A portable natural language interface*

by BIPIN C. DESAI, JOHN McMANUS, and PHILIP J. VINCENT
Concordia University
Montreal, Quebec

ABSTRACT

A natural language interface allows database system users to input a query in a natural language such as English or French rather than in a formal query language. Such interfaces could also provide for natural language updates, but this paper deals only with queries.

The goal of a natural language interface is to permit users to express their information needs in their own language and in conceptual terms particular to their understanding of the database application domain. Users also are freed from knowing about database management systems (DBMS), data models and database schemas.

Allowing a user to access a database using natural language shifts onto the computer system (the interface and the DBMS) the burden of mediating between two views of data: the way in which the data is stored (the database view) and the way in which an end user thinks about it (the user’s view). A DBMS, particularly a relational one, accomplishes part of this task. The interface must reconcile the user’s view with the DBMS’ view.

To achieve such data independence, the interface must incorporate a considerable amount of knowledge including knowledge about natural language, the domain database application, and DBMSs and their query languages.

*This work was supported in part by a contract from the Canadian Workplace Automation Research Center of the Department of Communication, Government of Canada.
INTRODUCTION

A natural language interface is portable if it can be transferred with minimum effort from the database for which it was designed to a new database. The degree of portability is reflected in the amount of effort required to transfer the interface. If an interface is handtailored to a particular database, major reprogramming is required to convert it to a new database. In the extreme case, the effort will be equal to the task of programming the interface to the original database. On the other hand, an interface that requires no effort to transfer is beyond the capabilities of current research. Such an interface would have to learn by itself the characteristics of a new database and adapt its linguistic and computational abilities accordingly.

The general design structure for an interface that allows a moderate degree of portability is presented in the next section of this paper. The goal of the design structure is to minimize the amount of reprogramming necessary for three types of database transfer: (1) change of DBMS, (2) change of application domain, and (3) conceptual reorganization of the database.

The key to achieving some degree of portability is modularity. Early natural language systems were handtailored for particular applications. In their data structures and procedures, they intermixed knowledge about language with knowledge about the domain application. They also conflated user request with how to obtain the information requested. The early systems were inherently not portable because they were not modular.

Natural language interfaces can be modularized in three dimensions. The first dimension keeps distinct the three main types of knowledge required; that is, knowledge about language, the application domain, and databases. The second separates procedural knowledge from declarative knowledge. The ability to parse an English sentence is inherently procedural whereas the vocabulary a parser accepts is naturally declarative. A parser and its lexicon should therefore be kept separate. The third dimension distinguishes between general knowledge and domain-specific knowledge. This dimension is crucial for portability. As much of the interface as possible should be designed using only general knowledge. A transfer to a different DBMS or application thus would not require changing modules that incorporate general knowledge only.

GENERAL DESIGN STRUCTURE

Figure 1 graphically illustrates a possible design of a portable natural language interface. The design includes three procedural modules and three declarative modules. The procedural modules are the parser, the semantic analyzer, and the query generator. The declarative modules are the lexicon, the database mapping, and the semantic model.
semantic model, and the DB mapping. The DBMS and data
base also are shown in Figure 1, but they are not part of the
interface.

A natural language query is input to the interface and the
interface's final output is a formal DBMS query. The pro-
cedural modules, each using knowledge in the declarative
modules, successively transform the natural language query.
Using the lexicon and the semantic model, the parser outputs
a parse tree representation of the query. Using the semantic
model, the semantic analyzer transforms the parse tree into a
semantic representation of the query. Finally, using informa-
tion in the DB mapping, the query generator in turn trans-
forms this semantic representation into a formal DBMS
query.

The procedural modules are written as generally as possi-
ble, incorporating domain-independent knowledge only. The
domain-specific knowledge is isolated in the declarative mod-
ules. The lexicon contains the words users may use in their
queries. The semantic model is a formal representation of the
application domain. It should not be confused with the DBMS
conceptual schema; rather, it corresponds to a DBMS enter-
prise schema. The DB mapping contains the mapping be-
tween the concepts of the semantic model and the correspond-
ing structures of the database.

Following is a description of the portability achieved by the
design. If the application domain is changed, only the lexicon,
the semantic model, and the DB mapping have to be modi-
fied. If the DBMS is changed, only the query generator is
modified. If the database is reorganized without semantic
change, only the DB mapping is changed. The parser and the
semantic analyzer are perfectly portable; immune to any do-
main, DBMS, or database transfer.

THE IMPORTANCE OF THE SEMANTIC MODEL

Language can be described as the encoding of thought for the
purpose of communication. Communication is between a
sender and a receiver. The sender formulates a thought, en-
codes it into language, and sends it to the receiver. The
receiver receives the sender's language and decodes it, at-
tempting to recreate the sender’s original thought. An act of
communication is deemed more or less successful according to
how perfectly the original and recreated thoughts match. A
match is possible only if a model of mutual comprehension
exists. Culture provides such a model for normal human com-
munication.

