist, and building tables for each new application took less time than the previous one, but it still requires several staff-months to bring up a new application.

a. User-View Representation

All information on the user's view of the database is kept in the dictionary. The dictionary consists of two kinds of words and definitions. Function words, such as prepositions and conjunctions, are pre-stored in each application's dictionary and are used by the analyzer for direction on how to connect the semantic-graph nodes during analysis. Content words are application dependent. The definitions of content words are semantic-graph nodes. The connectivity of the graph is indicated by semantic case slots and pointers contained in the nodes. A form of semantic-case is used to indicate the attributes of an entity (e.g., adjectives, prepositional phrases, and other modifiers of a noun).

b. Mapping Functions

The list of mapping functions is derived from the dictionary. Every possible connection of every node has to be considered. Frequently, design considerations in the mapping-function list necessitate going back and modifying the content of the dictionary. This is an example of the overlap of the linguistic and database issues in assigning an interpretation to a question.

c. Database Representation

The structure of the data in the user's database is represented in two tables, called the CAN (for canonical) and REL (for relationships) tables. Taking advantage of the fact that any database can be represented in relational form, EUFID lists each database group as if it were a relation. Group-to-group linkage (represented in the REL table) is dealt with as if a join* were necessary to implement the link. For hierarchical and network DBMSs the join will not be needed: the link is "wired in" to the database structure. EUFID nevertheless assumes a join mainly in order to facilitate the writing of group-to-group links in IL, which is a relational language. The CAN table includes database-specific information for each field (attribute) of each group (relation), such as field name, containing group, name of domain from which attributed gets its values, and a pointer to a set of conversion functions for numeric values which can be be used to convert from one unit of measure to another (e.g., feet to meters).

These data are used by the run-time modules which map and translate the tree-structured output of the analyzer to IL on the actual group/field names of the database, and then to the language of the DBMS. These modules are discussed in the next sections.

2. The EUFID Analyzer

The current version of the EUFID analyzer employs a variant of the Cocke-Kasami-Younger algorithm for parsing its input. This classical nonpredictive bottom-up algorithm has been used in a family of "chart parsers" developed by Kay, Earley, and others [AHO72]. The main features of these parsers are: (1) They use arbitrary context-free grammars. There are no restrictions on rules which have left-recursion or other characteristics which sometimes cause difficulty. (2) They produce all possible parses of a given input string. The grammars they use may be ambiguous at either the nonterminal- or terminal-symbol levels. In natural-language processing, this allows for a precise representation of

* The term "join" refers to a composite operation between two relations in a relational DBMS.

both the syntactic and lexical ambiguities which may be present in an input sentence. (3) They provide partial parses of the input. Each non-terminal symbol derives some input substring. Even if no such substring spans the entire sentence, i.e., no complete parse is achieved, analyses of various regions of the sentence are available. (4) They are conceptually straightforward and easy to implement. The speed and storage considerations which have kept such parsers from being widely used in compilers are less relevant in the analysis of short strings such as queries to a DBMS.

The grammar used by the EUFID parser is essentially semantic. The symbols of the grammar represent the concepts underlying lexical items, and the rules specify the ways in which these concepts can be combined. More specifically, the concepts are organized into a case system. Each rule states that a given pair of constituents can be linked if the conceptual head of one constituent fills a case on the conceptual head of the other. A degree of context sensitivity is achieved by attaching predicates to the rules. These predicates block application of the rules unless certain (usually syntactic) conditions hold true. The parser uses syntactic information only "on demand", that is, only when such information is necessary to resolve semantic ambiguities. This adds to its coverage and robustness, and makes it relatively insensitive to the phrasing variations which must be explicitly accounted for in many other systems.

3. Mapping

The mapper module converts the output of the analyzer to input for the translator module. Analyzer output is a tree structure where the nodes are semantic-graph nodes corresponding to the content words in the user's question and obtained from the dictionary.

Input to the translator module is a string in the syntax of IL which contains the names of actual groups and fields in the database. The mapping algorithm, thus, has to make several levels of conversion simultaneously:

- it must convert a tree structure into a linear string of tokens,
- it must convert semantic-graph nodes into database group- and field-names, and
- it must convert the connectivity of the tree (representing concept-to-concept linkage in English) into the (frequently very different) group-to-field and group-to-group connections of the database.