The basic process of communication occurs analogously in
a natural language database system. A user formulates his or
her information need according to his or her understanding of
the application domain. Then the user encodes this need into
language and sends it to the interface. The interface (i.e., the
parser and semantic analyzer modules) decodes the language
and recreates the user’s original thought. A model of
mutual comprehension is necessary, and is provided by the
semantic model which is the interface’s representation of the
application domain and, to complete the analogy, the com-
mon culture of the user and the interface.

The semantic model is therefore crucial to the interface.
Each natural language question is translated, or decoded, into
a query on and in the terms of the semantic model. Its exact
form depends upon the formalism of the semantic model.
There is no general consensus about which formalism should
be used in implementing the semantic model. Examples in-
clude object-based data models and artificial intelligence
knowledge representation schemes. The other declarative
knowledge in the system is defined with reference to the se-
matic model. Each word in the lexicon is associated with a
particular concept in the semantic model. The DB mapping
relates the concepts of the semantic model to the database
structures.

Is the semantic model necessary? In other words: Could
there be a mapping directly between the words of the lexicon
and the database structures? The argument against this direct
mapping is that a database schema does not adequately repre-
sent the domain semantics and fails to provide a model of
mutual comprehension. Without a separate semantic model,
the burden of handling the domain-specific semantics de-
volves onto the semantic analysis procedure. This procedure,
as outlined in the previous section, is meant to be domain
independent and portable. Eschewing a semantic model
therefore results in a handtailored non-portable interface.

THE PROBLEM OF AMBIGUITY

A natural language interface has to cope with the inherent
ambiguity of natural language. Ambiguity serves a useful pur-
pose in human communication by reducing the verbiage nec-
essary to express an idea. The ambiguity is resolved by context
or by interaction.

There are two main types of ambiguity: syntactic and se-
matic. Syntactic ambiguities arise when there are multiple
valid parses of the same query. For example, “Which course
has the largest enrollment of students in computer science?”
can be parsed with “in computer science” modifying either
the course or the students, with different interpretations re-
sulting. Semantic ambiguities occur when the parsed constitu-
ents have several possible meanings. For example, “Where is
the Netherlands?” may request the position of a ship or a
country, though syntactically it is unambiguous.

Many ambiguities can be resolved with recourse to the se-
matic model. However, complete automatic resolution of all
ambiguities is not possible. The system must echo back para-
phrases of the possible meanings of the query and thereby
allow a user to choose the intended interpretation.

PARSING

To parse a query, it is necessary to use a grammar that de-
scribes the structure of strings accepted by the interface.
Given such a grammar, the parser assigns a structure, or parse
tree, to each grammatical query it processes. The grammar,
which should allow a user wide linguistic variation, is incorpo-
rated within the parser module. The domain-specific knowl-
edge the parser requires is in the lexicon or dictionary.

The lexicon contains all the words accepted by the parser.
Associated with each word is its syntactic category and its
association to the semantic model. The entry for red would
include adjective as its syntactic category and instance-of-color as its conceptual association. The entry for part would indicate that it is a noun and that it is associated with the entity type “part” assuming an entity-based semantic model. The entry for who would indicate that it is an interrogative personal pronoun and indicate the set of entity types that it might refer to. The entry for supply would have verb as its syntactic category and the relationship or aggregation “supply” as its associated concept.

Assuming such a lexicon and the simplified grammar of Figure 2, a parse of: “Who supplies red parts?” would produce the tree of Figure 3.

In addition to the parse tree in Figure 3, the parse would pass to the semantic analyzer: (1) pointers to the appropriate concepts, (2) the morphological information that the input string contains the third person singular form of supply and the plural form of part, and (3) the further syntactic information that who is the subject and part is the object of supply. The semantic analyzer would access the semantic model and disambiguate who by checking which entities can serve in the role of subject to the concept “supply.” The morphological information of third person singular indicates that who refers to the specific entities. If entity types were desired, the phrase would have been: “Who supplies red parts?”

This description represents one extreme of the use of semantics in parsing: a completely syntactic parse followed by semantic analysis. However, pure syntactic parsing can cause problems. Natural language viewed syntactically has many ambiguities. The major type of syntactic ambiguity arises from the fact that modifying phrases and clauses can be physically separated from the constituents they modify. For example, the question “Who drove down the street in the car?” had syntactically valid readings of: “the street is in the car.”

A simple semantic intervention would rule out this possible parsing. Indeed, when the length of the query and the number of modifiers increases the number of parses grows exponentially.