The mapper makes use of a table of mapping functions. The table contains at least one mapping function for every content word in the dictionary. The analyzer's tree is traversed bottom up, applying mapping functions to each node on the way. Mapping functions are context sensitive with respect to those nodes below it in the tree: nodes that have already been mapped. A new tree is gradually formed and connected this way. Mapping functions may indicate that the map of a semantic-graph node is a database node (that is, a group or field name), or a pre-connected sub-tree of database nodes. The mapping function may also indicate removal of a database node or modification to the existing structure of the tree being constructed.

The new tree is created in terms of the database groups and fields and its structure reflects the connectivity of the database. A final stage of the mapper traverses this new tree and generates the IL statement of the query using a table of the syntax and keywords of IL and the database names from the tree.

An alternative method of mapping that is now being investigated involves breaking the process into two basic parts. The first step would be to map the tree output of the analyzer to an IL query on what C. J. Date calls the "conceptual schema" of the database [DATE77]. A second step would take this IL input and re-arrange the schema connectivity (and names of groups and fields) from that of the conceptual schema to that of the actual target database, generating another IL query as input to the current translators.

4. Translating

The final run-time module in EUFID is a syntax translator that converts IL to the actual DBMS query language. If necessary, the translator can also add access-path information related to database search. Currently, two translators have been written. One converts IL to QUEL, a relatively simple conversion into the language of the relational database management system INGRES [STON76]. The other translator converts IL into the query language of the World-Wide Data Management System (WWDMS) [HONE76] used by the Department of Defense, and also handles additional access path information. This translator was quite difficult to design and build because of the highly procedural nature of the WWDMS query system.

The output of a translator is sent to the appropriate DBMS. In the EUFID system running at SDC, a QUEL query is submitted directly to INGRES running on the same PDP-11/70 as EUFID. For testing purposes, queries generated by the WWDMS translator were transmitted from a PDP-11/70 to a Honeywell H6000 with a WWDMS database.

5. Application Description

EUFID runs on three different application databases. The METRO application involves monitoring of shipping transactions between companies in a city called "Metropolis". There are ten companies located in any one of three neighborhoods. Each company rents warehouse space for shipping/receiving transactions, and has local offices which receive goods. The data is organized relationally using the INGRES database management system. That means that there are no navigational links stored in the records (called "relations") and there is no predefined "root" to the database structure. Access may be made from any relation to any other relation as long as there is a field in each of the two relations which has the same "domain" (set of values).

AIREP (ADP Incident REPorting) is a network database, implemented in WWDMS. It contains reports about hardware and software failures and resolution of the problems in a large computer system. Active problems are maintained in an active file and old, solved problems are moved to an historical file. If a problem is reported more than once, an abbreviated record is made for the additional report, called the "duplicate incident" record. This means that there are four basic type of report: active incidents, duplicate incidents, historical incidents, and historical duplicate incidents. In addition, there are records about sites, problems, and solutions.

The APPLICANT database is a relational database implemented in INGRES that contains information about job applicants and their backgrounds. The central entity is the "applicant", while other relations describe the applicant's specialties, education, previous employment, computer experience, and interviews.

Each database has different features that may present problems for a natural-language interface but which are typical of 'real-world' applications. METRO has relatively few entities but has complex relationships among them. APPLICANT has many updates and many different values, some coming from open-ended domains. AIREP has a network database structure and contains the same data structure in four different files.

III LEVEL OF SUCCESS

Most of the EUFID design goals were actually met. EUFID runs on a minicomputer, a DEC PDP 11/70. It is application, database, and DBMS independent. A typical question is analyzed, mapped and translated in five to fifteen seconds even with grammatically incorrect input.

The analyzer contains a good spelling corrector and a good morphology algorithm that strips inflectional endings so that all inflected forms of words need not be stored explicitly. A "synonym editor" permits the user to replace any word or string of words in the dicionary with another word or string, to accommodate personal jargon and expressability. A "Concept Graph Editor" allows a database administrator to modify tables and define user profiles so that different users may have limited views of the data for security purposes.

The analysis strategy, based on a semantic grammar, permits easy and natural paraphrase recognition, although there are linguistic constructs it cannot handle. These are discussed below.

An English word may have more than one definition without complicating the analysis strategy. For example, "ship" as a vessel and as a verb meaning "to send" can be defined in the same dictionary. Words used as database values, such as names, may also have multiple definitions, e.g., "New York" used as the name of both a city and a state.

The mapper, despite its many limitations, can correctly map almost all trees output by the analyzer. It is able to handle English conjunctions, mapping them appropriately to logical ANDs or ORs, and understanding that some "ands" may need to be interpreted as OR and vice-versa under certain circumstances. It is able to generate calls on DBMS calculations (e.g., average) and user-defined functions (e.g., marine great-circle distance) if the user-function exists and is supported by the DBMS.

Questions involving time are interpreted in a reasonable way. Functions are defined for "between" and "during" in the METRO application. The AIREP application allows time comparisons such as "What system was running when incident J123 occurred" which require a test to see if a point in time is within an interval.

The mapper can translate "user values" (e.g., "Russian") to database values (e.g., "USSR"), and convert one unit of measure (e.g., feet) to another (e.g., meters).

EUFID can interface to very complex relational and CODASYL-type databases having difficult navigation and parallel structures. In the AIREP application a consistent WWDMS navigational methodology is used to access non-key records. The system can also map to the parallel, but not identical, structures for duplicate and historical incidents.

In the INGRES applications, EUFID is able to use and correctly map to "relationship relations" which relate two or more other relations. For example, the METRO relation "cw" contains a company name, a warehouse name, and a date. This represents the initial business contact. A user might ask, "When did Colonial start to do business with Superior?" or "When did business begin between Colonial and Superior?", either of which must join both the company ("c") and the warehouse ("w") relations to the "cw" relation.

The system control module keeps a journal of all user-system interaction together with internal module-to-module data such as the IL for the user's question and the generated DBMS query. The system also employs a very effective HELP module which, under certain circumstances, is context sensitive to the problem affecting the user.

IV PROBLEMS

This section describes problems associated with EUFID development that appear to be common to natural-language interfaces to database management systems. They are loosely classified into the major areas of application, language and database management issues, although there may be overlap. Criteria for evaluating whether an application is appropriate for a natural-language front-end are also described.

A. APPLICATION DEFINITION PROBLEMS

The primary issue in this area is concerned with problems of defining, creating, and bringing up the necessary data for a new application. The discussion points out the difficulties associated with systematic knowledge acquisition.

1. User Model

A single database may be used by different groups of users for different purposes. For example, some users of the APPLICANT database may wish to fill a specific job opening while others may collect statistics on types of applicants. The language used for these two functions can be quite different, and it is necessary to have extensive interaction with cooperative users in order to characterize the kinds of dialogues they will have with the system.

Not only must representative language protocols be collected, but desired responses must be understood. For example, to answer a question such as "What is the status of our forces in Europe", the system must know whether 'our' refers to U.S. or NATO or some other unit.

The importance of this interaction between potential users and system developers should not be underestimated, as it is the basis for defining much of the knowledge base needed by the system, and may also be the basis for eventual user acceptance or rejection of the NLI system.

2. Value Recognition

A "value" is a specific datum stored in the database, and is the smallest piece of data obtainable as the result of a database query. For example, in response to the question "What companies in North Hills shipped light freight to Superior?" the METRO DBMS returns two values: "Colonial" and "Supreme". Values can also be used in a query to qualify or select certain records for output, e.g., in the above question "North Hills" and "Superior" are values that must be represented in the query to the DBMS. As long as the alphanumeric values used in a particular database field are the same as words in the English questions, there are no difficult problems involved in recognizing values as selectors in a query.

There are three basic ways to recognize these value words in a question. They can be explicitly listed in the dictionary, recognized by a pattern or context, or found in the database itself.

If the value words are stored in the dictionary, they can be subject to spelling correction because the spelling corrector uses the dictionary to locate words which are a close match to unrecognized words in a question. This means, though, that all possible values and variant legitimate spellings of values for a concept must be put either into the dictionary or into the synonym list. This is reasonable for concepts which have a small and controlled set of values* such as the names of the

* A set of values is called a "domain".

companies in METRO, but may become unwieldy for large sets of values.

If a value can be recognized by a pattern, it is not necessary to itemize all instances in the dictionary. For example, a date may be entered as "yy/mm/dd" so that any input matching the pattern "nn/nn/nn" is recognized as a date. This is the approach used for dates and for names of applicants in the APPLICANT database, where names of people match the pattern "I.I.Lastname".

In another approach, OnLine English [CULL80] and Intellect [HARR78, EDP82] (two variations of ROBOT) used the database to recognize values. This is a satisfactory solution if the database is small or if the small number of different values is stored in an index accessible to the NLI, and if the values in the database are suitable for use in English questions.

Each of these solutions has disadvantages. If values are stored in the dictionary there may be many different ways to spell each particular value. For example, the company name for "System Development Corporation" may also be given as "S.D.C.", "S D C", or "System Development Corp". While each different spelling could be entered as a synonym for the "correct" spelling in the database, this would result in an enormous proliferation of the dictionary entries and problems with concurrency control between the updates directed to the data management system and the updates to the dictionary. A creative solution might be to define rules for synonym generation and apply them to database updates.