The other extreme of the use of semantics in parsing is a semantic grammar in which semantic and syntactic categories are intermixed in the constituent structures the grammar uses to describe the language. The problem with this approach is that it introduces domain-specific semantics within the parser and makes the interface less portable. It also makes it much more difficult to provide wide linguistic coverage. In a syntactic parser, the passive form of all verbs can be allowed with the introduction of one general transformation rule. In a semantic grammar, on the other hand, the passive transformation would have to be added for every verb.

There seems to be a general consensus that the best approach is a syntactically based parser with general semantic checks or routines. The syntactic base permits portability and wide linguistic coverage. With semantic intervention many ambiguities can be resolved early in the parsing process. Syntactic and semantic disambiguation proceed together. Modifier placement is determined by seeing which concepts are linked in the semantic model. Semantic or lexical ambiguity caused by words having multiple meanings is resolved by maintaining a set of candidate meanings for each word during parsing and restricting the set as the syntactic structure provides a context. Often not all ambiguities will be resolved and the interface should present the multiple interpretations to the user as described in the previous section.

Semantic checking may also reveal meaningless queries. It may not be possible to associate a modifying phrase with any head noun phrase. For example, “Who supplies red projects?” is a meaningless query if projects do not have the attribute color. Similarly, the restriction of candidate meanings for words can lead to recognizing a meaningless query when the set of candidate meanings becomes null. This would occur in the following question: “Which projects supply red parts?”

Adding semantics to the parsing process allows the output of the parse to be more than just a syntactic parse tree. An incipient semantic representation can be created. It would be based on the association of concepts implied by modifying phrases and clauses and by verb-noun phrase relationships.

THE LEXICON

Kaplan defines three types of lexical entries: general, structural, and volatile. General entries are those that apply in practically any domain. These include closed classes of words such as auxiliary verbs, prepositions, and conjunctions.
eral entries constitute a permanent part of the interface. Structural entries are those that make reference to aspects of the semantic model and must be changed for each new application. These are the nouns, verbs, adjectives and their many synonyms that typically are used in a particular domain. The impossibility of predicting during system design all the various synonyms that may be used argues for an interactive synonym generator that enables the user himself to extend the vocabulary accepted by the interface. Volatile entries are those which refer to specific values in the database. They also are the most problematic. Keeping each value in the lexicon is too expensive because doing so essentially entails a duplication of the database. Searching the database is not a feasible solution because the interface may not know where the unknown lexical item is located in the database and because possible but not actual values can be used. One proposed solution is to represent values from limited domains or those used frequently and to disallow the user of other values in queries. However, a better solution is for the interface to ask a user for clarification when it is confronted with unknown lexical items.

SEMANTIC ANALYSIS

The semantic model encodes the user’s view of the domain. In its simplest form, a semantic model must represent a user’s knowledge of the objects in the domain, the properties of those objects, and the relationships among them. Various representation formalisms such as semantic data models, semantic networks, and case grammar frames are used. The common point among them is that they are richer semantically than database schemas and are able to provide a data independence that DB schemas alone cannot provide.

The semantic analysis must recognize the propositional content in a user’s query. The basic proposition in a query is determined by the main verb and the noun phrases to which it relates. This verb-noun phrase relationship is similar to the way a predicate relates its arguments and the way aggregation relates entities and attributes. The main verb of a query can thus be seen as defining its main predicate. Additional propositions or predicates are defined by modifying phrases and clauses. These predicates are, in effect, nested within the main predicate. Adjectival and prepositional phrases have very simple predicate realizations. For example, green becomes color (x.green) and in London becomes in(x, London). Clausal modifiers add another level to the nesting because the verbs within the clauses also represent predicates. Therefore, the semantic analyzer must unnest all the predicates and relate the resulting separated predicates through common variables.

This propositional content of the query must be transformed into a conceptual calculus form analogous to relational calculus. That is, the propositional content must be transformed into a declarative description of the information desired in terms of the users’ concepts rather than the relations of a relational database.

QUERY GENERATION

The task of the query generator is to take a conceptual calculus query and transform it into a query of the underlying DBMS. First it must map the concepts of the semantic model into the underlying database structures and then translate the query. If the back-end DBMS is relational, mapping and translating may be relatively straightforward. However, if the DBMS is a hierarchical or network system, the task is complicated by an extra burden of having to physically navigate through the files and records of the database.

CONCLUSIONS

The basic framework for a portable natural interface to a DBMS is presented in this paper. The purpose of the interface is to make a database more user-friendly. To achieve portability, modularity and separation of general and domain-specific knowledge are necessary. The procedural modules are the parser, the semantic analyzer, and the query generator. The declarative modules, in which all the domain-specific knowledge is isolated, are the lexicon, the semantic model, and DB mapping. The semantic model, more semantically expressive than a DB schema, is necessary to provide portability and data independence.

REFERENCES