A somewhat different example is from the APPLICANT application which has many open ended domains, such as names of applicants and previous employers. In this case, the application designer may have to treat certain fields as "retrieve-only", meaning that the data can be asked for but not used as a selection criterion. A database with a large number of retrieve-only fields may be a poor candidate for an NLI.

Patterns can be used only if they can be enforced, and probably few values really fit the patterns nicely. Proper names are a poor choice for patterns because of variations such as middle initial or title such as "Dr." or "Jr.". Also, spelling correction cannot be performed unless the value is stored in the dictionary.

Finally, the solution of using the database itself to recognize values is unsatisfactory to a general NLI for anything other than trivial databases, unless an inverted index of values is easily accessible. There are the problems of spelling correction and synonyms for database values, the inefficiency involved in accessing the DBMS for every unrecognized word, and the difficulty of knowing which fields in the database to search.

3. Semantic Variation By Value

Databases are generally designed with a minimum number of different record types. When there are entities which are similar, but possibly have a small number of attributes which are not shared, the entities will be stored in the same record type with null values for the attributes that do not apply. The user, in his questions, may view these similar entities as very different entities and talk about them differently.

We did not encounter the problem with METRO or AIREP. For example, in METRO, the user asks the same type of questions about the company named "Colonial" as about the company named "Supreme". In APPLICANT, however, each applicant has a set of "specialties" such as "computer programmer", "accounting clerk", or "gardener". These are all stored as values of the specialty field in the database. Unfortunately, in this case different specialties evoke completely different concepts to the end user. The user may ask questions such as, "What programmers know COBOL?", "Who can program in COBOL?", and "How many applicants with a specialty in computer programming applied in 1982?". Notice the new nouns and verbs that are introduced by this specialty name.

A value domain such as specialties should be handled with an ISA hierarchy. Each different type of specialty such as gardener or programmer could have a different concept that is a subset of the concept "specialty". Some questions could be asked about all specialties and others could be directed only to certain subconcepts. However, there is no ISA hierarchy in EUFID, and it would have been inefficient to treat each specialty and subspecialty as a separate concept since there are 30 specialties and 196 subspecialties. Therefore, we required the users to know the exact values, to know which values are for specialties and which are for subspecialties, and to ask questions using the values only as nouns. This is not "user friendly".

Even if it were possible to build a different concept for each different skill, there is an update problem. When a new value is added to a value domain where there are uniform semantics (as in adding a new company name in METRO), the new value is simply attached to the existing concept. When the new value has different semantics, the newly associated concepts, nouns, and verbs cannot be added automatically. If the NLI supports an ISA hierarchy, someone will need to categorize the new value and add a new node to the hierarchy or specify a position in the hierarchy.

4. Automation of Definition

A natural-language interface system will not be practical until a new application can be installed easily. "Easily" means that the end-user organization must be able to create and modify the driving tables for the application relatively quickly without the help of the NLI developer, and must be able to use the NLI without restructuring the database.

Each EUFID application required "handcrafted" tables that were built by the development staff. Each new application was done in less time than the previous one, but still required several staff-months to bring up. Clearly, the goal of facilitating the building of the tables by end users was not met. Computer-assisted tools for defining new applications are a prerequisite for practical NLIs.

B. LANGUAGE PROBLEMS

The basic approach to language analysis in EUFID involves a bottom up parser using a semantic grammar. The symbols of the grammar are concepts underlying lexical items, and the rules of the grammar are based on a case framework. Essentially syntactic information is used only when needed to resolve ambiguity. The language features that this technique has to handle are common to any NLI, and some of the problem areas are described in the following sections.

1. Anaphora and Ellipsis

To support natural interaction it is desirable to allow the use of anaphoric reference and elliptical constructions across sentence sequences, such as "What applicants know Fortran and C?", "Which of them live in California?", "In Nevada?", "How many know Pascal?". One of the biggest problems is to define the scope of the reference in such cases. In the example, it is not clear whether the user wishes to retrieve the set of all applicants who know Pascal or only the subset who live in Nevada.

One solution is to provide commands that allow users to define subsets of the database to which to address questions. This removes the ambiguity and speeds up retrieval time on a large database. However, it moves the NLI interaction toward that of a structured query language, and forces the user to be aware of the level of subset being accessed. It is also difficult to implement because a subset may involve projections and joins to build a new relation containing the subset. The NLI must be able dynamically and temporarily to change the mapping tables to map to this new relation.

2. Intelligent Interaction

One of the EUFID design goals was to respond promptly either with an answer or with a message that the question could not be interpreted. The system handles spelling or typographical errors by interacting with the user to select the correct word. However, when all of the words are recognized but do not connect semantically, it is difficult to identify a single point in analysis which caused the failure.

It is in this area that the absence of a syntactic mechanism for determining well-formedness was most noticeable. There are times when a question has a proper syntactic structure, but contains semantic relationships unrecognizable to the application as in "What is the location of North Hills?". A response of "Location is not defined for North Hills in this application" should be derivable from the recognizable semantic failure. Similarly, it would be useful to have a framework for interpreting partial trees, as in the question "What companies does Mohawk ship to?" where Mohawk is not a recognized word within the application. An appropriate response might be "Companies ship to receiving offices and companies; Mohawk is neither a receiving office nor a company. The names of offices and companies are ...". Interpretation of partial analyses is not possible within the EUFID system; it either succeeds or fails completely.

3. Yes/No Questions

In normal NLI interaction users may wish to ask "yes/no" questions, yet no DBMS has the ability to answer "yes" or "no" explicitly. The EUFID mapper maps a yes/no question into a query which will retrieve some data, such as an "output identifier" or default name for a concept, if the answer is "yes" and no data if the answer if "no". However, the answer may be "no" for several reasons.

For example, a "no" response to the question "Has John Smith been interviewed?" may mean that the database has knowledge about John Smith and about interviews and Smith is not listed as having had an interview*, or the database knows about John Smith and no data about interviews is available. A third possibility could be that the database has information about John Smith and his employment situation (already hired), and the response might include that information, as in "No, but he has already been hired".

4. Conjunctions

The scope of conjunctions is a difficult problem for any parsing or analyzing algorithm. The natural-language use of "and" and "or" does not necessarily correspond to the logical meaning, as in the question "List the applicants who live in California and Arizona.". Multiple conjunctions in a single question can be ambiguous as in "Which minority and female applicants know Fortran and Cobol?". This could be interpreted with logical "and" or with logical "or" as in "Which applicants who are minority or female know either Fortran or Cobol?".

The EUFID mapper will change English "and" to logical "or" when the two phrases within the scope of the conjunction are values for the same field. In the example above, an applicant has only one state of residence.

5. Negation

Negative requests may contain explicit negative words such as "not" and "never" or may contain implicit negatives such as "only", "except" and "other than" [OLNE78]. The interpretation of negatives can be very difficult. For example, "Which companies did not ship any perishable freight in 1976" could mean either "Which (of all the companies) shipped no perishable freight in 1976?" or "Which (of the companies that ship perishable freight) shipped none in 1976?". Moreover, if some companies were only receivers and never shippers it is uncertain whether they should be returned in the answer. It is also difficult to take a complement of a set of data using the many data management systems that do not support set operators between relations.

Questions which require a "yes" or "no" response are difficult to answer because often the "no" is due to a presupposition which is invalid. This is especially true with negation. For example, if the user asks, "Does every company in North Hills except Supreme use NH2?", the answer may be "no" because Supreme is not in North Hills.

The current implementation of EUFID does not allow explicit negation, although some negative concepts are handled such as "What companies ship to companies other than Colonial?". "Other than" is interpreted as the "!=" operator in exactly the same way that "greater than" is interpreted as ">".

C. INTERPRETATION AND DATABASE ISSUES

Many questions make perfect sense semantically but are difficult to map into DBMS queries because of the database structure. The problems become worse when access is through an NLI because of increased expectations on the part of the user and because it may be difficult for a help system adequately to describe the problem to the user who is unaware of the database structure.

1. IL Limitations

The design of the IL is critical. It must be rich enough to support retrieval from all the underlying DBMSs. However, if it contains capabilities that do not exist in a specific DBMS, it is difficult to describe this deficiency to the user.

In APPLICANT, the user cannot get both the major and minor fields of study by asking "List applicants and field of study", because a limitation in the EUFID IL prevents making two joins between education and subject records. This problem was corrected in a subsequent version of IL with the addition of a "range" statement similar to that used by QUEL [STON76].

The current IL does not contain an "EXISTS" or "FAILS" operator which can test for the existence of a record. Such an operator is frequently used to test an interrecord link in a network or hierarchical DBMS. It is needed to express "What problems are unsolved?" to the AIREP application, which requires a test for a database link between a

* There is the important distinction between a "closed world" database in which the assumption is that the database covers the whole world (of the application) and an "open world" database in which it is understood that the database does not represent all there is to the real world of the application. In the open world database, which we encounter most of the time, a response of "not that this database knows of" might be more appropriate.

problem set and a solution set.

2. Mixed Case Values

EUFID allows a value in the database to be upper or lower case and will convert a value in the question either to all upper or all lower case in the IL, or leave it as input by the user. If the database values are mixed case, it is not possible to convert the user's input to a single case. If the user does not enter each letter in the proper case, the value will not match.

3. Granularity Differences

The NLI user is not expected to understand exactly how data is stored, and yet must understand something about the granularity of the data. Time fields often cause problems because time may be given by year or by fractions of a second. Users may make time comparisons that require more granularity than is stored in the database. For example, the user can ask "What incidents were reported at SAC while system release 3.4 was installed?". If incidents were reported by day but system release dates were given by month, the system would return incidents which occurred in the days of the month before the system release was installed.

4. Nested Queries

A very simple question in English can turn into a very complicated request in the query language if it involves retrieval of data which must be used for qualification in another part of the same query. In IL these are called "nested queries". Most often some qualification needs to be done both "inside" and "outside" the clause of the query that does the internal retrieve. For example, the question "What incident at SAC had the longest downtime?" from our AIREP application is expressed in IL as

```
retrieve [INCA.ID]
where (INCA.SITENAME = "SAC")
and (INCA.DNTM =
 {retrieve [ max (INCA.DNTM)]
     where (INCA.SITENAME = "SAC")})
```

The nested part of the query is enclosed in braces. "INCA" is the database name of the active incident records. Notice that removing the "INCA.SITENAME = 'SAC'" clause from either the inner or outer query would result in an incorrect formulation of the question.

A similar example from the METRO application is the question, "What company shipped more than the average amount of light freight in 1980?" which will generate the IL query

```
retrieve [cct.scname]
where (cct.date = 1980)
and (cct.lf >=
     {retrieve [avg (cct.lf)]
      where (cct.date = 1980)})
```

Here, "cct" is the name of the company-to-company transaction relation. "Scname" is the name of a shipping company in this relation. Note again that the qualification on "1980" needs to be done both inside and outside the nested part of the query.

In the query language for INGRES such a request is expressed in a manner very similiar to the IL expressions. For WWDMS a very complex procedure is generated. In all cases, the DBMS needs to answer the inner request and save the result for use in qualifying the outer request. There are many database management systems that cannot handle such questions and these IL statements cannot be translated into the system's query language.

5. Inconsistency In Retrieval

The NLI presents a uniform view of all databases and DBMSs, but it is difficult to truly mask all differences in the behavior of the DBMSs because they do not all process the equivalent query in the same way. For example, when data are retrieved from two relations in a relational database, the two relations must be joined on a common attribute. The answer forms a new relation which may be displayed to the user or stored. Since the join clause acts as qualification, a record (tuple) in either relation which has no corresponding tuple in the other relation does not participate in the result. This is a different concept from the hierarchical and network models where the system retrieves all records from a master record and then retrieves corresponding records from a subfile. This difference can cause anomalies with retrieval. For example, in a pure relational system "List applicants and their interviews" would be treated as "List applicants who have had interviews together with their interview information." A hierarchical or network DBMS would treat it as "List all applicants (whether or not they have been interviewed) plus any interview information that exists.". This second interpretation is more likely to be the correct one.

D. OVERALL NLI DESIGN

There are several problems that affect the selection of applications for the NLI. Some databases and data management systems may not be appropriate targets for natural-language interfaces. Some DBMS functions may be difficult to support. It is important to have a clear understanding of these problems so that the NLI can mediate between the user view, as represented by the natural-language questions, and the underlying database structure.

1. A Design Consideration

For any database there are natural-language questions that cannot be interpreted because the concepts involved lie outside the world of the database. Questions can also involve structural complexity that is not representable in the DBMS query language. A particularly difficult decision in the overall design of an NLI is the issue of where in the chain of events of processing a user's question into a DBMS query to trap these questions and stop processing.

One approach is to decide that if a question is not meaningful to the world of the database it should not be meaningful to the NLI and, therefore, not analyzable on semantic grounds. Another assumes that if the NLI can analyze a question that cannot be asked of the database, it has a much better chance of describing to the user what is wrong with the question and how it might be rephrased to get the desired information.

Codd made good use of the dialogue procedures of the RENDEZVOUS [CODD74] system to avoid questions that the DBMS could not handle, as well as avoiding generation of DBMS queries that did not represent the user's intent. Such a system, however, requires a very large semantic base (much larger than that of the database) in order to make meaningful communication with the user during the dialogue.

2. Class of Database to Support

Some databases are simply not good candidates for an NLI because of characteristics mentioned in previous sections such as many retrieve-only fields, or domains that have a high update rate but cannot be recognized by a pattern.

There are also some structural problems that must be recognized. If the database contains "flat" files about one basic entity, it is reasonably easy to map queries and to explain problems to the user when the mapping cannot be made. However, there can be "reasonable" queries that cannot be answered directly because of the database structure. Hierarchical DBMSs present the most problems with navigation because access must start from the root. For example, if the APPLICANT database were under an hierarchical DBMS, the question "List the specialties for each applicant" could be answered directly but not "What are the specialties?" as there would be no way to get to the specialty records except via particular applicant records.

An array allows more than one instance of a field or set of fields in a single record. There may be arrays of values or even arrays of sets of values in nonrelational databases. When the user retrieves a field that is an array the DBMS requires a subscript into the array. Either the user must specifiy this subscript or the NLI must map to all members of the array with a test for missing data.

3. Class of DBMS to Support

For systems such as EUFID, the database must be organized within a data management system so that the data is structured and individual fields are named. If the data is just text, the EUFID approach cannot be used.

Current NLI systems are designed to be used interactively by a user, which means that the DBMS should also have an interactive query language. However, not all data management systems are interactive. WWDMS [HONE76] has a user query language, but queries are entered into a batch job queue and answers may not return for many minutes. If an NLI front end is to be added to such a DBMS, it must have the capability to generate query programs without any access to the database for parsing or for processing the returned answer.

The query language should support operations equivalent to the relational operations of select, project, and join. Also, the query language should support some arithmetic capability. Most have aggregate functions such as SUM and COUNT. WWDMS does not have an easy-to-use average operation, but it does have a procedural language with arithmetic operators so that EUFID can produce a "query" that procedurally calculates an average.

Basic calculations should be supported such as "age=today-birthdate". It is also desirable to be able to call special functions to do complex calculations

such as navigational calculations required in a naval database.

4. Support for Metadata

Metadata is data about the data in the database. It would be able to tell the user of the METRO application, for example, the kind of information the database has for warehouses and other entities in the application. Such metadata might be extensions of active integrated data dictionaries now available in some DBMSs.

In an application-level system the user should be able to query the metadata to learn about the structure of the database. A different mode, such as the menus used by the EUFID help system, could be used to access metadata, or English language questions to both meta information and the database could be supported.

5. Updates

Some potential users would like a natural-language interface to include the capability to update the database. Currently, updating through any high level view of the database should be avoided, especially when the view contains joins or derived data, because of the risk of inadvertently entering incorrectly-interpreted data.

V SUMMARY AND CONCLUSIONS

For many years, researchers have been attempting to build robust systems for natural-language access to databases. It is not clear that such a system exists for general use [OSI79]. There are problems that need to be solved on both the front end, the parsing of the English question, and the back end, the translation of the question into a data management system query. It is important to understand the types of requests, types of functions, and types of databases that can be supported by a specific NLI.

Some general guidelines that can be applied to the selection of applications for current NLI front ends are suggested below:

1. the underlying DBMS should have an interactive query language,
2. the DMS view should be relational or at least support multiple access paths,
3. the database should not contain arrays either of values or of structures,
4. the input must be controlled to standardize values,
5. there should be few fields that have values that change rapidly, cannot be recognized by a pattern, and that must be used in qualification,
6. the users of the NLI should have a common use for the data and a common view of the data, and
7. there must be some user who understands the questions that will be asked and is available to work with the developers of the NLI.

We believe that current system development is limited by the need for good semantic modelling techniques and the length of time needed to build the knowledge base required to interface with a new application. When the knowledge base for the NLI is developed, the database as well as sample input must be considered in the design. Parsing of questions to a database cannot be divorced from the database contents since semantic interpretation can only be determined in the context of that database. On the other hand, a robust system cannot be developed by considering only database structure and content, because the range of the questions allowed would not accurately reflect the user view of the application and also would not account for all the information that is inferred at some level.

ACKNOWLEDGEMENTS

We would like to acknowledge the many people who have contributed to EUFID development: David Brill, Marilyn Crilley, Dolores Dawson, LeRoy Gates, Iris Kameny, Philip Klahr, Antonio Leal, Charlotte Linde, Eric Lund, Filip Machi, Kenneth Miller, Eileen Lepoff, Beatrice Oshika, Roberta Peeler, Douglas Pintar, Arie Shoshani, Martin Vago, and Jim Weiner.

REFERENCES

[AHO72] Aho, A. V. and J. D. Ullman, "The Theory of Parsing, Translation, and Compiling", Vol. I: Parsing, Prentice-Hall, 1972, pp. 314-230.

[BURG80] Burger, J. F., "Semantic Database Mapping in EUFID", Proceedings of the 1980 ACM/SIGMOD Conference, Santa Monica, Calif., May 14-16, 1980.

[BURG82] Burger, J. F. and Marjorie Templeton, "Recommendations for an

Internal Input Language for the Knowledge-Based System", System Development Corporation internal paper N-(L)-24890/021/00, January 5, 1982.

[CODD74] Codd, E. F., "Seven Steps to Rendezvous with the Casual User", Proc. IFIP TC-2 Working Conference on Database Management Systems, Cargese, Corsica, April 1-5, 1974, in J. W. Kimbie and K. I. Koffeman (Eds.), "Data Base Management" North-Holland, 1974.

[CULL80] Cullinane Corporation, "IQS Summary Description", May 1980.

[DATE77] Date, C. J., "An Introduction to Database Systems", second edition, Addison-Wesley Publishing, Menlo Park, CA, 1977.

[EDP82] "Query Systems for End Users", EDP Analyzer, Vol. 20, No. 9, September, 1982.

[HARR78] Harris, L. R., "The ROBOT System: Natural Language Processing Applied to Data Base Query", Proceedings ACM 78 Annual Conference, 1978.

[HEND77] Hendrix, G. G., E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a Natural Language Interface to Complex Data" SRI Report 78-305, August 1977.

[HONE76] Honeywell, WWMCCS: World Wide Data Management System User's Guide, Honeywell DB97 Rev.3, April 1976.

[KELL71] Kellogg, C. H., J. F. Burger, T. Diller, and K. Fogt, "The CONVERSE Natural Language Data management System: Current Status and Plans", Proceedings of the ACM Symposium on Information Storage and Retrieval, University of Maryland, College Park, MD, 1971, pp. 33-46.

[MYLO76] Mylopoulos, J., A. Borgida, P. Cohen, N. Roussopoulos, J. Tsotsos, and H. Wong, "TORUS: A Step Towards Bridging the Gap between Data Bases and the Casual User", in Information Systems Volume 2 1976, Pergamon Press, pp 49-64.

[OLNE78] Olney, John, "Enabling EUFID to Handle Negative Expressions", SDC SP-3996, August 1978.

[OSI79] Operating Systems, Inc., "An Assessment of Natural Language Interfaces for Command and Control Database Query", Logicon/OSI Division report for WWMCCS System Engineering, OSI Report R79-026, 29 June 1979.

[SCHA77] Scha, R. J. H., "Phillips Question-Answering System PHLIQA1", in SIGART Newsletter Number 61, February 1977, Association for Computing machinery, New York.

[SIMM65] Simmons, R. F., "Answering English Questions by Computer -- a Survey", Comm. ACM 8,1, January 1965, 53-70.

[STON76] Stonebraker, M., et. al., "The Design and Implementation of INGRES", Electronics Research Laboratory, College of Engineering, University of California at Berkeley, Memorandum No. ERL-M577, 27 January 1976.

[TEMP79] Templeton, M. P., "EUFID: A Friendly and Flexible Frontend for Data Management Systems", Proceedings of the 1979 National Conference of the Association for Computational Linguistics, August, 1979.

[TEMP80] Templeton, M. P., "A Natural Language User Interface", Proceedings of "Pathways to System Integrity", Washington D.C. Chapter of ACM, 1980.

[THOM69], Thompson, F. B., P. C. Lockemann, B. H. Dostert, and R. Deverill, "REL: A Rapidly Extensible Language System", in Proceedings of the 24th ACM National Conference, Association for Computing machinery, New York, 1969, pp 399-417.

[WALT77] Waltz, D. L., "Natural Language Interfaces", in SIGART Newsletter Number 61, February 1977, Association for Computing machinery, New York.

[WALT78] Waltz, D. L., "An English language Question Answering System for a Large Relational Database", Communications of the ACM 21, 7(July 1978), pp 526-539.

[WOOD72] Woods, W. A., R. M. Kaplan, B. Nash-Webber, The Lunar Sciences Natural Language Information System Final Report, Report number 2378, Bolt, Beranek, and Newman, Inc., Cambridge, MA, 15 June 1972.